Vol.2011-DPS-146 No.44
Vol.2011-CSEC-52 No.44
2011/3/11

emit 命令を用いた XEN 仮想マシン上のセキュリティインシデントの可観測化と可視化

安藤類央[†1]　高橋一志[†2]　須崎有康[†3]

本論文では、XEN 上で稼動する仮想マシン上で起こるセキュリティインシデントを、ドメイン U の EMIT 命令発行とハイパーバイザーの修正によりドメイン 0 に通知することで可観測化し、コンソールログを定量化、可視化する手法を提案する。ログの通知時には仮想 CPU のコンテキストを用いて、EMIT 命令によりドメイン U から HYPERCALL を発行することにより VMENTER と VMEXIT を任意に切り替えることでドメイン 0 からドメイン U へイベントログ文字列を送信する。評価実験では、ドメイン U の Windows OS のレジストリアクセスをドメイン 0 へ通知し、自己組織化マップによってイベントの可視化を行った。

A visualization of security incident of XEN virtual machine using emit instruction

Ruo Ando,[†1] Kazushi Takahashi[†2]
and Kuniyasu Suzaki[†3]

In this paper we propose an interdomain communication protocol of XEN virtual machine by involing emit instruction. DomainU can convey information to hypervisor using virtual CPU context in an arbitrary point of Windows kernel by generating hypercall. In experiment, a registry access of virtualized Windows OS is transferred to hypervisor of which log is visualized by self organization map.

†1 情報通信研究機構
National Institute of Information and Communication Technology
†2 東京大学 大学院 情報理工学系研究科
Graduate School of Information Science and Technology, The University of Tokyo
†3 産業技術総合研究所
Advanced Industrial Science and Technology

1. はじめに

仮想化技術の普及に伴い、ハイパーバイザーを用いた研究が盛んになっている。これらはメモリ関連の detectablity についてのトピックが多く、Windows OS の高レベルのイベントであるファイルアクセスやレジストリアクセスについて扱ったものは少ない。本論文では、高レベル API のインターセプトとこれによって得られるイベントログをハイパーバイザー側に通知することで可観測化し、またモニタ側での解析処理による可視化を行う手法を提案する。

2. 関連研究

ハイパーバイザーを用いたマルウェアの解析を、AMD プロセッサのスペックを用いて行ったものに [1] がある。仮想マシンの観測については [2] で VM introspection として提案されている。仮想マシンの monitoring については、XEN[3], VMWARE ESX[4] などで行われることが多いが、debugging や reply[6] に関しては KVM[5] を用いられることもある。仮想マシン上の不正 process の検出に関しては [10][11] で提案されている。domainU と domain0 の間のセマンティックギャップを解消する方式として、仮想マシン上でどのようなバイナリファイルが実行されたか識別する [12] がある。仮想 Windows OS の active monitoring については [10] で提案されている。

3. 仮想マシンの可観測化

仮想マシンの可観測化に必要な要素は、[1] 仮想 Windows OS の修正、[2]XEN ハイパーバイザーの修正、[3] ドメイン間通信方法の 3 点である。[1][2] に関しては、仮想 Windows のどの処理レイヤーに probe を挿入し、ハイパーバイザーのどの部分で通知を受けるかが重要になる。本論文では、Windows カーネルの SSDT の修正箇所を観測通知点とし、XEN 側の hypercall の割り込みハンドラを通知受信点とした。これにより、レジストリアクセスのログを仮想マシンモニタ側から観測可能にした。

3.1 ドメイン間通信

仮想マシンを可観測化するためのコンセプトとして、VM introspection がある。仮想マシンの状態を取得する手法には、passive monitor と active monitor の 2 種類があるが、双方ともメモリと I/O デバイスを対象にしており、ファイルアクセスやレジストリアクセスのセマンティックギャップを解消するための手法の提案は少ない。XEN に関しては、domainU(仮想

© 2011 Information Processing Society of Japan

Vol.2011-DPS-146 No.44
Vol.2011-CSEC-52 No.44
2011/3/11

マシン) の設定やログを通知するための機構に、Xenstore や共有メモリを用いた paravitual driver などがある。本論文では、domainU のリソース消費を最小限に抑えるため、共有メモリや専用のファイルシステムは用いずに仮想レジスタによるシリアル通信を適用した。図 1 は、提案ドメイン間通信を示したものである。domainU とハイパーバイザーは、仮想 CPU のコンテキストを介して通信を行う。はじめに、domainU からイベントの種別と捕捉したイベントログの長さを通知する。これにより、ハイパーバイザーは、長さ分のメモリを確保する。その後、domainU とハイパーバイザー間で一文字ずつデータを転送する。その際、分割した文字と何文字目かの情報を通知することにより、ハイパーバイザー側でログ文字列を再構築できるようにする。最後に、domainU からログ出力要求に応じて、XEN は dmesg のバッファからログを出力する。本手法により、仮想マシン側ではメモリと IO 要求を発生させずに、ログ情報を共有することが可能になる。

3.2 仮想 Windows OS の修正

3.2.1 SDT の修正

仮想 Windows OS の修正の際の目標は、ファイルアクセス、レジストリアクセスを、ハイパーバイザーに通知可能な状態で捕捉することである。そのため、モニタやフィルタのソフトウェアをコンパイル可能な形で実装する必要がある。本論文では、イベントの捕捉に SDT(service decsriptor table) の修正を用いた。SDT は Windows カーネルシステムコールのネイティブコードへのアドレステーブルを持つ構造体であり、このテーブルにある該当システムコールのアドレスを書き換えることで、API の intercept を行うことが可能である。SDT は C:/Windows/system32 の ntoskrnel.exe に記述されており、テーブルの修正は下記の 2 つのマクロを用いた。

```
#define
SYSCALL_INDEX(f)
*(PULONG)((PUCHAR)(f)+1)
```

指定したシステムコールアドレスの+1 から 4byte はインデックスが格納されているアドレスへのポインタとなっている。システムコールアドレス (+0) は op コード B8 で、先の 4byte を合わせると、ニーモニックは mov eax, [xxxx] （xxxx = インデックス）となる (SDT のテーブル参照のためレジスタ eax にインデックスをコピーしている)。

```
SYSTEMSERVICE(f)
KeServiceDescriptorTable->
ServiceTable[SYSCALL_INDEX(f)]
```

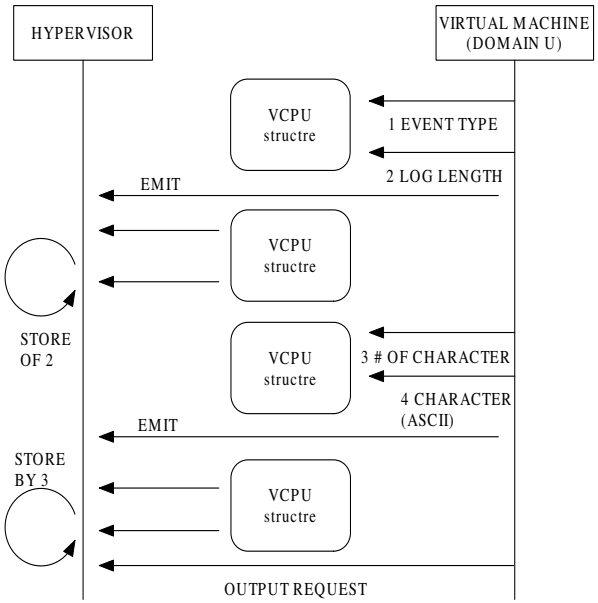

図 1 仮想 CPU のコンテキストを用いたドメイン間通信。domainU とハイパーバイザー双方でログ文字列長、文字列の位置、文字コード (ASCII) を用いてシリアルにイベントログ情報を共有する。

ⓒ 2011 Information Processing Society of Japan

Vol.2011-DPS-146 No.44
Vol.2011-CSEC-52 No.44
2011/3/11

SDT 内の上記インデックスに格納されているシステムコールコードへのアドレスを返すマクロ。ここの値を自身が用意したフック関数のアドレスに書き換えことにより、SDT の修正が可能になる。

3.2.2 emit 命令の利用

Windows の開発環境では、hypercall を直接呼び出す命令はサポートされていない。そのため、本論文では emit 命令を用いて hypercall を仮想 Windows から呼び出す方法を用いた。emit は、インラインアセンブラなどで非対応の命令を実行する際に用いる。

```
_emit 0x0f
_emit 0x01
_emit 0xc1
```

上記命令により機械語を直接操作し、MSIL(microsoft intermediate Language) ストリームに指定した文字列と文字列数を書込みことにより、hypercall のハンドラを仮想 windows 上から activate することが可能である。

3.3 XEN hypervisor の修正

仮想 Windows OS 上での emit 命令を用いることにより、カーネルの任意の位置で hypercall を呼び出すことが可能である。この一連の命令により、xen-3.4.1/xen/arch/x86/hvm/hvm.c の下記関数が呼び出される。

```
int hvm_do_hypercall
(struct cpu_user_regs *regs)
{
    struct vcpu *curr = current;
    struct segment_register sreg;
    int mode = hvm_guest_x86_mode(curr);
    uint32_t eax = regs->eax;
```

ここで現在の仮想 CVPU の状態を得ることが可能であり、各レジスタには仮想 Windows OS 上の emit 命令発行直後の値が格納されており、これにより domain0 と U の間で値をやり取りすることが可能になる。

4. 仮想マシンの可視化

本節では、仮想マシンの可視化の際に用いたデータ、前処理、アルゴリズムについて述べる。

4.1 処理データ

本節では、仮想マシンの可視化に可観測化したレジストリアクセスログを用いた。下記にログの出力例を示す。

```
128799977931406250 ctfmon.exe(324) EnumKey 0x00000000
HKEY_LOCAL_MACHINE\SOFTWARE\Microsoft\CTF\TIP
{78CB5B0E-26ED-4FCC-854C-77E8F3D1AA80}
```

ここで、上列は８つの値で構成される。

- システム時間
- 呼び出し元のプロセス情報
- 呼び出されたレジストリ関数名
- 関数の返り値
- レジストリキー
- レジストリバリュー
- 設定値
- 取得値

レジストリアクセスログを定量化する際には、レジストリキー、レジストリの関数名での出現単語の頻度をカウントし、行列を生成した。

4.2 自己組織化マップ

本論文ではレジストリアクセスデータの可視化に自己組織化マップ [14] を用いた。自己組織化マップは、教師なし学習アルゴリズムの１つであり、K 平均法等と異なりクラスタ数を所与としないところに特徴がある。また、自己組織化マップの特長に、データ間のトポロジーを変えずに学習を行うことが可能という点がある。競争層を２次元に設置すると、多次元の入力データ間の位相関係を保持しつつ、可視化を行うことが可能である [15]。

動作式は下記になり、その他のニューラルネットワークのアルゴリズムと同じくニューロン間の重みを、入力ベクトルと加重ベクトルの差分により修正する。

$$Wv(t+1) = Wv(t) + \Theta(v,t)a(t)(D(t) - Wv(t))$$

ここで W はノード間の加重係数行列、D は入力ベクトル、アルファは学習係数である。自己組織化マップでは、各ノードに対して BMU(best matching unit) を決定する。上式のシータは BMU からの距離によって変化する。学習過程を終了させる敷居値は、学習回数を

ⓒ 2011 Information Processing Society of Japan

Vol.2011-DPS-146 No.44
Vol.2011-CSEC-52 No.44
2011/3/11

用いる。自己組織化マップの可視化の過程は後述する。

4.3 仮想マシンの可視化

本論文では、提案手法の評価実験として、セキュリティインシデントの発生した仮想マシンと状態の類似した仮想マシンを自己組織化マップでの可視化を行った。

評価実験では、ウィルスの動作を感染動作をインストール、通信（パケット送信）に分類し、それぞれ動作が類似するアプリケーションとデバイスドライバのインストール、P2P アプリケーションと WEB ブラウザの実行時のデータを用意し、分類識別を行った。処理を行った動作パターンは下記の通りである。

```
<Pattern I>
input 1: installation of text editor
input 2: installation of device driver
input 3: installation and
 execution of malware

<Pattern II>
input 1: running P2P application
input 2: running Internet Explorer
input 3: installation and
         execution of malware

<Pattern III>
input 1: running P2P application
input 2: installation of device driver
input 3: installation and
         execution of malware
```

それぞれの可視化結果を図２～４に示す。図２は、マルウェアによる感染と悪意のないソフトウェアのインストールを可視化識別したものである。マルウェアのインストールは図の中央側、ソフトウェアのインストールはアプリケーションとデバイスドライバの２つであるが、これは図の右側に配置されている。 図３は、マルウェアの感染実行と、アプリケーションのインストール、ネットワークアプリケーションの実行を可視化識別したものである。これらの３ケースはそれぞれ分離され、図の３箇所に配置される結果になった。マルウェアの感染実行と通信系アプリケーションの実行を可視化識別したものである。通信系アプリケーションは P2P と WEB ブラウザであり、同２状態は図の中央から右側に、マルウェアの感染状態は左側に配置されている。

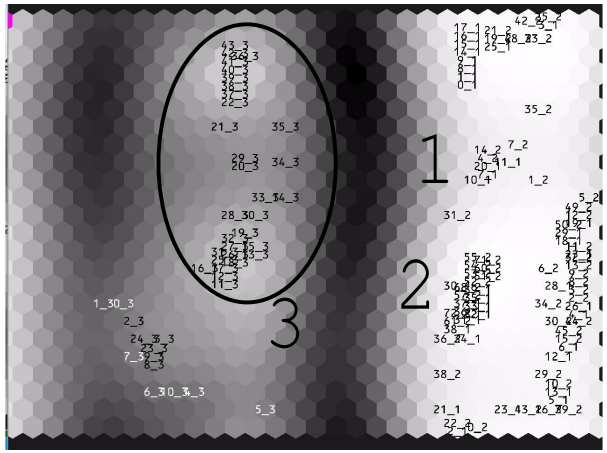

図 2 Clustering and visualization of installing application(1), device driver(2) and malicious software(3). Cluster of malware is well distinguished on the left side of figure.

5. まとめと今後の課題

仮想化技術の発展により、セキュリティインシデントやマルウェア感染動作の解析をハイパーバイザーを用いて行う研究が行われている。これらの研究は、主にメモリ監視による方法が多く、ファイルアクセスやレジストリアクセスなどの高レベルＡＰＩのセマンティックギャップの補足が難しいという面がある。また、xenstore などの専用ファイルシステムや共有メモリを用いた方法は仮想マシンにメモリ利用やファイルＩＯなどの負荷がかかる。本論文では、XEN 上で稼動する仮想マシン上で起こるセキュリティインシデントを、ドメイン U の EMIT 命令発行とハイパーバイザーの修正によりドメイン０に通知することで可

ⓒ 2011 Information Processing Society of Japan

Vol.2011-DPS-146 No.44
Vol.2011-CSEC-52 No.44
2011/3/11

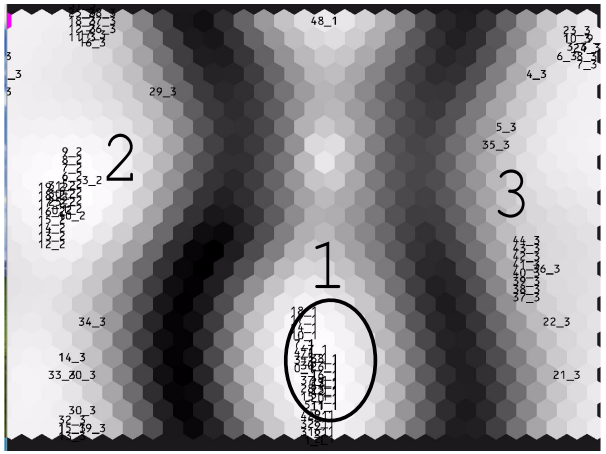

図 3 Clustering and visualization of running application(1), installing device driver(2) and malicious software(3). Each behavior is well clustered on left, center and right side of figure.

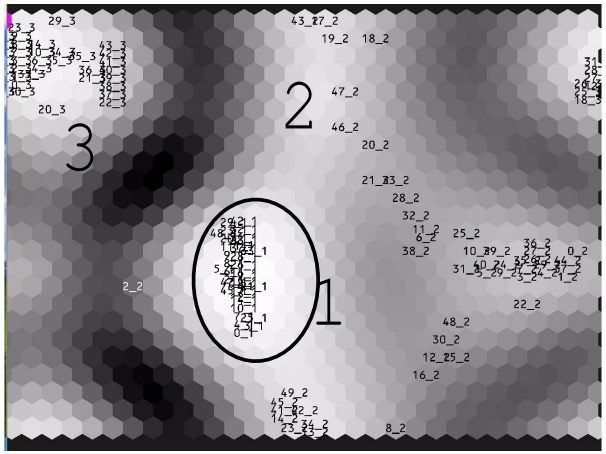

図 4 Clustering and visualization of running network application of P2P(1) and Web(2). Also, malware's behavior(3) is clustered on the left of figure.

観測化し、コンソールログを定量化、可視化する手法を提案した。仮想 Windows OS 上ではＳＤＴを修正し、ログ文字列をメモリ上で構築し、EMIT 命令発行時の仮想ＣＰＵのコンテキストを用いて仮想マシンモニタ側へ情報を通知する。評価実験では、ドメインＵの Windows OS のレジストリアクセスをドメイン０へ通知し、自己組織化マップによってイベントの可視化を行い、マルウェア感染動作、ネットワークアプリケーション実行、ソフトウェアのインストールの３ケースについて可視化識別を行った。

参 考 文 献

1) A Virtual Machine Introspection Based Architecture for Intrusion Detection Tal Garfinkel and Mendel Rosenblum In the Internet Society's 2003 Symposium on Network and Distributed System Security (NDSS), pages 191–206, February 2003.
2) Virtual Machine Introspection: Observation or Interference? K. Nance, M. Bishop, and B. Hay, " Virtual Machine Introspection: Observation or Interference?, "IEEE Security and Privacy 6(5) pp. 32?37 (Sep. 2008).
3) P. Barham, B. Dragovic, K. Fraser, S. Hand, T. Harris, A.Ho,R. Neugebauer, I. Pratt, and A.Wareld. Xen and the Art of Virtualization. In Proceedings of the 19th ACM SOSP, pages 164?177, October 2003.
4) C. A. Waldspurger. Memory resource management in VMware ESX server. In Proceedings of the 5th Symposium on Operating Systems Design and Implementation, pages 181-194, December 2002.
5) Kernal Virtual Machine available at: http://sourceforge.net/projects/kvm
6) George W. Dunlap, Samuel T. King, Sukru Cinar, Murtaza Basrai, and Peter M. Chen. ReVirt: Enabling intrusion analysis through virtual-machine logging and replay. In Proceedings of the 2002 Symposium on Operating Systems Design and Implementation (OSDI 2002), Boston, MA, December, 2002
7) S. King, G. Dunlap, and P. Chen, " Debugging Operating Systems with Time-Traveling Virtual Machines, "Proc. Annual Usenix Tech. Conf., Usenix Assoc., 2005;
8) A. Whitaker et al., " Constructing Services with Interposable Virtual Hardware, " Proc. 1st Symp. Networked Systems Design and Implementation (NSDI 04), Mar. 2004.
9) B. Payne et al., " Lares: An Architecture for Secure Active Monitoring Using Virtualization, " Proc. IEEE Symp. Security and Privacy, IEEE CS Press, 2008, pp. 233?247.
10) S. Jones, A. Arpaci-Dusseau, and R. Arpaci-Dusseau, " VMM-based Hidden Process Detection and Identification Using Lycosid, " Proc. ACM Int 'l Conf. Virtual

ⓒ 2011 Information Processing Society of Japan

Vol.2011-DPS-146 No.44
Vol.2011-CSEC-52 No.44
2011/3/11

Execution Environments (VEE 08), ACM Press, 2008, pp. 91?100.
11) S. Jones, A. Arpaci-Dusseau, and R. Arpaci-Dusseau, “ AntFarm: Tracking Processes in a Virtual Machine Environment, ”Proc. Annual Usenix Tech. Conf., Usenix Assoc., 2008, pp. 1?14.
12) Lionel Litty, H. Andres Lagar-Cavilla, Hypervisor Support for Identifying Covertly Executing Binaries, Usenix 2008, The 17th USENIX Security Symposium July - August, 2008
13) T. Kohonen, ”Self-Organizing Maps”, Springer, 2000, ISBN 978-3540679219.
14) Willshaw, D.J.; von der Malsburg, C. (1976). How patterned neural connections can be set up by self-organization ”. Proceedings of the Royal Society of London. Series B, Containing papers of a Biological character. 194 (1117): 431-45. PMID 12510
15) Ruo Ando, Nguyen Anh Quynh, Kuniyasu Suzaki, ”A visualization ofanomaly memory behavior of full-virtualized Windows OS using virtualmachine introspection”,the 2009 International Conference onInformationSecurity and Privacy (ISP-09) 13-16 of July 2009 in Orlando, FL, USA
16) 安藤類央, 三輪信介, 門林雄基, 篠田陽一, ”仮想 Windows OS の完全性検証のためのドメイン間プロトコルの実装”, SCIS2011 暗号と情報セキュリティシンポジウム

ⓒ 2011 Information Processing Society of Japan