Computer Security Symposium 2018
22 - 25 October 2018

AlkanetにおけるMeltdown対策済Windowsのシステムコールトレース手法

山下 雄也[1,a)]　鄭 俊俊[1]　齋藤 彰一[2]　瀧本 栄二[1]　毛利 公一[1]

概要：64bit マルウェアの出現に伴って 64bit マルウェアの解析環境が求められている．我々は仮想化技術を用い，Windows 10 上で 64bit マルウェアの解析を可能とするシステムコールトレーサ Alkanet 10 を開発している．しかし，Meltdown 脆弱性の発見に伴って Windows で対策が取られたため，Alkanet は従来のトレース方式での動作ができなくなった．本論文では，Windows の Meltdown 対策機能である KVAS についての調査で得られた知見について述べる．具体的には，KVAS の各処理について，内部実装を交えて述べる．さらに，それに基づいて改良した Alkanet のトレース方式について述べる．

キーワード：Meltdown，システムコールトレース，仮想化技術

Implementation of System Call Tracing for Windows Having Countermeasures Against Meltdown on Alkanet

Yuya Yamashita[1,a)]　Junjun Zheng[1]　Shoichi Saito[2]　Eiji Takimoto[1]　Koichi Mouri[1]

Abstract: Along with the advent of 64-bit malware, analysis environment of 64-bit malware is now required. We are developing Alkanet 10, which is a system call tracer for 64-bit malware analysis on Windows 10 using virtualization technology. However, with the discovery of the Meltdown vulnerability, countermeasures were taken in Windows, so Alkanet can no longer work with its traditional tracing methods. In this paper, we describe the findings obtained from the survey on KVAS which is a Meltdown countermeasure function of Windows. Specifically, we describe each process of KVAS with internal implementation. Furthermore, we describe Alkanet's tracing methods improved based on it.

Keywords: Meltdown, system call trace, virtualization technology

1. はじめに

近年，オペレーティングシステムの 64bit 環境の普及に伴い，マルウェアの 64bit 化が進んでいる [1]．実際に，64bit マルウェアによる被害も発生しており，今後被害はさらに増加していくと考えられる．マルウェアへの対策のためには，その挙動を明らかにする必要があり，64bit マルウェアの解析環境が求められている．マルウェアの解析手法は，マルウェアを実際に動作させて挙動を追跡する動的解析とマルウェアのコードを逆アセンブルして挙動を解析する静的解析の 2 つに大別される．我々は，動的解析に着目し，Windows 上で動作するマルウェアの解析を可能とするシステムコールトレーサ Alkanet をこれまで開発してきた [2]．Alkanet は，仮想計算機モニタ (VMM; Virtual Machine Monitor) である BitVisor [3] の拡張機能として動作し，ゲスト OS である Windows XP 上のプロセスが発行したシステムコールをスレッド単位でトレースできる．また，我々は，解析対象となる Windows のバージョンを XP から 10 にするための Alkanet の拡張も行っている [4][5]．以下，本論文では，この Windows 10 x64 を対象

[1] 立命館大学
Ritsumeikan University, Kusatsu, Shiga 525-8577, Japan
[2] 名古屋工業大学
Nagoya Institute of Technology, Nagoya, Aichi 466-8555, Japan
[a)] yyamashita@asl.cs.ritsumei.ac.jp

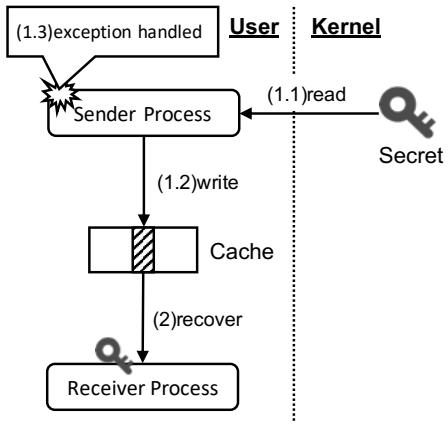

図 1　Meltdown 攻撃の全体像

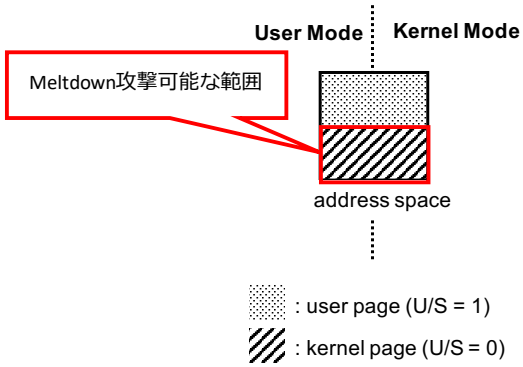

図 2　従来の OS におけるアドレス空間の構成

とする Alkanet を Alkanet 10 と呼称する．

2018 年 1 月，プロセッサの脆弱性として CVE-2017-5754 (Meltdown) が報告された [6]．この脆弱性は，アウトオブオーダ実行を行うプロセッサ上において，カーネルメモリの値をユーザ権限で不正に読み出せる，というものである．本脆弱性の発見に伴い，Microsoft は Meltdown 対策技術として Kernel Virtual Address Shadow (KVAS) を開発し，Windows に実装した [7]．一方，Alkanet 10 はこの影響を受け，従来のトレース方式での動作ができなくなった．KVAS を実装している Windows 10 を対象に Alkanet 10 を動作させるためには，KVAS の内部実装を調査し，それを基に Alkanet 10 の処理を適切に変更する必要がある．本論文では，KVAS の内部実装についての調査で得られた知見とそれを基に改良した Alkanet 10 のトレース方式について述べる．

本論文では，2 章で Meltdown の概要について述べ，3 章で KVAS とトレース方式の改良について述べる．4 章でシステムコールトレースの動作確認について述べ，最後に 5 章でまとめる．

2. Meltdown の概要

CVE-2017-5754 (Meltdown) は，Lipp らによって報告されたプロセッサの脆弱性である [6]．この脆弱性を悪用することで，アウトオブオーダ実行を行うプロセッサ上において，カーネルメモリの値をユーザ権限で不正に読み出すことができる．Lipp らは以下の条件を満たす環境上で，Meltdown 攻撃の成立が可能であると述べている．

- 攻撃者は攻撃対象コンピュータ上で任意のユーザプログラムを実行できる．
- 攻撃者は攻撃対象コンピュータに対する物理アクセスを行えない．
- 攻撃対象コンピュータは ASLR，KASLR，SMAP，SMEP，NX，PXN によって保護されている．
- 攻撃対象コンピュータ上の OS はバグを持たない．

Meltdown 攻撃の全体像を図 1 に示す．Meltdown 攻撃は，以下の 2 つの手順から構成される．

(1) 読出し対象となるカーネルアドレス空間内の情報 (Secret) を間接的な形でキャッシュに乗せる (図 1 中の Sender Process が行う処理)．

(2) キャッシュに対してサイドチャネル攻撃を行い，Secret を復元する (図 1 中の Receiver Process が行う処理)．

Meltdown 攻撃では，まずユーザ権限で Secret を読み出す (図 1(1.1))．Secret はカーネルアドレス空間内の情報であるため，この読出しはアクセス違反となり，例外が発生する．しかし，アウトオブオーダ実行を行うプロセッサ上においては，例外がハンドルされる (図 1(1.3)) 前に後続の命令が実行される．具体的には，読み出した Secret がデータバスに乗ってから例外がハンドルされるまでの間にタイムラグが存在し，この間に後続命令が実行される．これら後続命令では，Secret の読出し結果を用いた処理が可能である．また，後続命令による実行結果は，例外がハンドルされるタイミングでロールバックされるが，キャッシュの状態まではロールバックされない．Meltdown 攻撃はこれら 2 点を悪用し，後続命令で Secret を間接的な形でキャッシュに乗せる (図 1(1.2))．

次に，キャッシュに対してサイドチャネル攻撃を行い，Secret を復元する (図 1(2))．キャッシュに対するサイドチャネル攻撃手法は，様々なものが報告されているが，文献 [6] では Flush+Reload[8] と呼ばれる手法が用いられている．

3. KVAS とトレース方式の改良

3.1 従来の OS におけるアドレス空間の構成

従来の OS では，図 2 のようにユーザモードとカーネルモードで同一のアドレス空間を使用していた．このアドレス空間には，全てのユーザページとカーネルページがマップされるが，各ページはページテーブルエントリの属性フラグによって保護できる．たとえば，カーネルページに対応するページテーブルエントリの U/S フラグを 0 にすることで，ユーザ権限による不正なアクセスからカーネルページを保護していた．

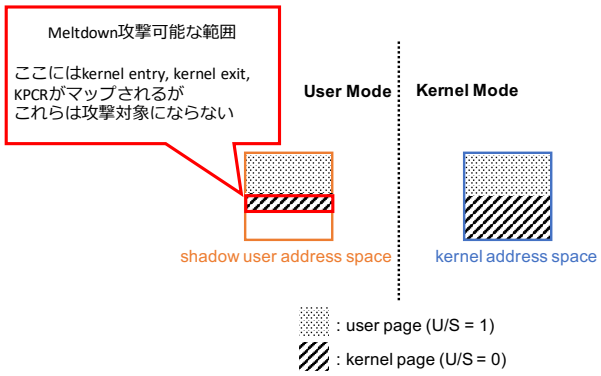

図 3　KVAS 有効時におけるアドレス空間の構成

しかし，Meltdown 脆弱性の発見に伴い，この保護機構を破れることが明らかとなった．当該脆弱性を悪用することで，図 2 中の全てのカーネルメモリの値をユーザ権限で不正に読み出すことができる．これに対し，Microsoft は Meltdown 対策技術である KVAS を開発し，Windows に実装した．

3.2　KVAS の概要

Meltdown 攻撃は 2 章で述べたように，例外がハンドルされる前に Secret がデータバスに乗ることで成立する．そのため，Secret が仮想アドレス空間にマップされていなければ，アドレス変換に失敗するために Secret がデータバスに乗らず，Meltdown 攻撃は成立しない．KVAS はこのアプローチを用いて，Meltdown 攻撃を回避する．

3.1 節で述べたように，従来はプロセス毎に 1 つのアドレス空間を割り当てていたが，KVAS はプロセス毎に 2 つのアドレス空間を割り当てる (図 3 参照)．具体的には，プロセッサの動作モード毎に以下の 2 つのアドレス空間を割り当てる．

- shadow user address space
- kernel address space

shadow user address space は，ユーザモードで動作中のプロセスが使用するアドレス空間である．ユーザコードはユーザページのみを必要とするため，このアドレス空間には全てのユーザページと KVAS の動作に必要となる最小限のカーネルページ (kernel entry，kernel exit，KPCR 構造体) のみをマップする．kernel entry，kernel exit，KPCR 構造体については後述するが，これらの中には Meltdown 攻撃の対象となりうるデータは存在しない．Meltdown 攻撃を行う Sender Process は，ユーザモードで動作するプロセスであるため，shadow user address space を使用する．一方，当該アドレス空間には，Meltdown 攻撃の対象となりうるデータがマップされていない．したがって，Sender Process が攻撃対象データをキャッシュに乗せることは不可能となり，Meltdown 攻撃を回避できる．

kernel address space は，カーネルモードで動作中のプロセスが使用するアドレス空間である．カーネルコードはカーネルページとユーザページを必要とするため，このアドレス空間には全てのカーネルページとユーザページをマップする．Meltdown 攻撃を行うプロセスは，ユーザモードで動作するため，kernel address space に全てのカーネルページをマップしても問題とならない．

KVAS は，shadow user address space と kernel address space 間の切替えを行うために，以下の 2 つのカーネル関数群を使用する．

- kernel entry
- kernel exit

kernel entry は，shadow user address space から kernel address space への切替えを行う関数群である．一方，kernel exit は，kernel address space から shadow user address space への切替えを行う関数群である．

KVAS のアドレス空間切替え処理の流れを図 4 に示す．KVAS は以下の手順でアドレス空間の切替えを行う．

(1) システムコールの発行や例外の発生といったモード切替えを伴うイベントが発生し，カーネルモードに切り替わる．カーネルモードへの切替え後，kernel entry に属するいずれかの関数が呼び出される．どの関数が呼び出されるかは，発生したイベントの種類によって決定される．

(2) kernel entry が shadow user address space から kernel address space への切替えを行う．kernel address space に対応するページテーブルのアドレスは，KPCR 構造体のメンバから取得する．

(3) kernel entry がメイン処理を行うカーネル関数を呼び出す．

(4) メイン処理を行うカーネル関数は，自身の処理完了後，kernel exit に属するいずれかの関数を呼び出す．ここで呼び出される関数は，様々な要因によって決定される．

(5) kernel exit が kernel address space から shadow user address space への切替えを行う．shadow user address space に対応するページテーブルのアドレスは，KPCR 構造体のメンバから取得する．

(6) kernel exit がユーザモードへの切替えを行い，ユーザコードに制御を戻す．

上記手順から分かるように，kernel entry と kernel exit の実行，および KPCR 構造体へのアクセスは，shadow user address space を使用している間にも発生する (手順 (1)，(2)，(5)，(6))．そのため，kernel entry，kernel exit，KPCR 構造体は shadow user address space にマップする必要がある．

3.3　システムコールトレース方式の改良

Alkanet 10 は図 5 のように，システムコールの入口関

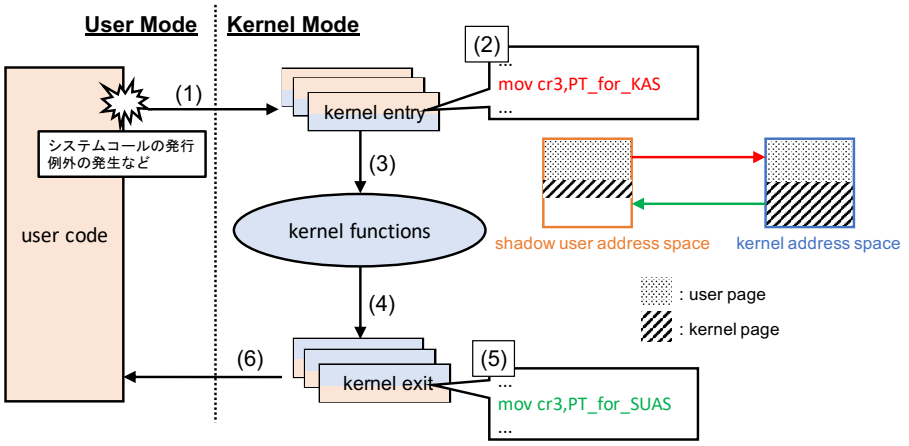

図 4　KVAS のアドレス空間切替え処理の流れ

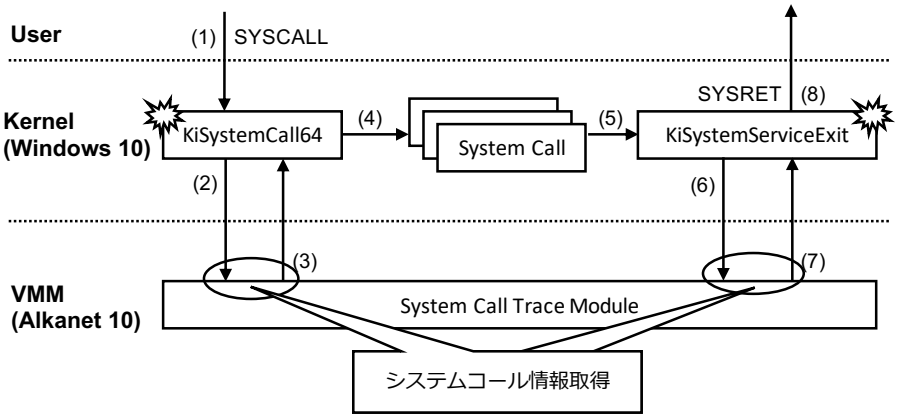

図 5　Alkanet 10 におけるシステムコールトレース処理の流れ

数である KiSystemCall64 と出口関数である KiSystemServiceExit にハードウェアブレークポイントを設定することで，システムコールフックを実現している．システムコールフック後は，各種レジスタやゲストメモリを読み出すことでシステムコールに関する情報を取得し，システムコールトレースログを作成する．

しかし，KVAS が実装された影響を受け，Alkanet 10 は上記トレース方式での動作ができなくなった．この原因は以下の 2 点である．

原因 (1)　システムコールの入口関数と出口関数が変更されたこと

原因 (2)　システムコールの入口関数と出口関数にアドレス空間切替え処理が追加されたこと

まず，原因 (1) とその解決策について述べる．システムコールの呼出しは，カーネルモードへの切替えを伴うイベントであるため，KVAS の動作対象である．そのため，システムコールの入口関数が kernel entry に属する関数に，出口関数が kernel exit に属する関数に変更された．具体的には，入口関数が KiSystemCall64 から KiSystemCall64Shadow に，出口関数が KiSystemServiceExit から KiKernelSysretExit に変更された．したがって，従来のブレークポイント設定では，システムコールフックを実現できなくなった．以上より，原因 (1) を解決するためには，Alkanet 10 のブレークポイント設定位置を KiSystemCall64Shadow と KiKernelSysretExit に変更する必要がある．

次に，原因 (2) とその解決策について述べる．KiSystemCall64Shadow と KiKernelSysretExit は，それぞれ kernel entry と kernel exit であるため，アドレス空間の切替え処理を有している．そのため，どのタイミングでフックを行うかによって，Alkanet 10 に制御が移った際の現行アドレス空間が変化する．一方，Alkanet 10 は，システムコールに関する情報をゲストメモリから取得するが，これら情報の多くは shadow user address space にマップされていない．したがって，現行アドレス空間が shadow user address space であるタイミングで Alkanet 10 に制御が移ると，Alkanet 10 は必要な情報を取得できず，システムコールトレースログの作成に失敗する．以上より，原因 (2) を解決するためには，現行アドレス空間が kernel address space であるタイミングで Alkanet 10 に制御が移るように，ブレークポイントを設定する必要がある．

表 1　Alkanet 10 のトレース対象システムコール

カテゴリ	システムコール名
File	NtCreateFile, NtOpenFile, NtDeleteFile, NtReadFile, NtWriteFile, NtSetInformationFile
Registry	NtCreateKey, NtOpenKey, NtOpenKeyEx, NtQueryKey, NtSetInformationKey, NtDeleteKey, NtSetValueKey, NtQueryValueKey, NtDeleteValueKey
Section	NtCreateSection, NtOpenSection, NtMapViewOfSection
Virtual Memory	NtWriteVirtualMemory, NtReadVirtualMemory, NtAllocateVirtualMemory, NtProtectVirtualMemory
Network	NtDeviceIoControlFile
Process	NtCreateProcess, NtCreateProcessEx, NtOpenProcess, NtCreateUserProcess, NtTerminalProcess
Thread	NtCreateThread, NtCreateThreadEx, NtOpenThread, NtSuspendThread, NtTerminateThread, NtResumeThread, NtGetContextThread, NtSetContextThread, NtQueueApcThread
Driver	NtLoadDriver, NtUnloadDriver
Time	NtQueryPerformanceCounter
Mutant	NtCreateMutant, NtOpenMutant
Sleep	NtDelayExecution

表 2　動作確認に使用した環境

項目	値
VMM	Alkanet 10
ゲスト OS	Windows 10 Pro 1709 (OS build 16299.309)
プロセッサ	Intel Core i5-6600

4. システムコールトレースの動作確認

4.1 目的と方法

3 章で述べた改良を行った Alkanet 10 が，KVAS 実装済み Windows 10 のシステムコールを正しくトレースできることを確認する．Alkanet 10 は表 1 に示すシステムコールをトレース対象としているが，本動作確認ではこのうち NtCreateFile，NtWriteFile，NtReadFile について確認を行う．その他のトレース対象システムコールについては，今後動作確認を行う予定である．

本動作確認は表 2 に示す環境を用い，以下の手順で行った．

(1) テストプログラムを用意する．当該テストプログラムは，CreateFile，WriteFile，CloseHandle を用いたファイル書込みと CreateFile，ReadFile，CloseHandle を用いたファイル読出しを行う．書込み対象ファイルおよび読出し対象ファイルは，いずれも”C:\Users\yyamashita\file.txt”である．
(2) Alkanet 10 と Windows 10 を起動し，システムコールトレースを開始する．
(3) テストプログラムを実行する．
(4) 取得できたログから，期待通りのシステムコールトレース結果が得られているか確認する．テストプログラムは CreateFile → WriteFile → CreateFile → ReadFile の順に Windows API を呼び出すため，期待されるシステムコール列は NtCreateFile → NtWriteFile → NtCreateFile → NtReadFile である．

4.2 結果と考察

動作確認の結果を図 6 と図 7 に示す． 動作確認実行時のテストプログラムのプロセス ID とスレッド ID は，それぞれ 0x1c40，0x1c1c であった．

図 6 では，no.6749 および no.6751 のログから，テストプログラムが NtCreateFile を呼び出し，”C:\Users\yyamashita\file.txt”をオープンしたことが分かる．また，この時に返されたファイルハンドルは，no.6751 のログから 0x80 であることが分かる．次に，no.6752 および no.6753 のログから，テストプログラムが NtWriteFile を呼び出したことが分かる．当該システムコールの引数に渡されたファイルハンドルが 0x80 であることから，書込み先ファイルが”C:\Users\yyamashita\file.txt”であることが分かる．

図 7 では，no.6774 および no.6775 のログから，テストプログラムが NtCreateFile を呼び出し，”C:\Users\yyamashita\file.txt”をオープンしたことが分かる．また，この時に返されたファイルハンドルは，no.6775 のログから 0x80 であることが分かる．次に，no.6776 および no.6777 のログから，テストプログラムが NtReadFile を呼び出したことが分かる．当該システムコールの引数に渡されたファイルハンドルが 0x80 であることから，読出し元ファイルが”C:\Users\yyamashita\file.txt”であることが分かる．

以上の結果はテストプログラムの挙動を正しく表している．よって，改良後の Alkanet 10 が少なくとも NtCreateFile，NtWriteFile，NtReadFile の挙動を正しくトレースできることを確認した．

5. おわりに

本論文では，Alkanet 10 における Meltdown 対策済 Windows 10 のシステムコールトレース手法について述べた．Meltdown 対策機能である KVAS が Windows 10 に実装さ

{"cpu_id":2,"no":6749,"logtype":"ENTER","proc_pid":"1c40","proc_tid":"1c1c","proc_name":"test_file_1806","name":"NtCreateFile","sys_no":"55","type":"syscall","args":["5ee3effa20","40100080","5ee3effa98","5ee3effa38","0","80","15e00000000","5e00000003","7ff600000060","0","0"],"additional_info":{"desired_access":"40100080","file_name":"\??\C:\Users\yyamashita\file.txt"}}

{"cpu_id":2,"no":6751,"logtype":"EXIT","proc_pid":"1c40","proc_tid":"1c1c","proc_name":"test_file_1806","name":"NtCreateFile","sys_no":"55","type":"sysret","args":["5ee3effa20","40100080","5ee3effa98","5ee3effa38","0","80","15e00000000","5e00000003","7ff600000060","0","0"],"ret_val":"0","additional_info":{"object_handle":"80","desired_access":"40100080","file_name":"\??\C:\Users\yyamashita\file.txt"}}

{"cpu_id":2,"no":6752,"logtype":"ENTER","proc_pid":"1c40","proc_tid":"1c1c","proc_name":"test_file_1806","name":"NtWriteFile","sys_no":"8","type":"syscall","args":["80","0","0","0","5ee3effb70","7ff6396a12e8","d","0","0"],"additional_info":{"FileHandle":"80","Buffer":"396a12e8","Length":13,"ByteOffset":"41aad6e8"}}

{"cpu_id":2,"no":6753,"logtype":"EXIT","proc_pid":"1c40","proc_tid":"1c1c","proc_name":"test_file_1806","name":"NtWriteFile","sys_no":"8","type":"sysret","args":["80","0","0","0","5ee3effb70","7ff6396a12e8","d","0","0"],"ret_val":"0","additional_info":{"FileHandle":"80","Buffer":"396a12e8","Length":13,"ByteOffset":"41aad6d8"}}

図 6　動作確認結果 (NtCreateFile → NtWriteFile)

{"cpu_id":2,"no":6774,"logtype":"ENTER","proc_pid":"1c40","proc_tid":"1c1c","proc_name":"test_file_1806","name":"NtCreateFile","sys_no":"55","type":"syscall","args":["5ee3effa20","80100080","5ee3effa98","5ee3effa38","0","80","15e00000000","5e00000003","60","0","0"],"additional_info":{"desired_access":"80100080","file_name":"\??\C:\Users\yyamashita\file.txt"}}

{"cpu_id":2,"no":6775,"logtype":"EXIT","proc_pid":"1c40","proc_tid":"1c1c","proc_name":"test_file_1806","name":"NtCreateFile","sys_no":"55","type":"sysret","args":["5ee3effa20","80100080","5ee3effa98","5ee3effa38","0","80","15e00000000","5e00000003","60","0","0"],"ret_val":"0","additional_info":{"object_handle":"80","desired_access":"80100080","file_name":"\??\C:\Users\yyamashita\file.txt"}}

{"cpu_id":2,"no":6776,"logtype":"ENTER","proc_pid":"1c40","proc_tid":"1c1c","proc_name":"test_file_1806","name":"NtReadFile","sys_no":"6","type":"syscall","args":["80","0","0","0","5ee3effb60","5ee3effbe0","5e0000000d","0","0"],"additional_info":{"FileHandle":"80","Length":13,"ByteOffset":"41aad6e8"}}

{"cpu_id":2,"no":6777,"logtype":"EXIT","proc_pid":"1c40","proc_tid":"1c1c","proc_name":"test_file_1806","name":"NtReadFile","sys_no":"6","type":"sysret","args":["80","0","0","0","5ee3effb60","5ee3effbe0","5e0000000d","0","0"],"ret_val":"0","additional_info":{"FileHandle":"80","Buffer":"e3effbe0","Length":13,"ByteOffset":"41aad6d8"}}

図 7　動作確認結果 (NtCreateFile → NtReadFile)

れたことで，システムコールの入口関数と出口関数が変更された．具体的には，システムコールの入口関数が KiSystemCall64Shadow に，出口関数が KiKernelSysretExit に変更されていた．Alkanet 10 のブレークポイント設定位置を修正し，システムコールトレースの動作確認を行った結果，少なくとも NtCreateFile，NtWriteFile，NtReadFile の 3 つのシステムコールを正しくトレースできることを確認した．

今後は，NtCreateFile，NtWriteFile，NtReadFile 以外のトレース対象システムコールについて，同様に動作確認を行っていく．また，Alkanet 10 はマルウェアが行うコードインジェクションの挙動を追跡するために，システムコールトレース機能に加えてスタックトレース機能も有している．そのため，スタックトレース機能についても動作確認を行っていく予定である．

参考文献

[1] Deep Instinct: Beware of the 64-bit Malware, http://info.deepinstinct.com/whitepaper-beware-of-the-64-bit-malware (2017).

[2] 大月勇人，瀧本栄二，齋藤彰一，毛利公一：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).

[3] Shinagawa, T., Eiraku, H., Tanimoto, K., Omote, K., Hasegawa, S., Horie, T., Hirano, M., Kourai, K., Oyama, Y., Kawai, E. et al.: Bitvisor: a thin hypervisor for enforcing i/o device security, *Proceedings of the 2009 ACM SIGPLAN/SIGOPS international conference on Virtual execution environments*, ACM, pp. 121–130 (2009).

[4] 大月勇人，中野進，明田修平，瀧本栄二，齋藤彰一，毛利公一：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，コンピュータセキュリティシンポジウム 2015 論文集， Vol. 2015, No. 3, pp. 839–846 (2015).

[5] 山下雄也，明田修平，瀧本栄二，齋藤彰一，毛利公一：仮想計算機モニタを用いた Windows 10 64bit 環境におけるスタックトレースの実現，コンピュータセキュリティシンポジウム 2017 論文集，Vol. 2017, No. 2 (2017).

[6] Lipp, M., Schwarz, M., Gruss, D., Prescher, T., Haas, W., Mangard, S., Kocher, P., Genkin, D., Yarom, Y. and Hamburg, M.: Meltdown, *ArXiv e-prints* (2018).

[7] swiat: KVA Shadow: Mitigating Meltdown on Windows - Security Research & Defense, https://blogs.technet.microsoft.com/srd/2018/03/23/kva-shadow-mitigating-meltdown-on-windows/ (2018).

[8] Yarom, Y. and Falkner, K.: FLUSH+ RELOAD: A High Resolution, Low Noise, L3 Cache Side-Channel Attack., *USENIX Security Symposium*, Vol. 1, pp. 22–25 (2014).