Vol.2009-OS-112 No.21
2009/8/6

VM Introspection を用いたWindows OS のメモリ上の異常挙動の可視化

安藤　類央[†1]　Nguyen Anh Quynh[†1]
須崎　有康[†1]

近年、ネットワークに接続される Windows OS のアプリケーションは質量ともに増加傾向にあり、本論文では、仮想化技術を用いたメモリ上の異常挙動を把握するための可視化手法を提案する。提案システムは、（１）観測系構築のための、Windows OS へのライブラリとフィルタドライバのインストール、（２）仮想マシンモニタのデバッグレジスタハンドラの修正によるメモリ状況の通知、（３）ホストＯＳ側での可視化ツールの導入にの３段階からなる。提案システムによって、Buffer overflow によるシェルコード実行、Ｐ２Ｐソフトウェアの実行を可視化し、異常検知を行った。

A visualization of memory state transition of Windows OS exploiting virtual machine introspection

Ruo Ando ,[†1] Nguyen Anh Quynh [†2]
and Kuniyasu Suzaki [†2]

Nowadays, a variety of applications of Windows OS has become more sophisticated and the complexity of its behavior has also been increased. Proposed system has been implemented in tree steps: (1) modification of Windows OS by inserting library and filter driver, (2) modification of the debug register handler of virtual machine monitor, and (3) deploying visualization tool on host OS. We generate the sequence of memory behavior using virtual machine introspection. present the visualization by proposed system, we deal with two popular security issues: BoF (buffer overflow) based shellcode execution and P2P network application. We have successfully detected the incidents of BoF based shellcode detection, and visualize the memory state transition of P2P application.

1. はじめに

近年、ネットワークに接続される Windows OS のアプリケーションは質、量ともに急激な増加傾向にある。その結果、仮想化技術の普及とも相俟って、ネットワークアプリケーション間の依存関係は複雑化しており、定性的なログ解析によるネットワーク上のクライアントの稼動状況の異常を把握する際の情報処理付加が増加している。本論文では、仮想化技術を用いたメモリ上の異常挙動を把握するための可視化手法を提案する。

ここ数年のデバッグ技術の発達は目覚しいものがあり、仮想化技術の発展とあわせて、より詳細な観測と、高粒度なロギングが可能になってきている。特に、Microsoft Windows が提供するデバッギングＡＰＩ、ネットワークアプリケーション、そしてカーネルモジュールの開発のための環境も急速に整備されつつある。また、VMWare などが提供しているＡＰＩや、オープンソースの仮想マシンモニタの普及により、ホストＯＳのドライバと仮想マシンモニタ側のメモリ管理ユニットや割り込みハンドラが連携し、ゲストＯＳの情報をホストＯＳ側に通達できることになった。

図１、Linux KVM (Kernel Virtual Machine) 上で実装した提案システムの概要を示したものである。ゲスト Windows OS 上にデバイスドライバとライブラリを挿入し、修正したVMMのモジュールと連携して、ログをホストＯＳ側に書き込む。その後、ホストＯＳ側の可視化ツールが、ゲストＯＳの状態（主にメモリの状態遷移）を可視化する。

2. 関連研究

現在、仮想マシンモニタとしてよく用いられているものに、KVM[1] と XEN[2][3] がある。HIDS と NIDS、そして VMM に IDS を配置する手法の比較検討が、[4] において行われている。[5][6] では、ハイパーバイザーのレイヤにセキュリティモジュールを配置する有効性が検討されている。特に、[7] は、仮想化された Windows XP のイベント検出を扱っている。本論文では、VMM 上の Windows OS 仮想化とデバッグ手法の改良によるインシデントの扱いに焦点を当てる。

†1 情報通信研究機構情報通信セキュリティセンター〒 184-8795 東京都小金井市貫井北町４－２－１
†2 産業技術総合研究所情報セキュリティセンター〒 101-0021 東京都千代田区外神田 1-18-13

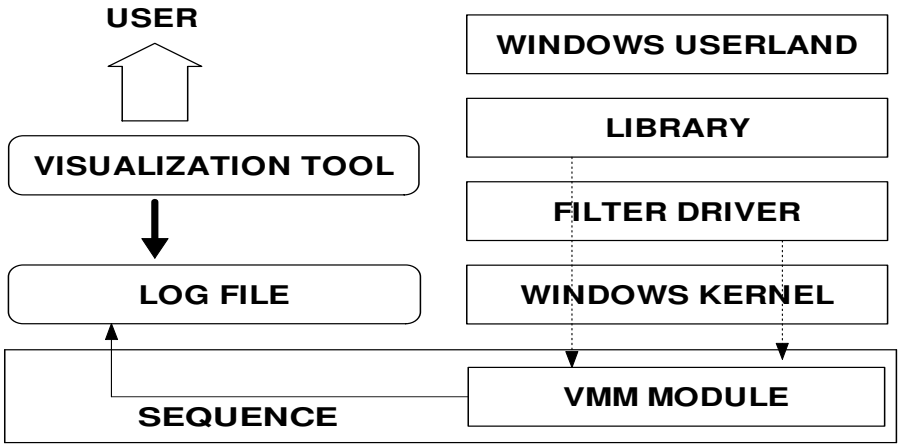

図 1 Brief illustration of proposed visualization system. Inserted library and filter driver on full-virtualized Windows extract a sequence of memory behavior. The sequence is transferred by modified register handler to visualization tool on host OS.

3. 提案システム

3.1 VM Instrospection

近年のプロセッサ性能の急速な向上は、複数の OS を同時に稼動させる仮想化技術の実用性を高めた。仮想マシンモニタなどのシステムは、従来のデバッガやダンプツールと比較して、OS やプロセスの外部観測を容易にした。また、完全仮想化モードは、プロテクトモード以来のプロセッサ構造の革新と指摘されており、この仮想化技術が提供する外部観測性は、デバッグやソフトウェアのチェック、特に VMM（仮想マシンモニタ）に関してはセキュリティ機能の実装の観点から、注目されている。

3.2 DLL Injection

DLL Injection とは、任意の関数をライブラリ化して特定の API が発行されたときに、実行させるもので、既存のアプリケーションに新しい機能を加えたいとき、ソフトウェアのデバッグの際に用いられる。開発の要請から、DLL（独自 API) を他のプロセスに任意のタイミングで実行させる DLL Injection という技術が用いられることがある。DLL Injection の方法には、Mi-crosoft Windows が提供している機能を使うもの、デバッグ機構の機能を使うもの、リモートスレッドを使うもの、そしてモジュールのインポートセクションの変更によるものなどがある。本論文では、モジュールのインポートセクションの変更による方法により、対象ソフトウェアのデータ送受信関数をフックし、ここに適切な操作を入れることにより、ソフトウェアの挙動を追跡し、問題となるトラフィックが発生または到着する前に防止を行うことを可能にした。図１は、インポートテーブルの変更よる DLL 注入を示したものである。インポートテーブル（インポートセクション）とは、セクションテーブルの中にあるデータ構造体（ヘッダ）で、プログラムが実行される前に必要な DLL のアドレスと、利用する DLL 内のシンボルのアドレスのリストが格納されている。ここで、フックしたい DLL のアドレスを、任意の DLL へのアドレスに書き換えることで、ソフトウェアの動作の修正を行うことができる。この方法は、特定の CPU の仕様に依存しなく、またスレッドの同期の問題もないため、非常に柔軟な方法として用いられることが多い。処理としては、１）注入したい DLL を用意する。２）対象となるソフトウェアのインポートテーブルアドレスを取得する。３）フックしたい関数のアドレスを探す。４）発見したアドレスを、注入したい DLL（関数）へのアドレスに書き換える。関数形は

```
void ReplaceIATTableInP2Psoftware
```

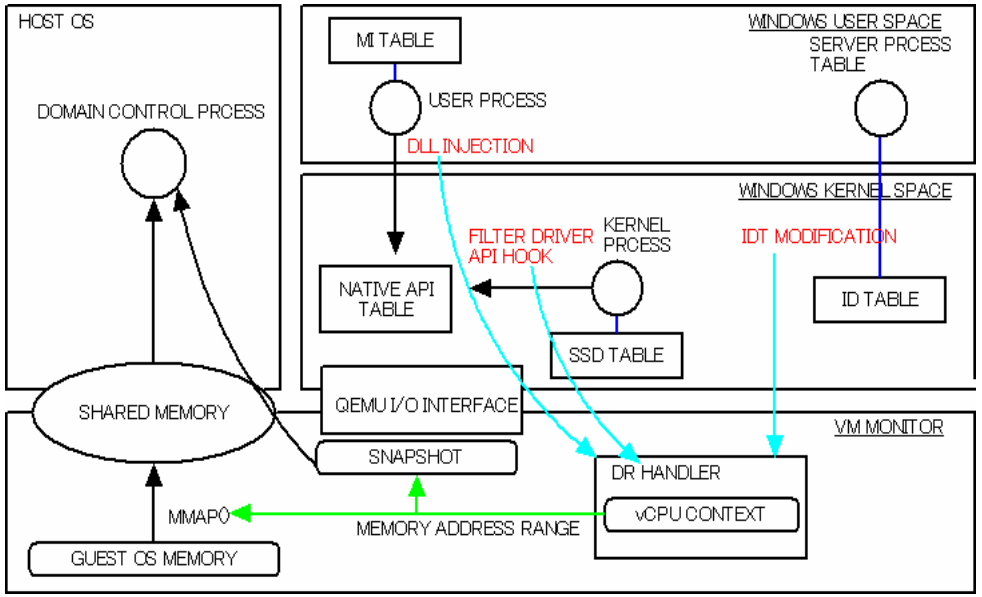

図 2 Detailed illustration of proposed system. Proposed system consists of three steps: [1]modification of Windows OS using DLL injection and filter driver, [2]modification of debug register handler of virtual machine monitor and [3]putting visualization tool on host OS. Figure 1 is a brief illustration of proposed system. Our module extracts a sequence from Windows memory behavior and the sequence is transferred from VMM module to visualization tool on host OS. In following section, we also discuss modification of Windows OS. In this paper we propose visualization of memory behavior of full-virtualized Windows OS using virtual machine introspection. In proposed system, memory behavior of Windows OS is visualized by an application of the concept of virtual machine introspection. Proposed system extracts a sequence of Windows memory behavior and transfers it to host OS by modifying the module of virtual machine monitor. Although all hardware accesses pass through VMM, VMM is not able to understand semantics of guest OS, which means that VMM has no information about what kind of event is happened above. To detect events on virtualized OS correctly and achieve fine-grained monitoring, Windows OS need to be modified. Figure 2 shows the detailed illustration of proposed system, particularly the modification of Windows OS. Windows modification OS consists of three steps: [1]inserting DLL into user process, [2]inserting filter driver into kernel space, and [3]modifying IDT (interruption descriptor table) of daemon process. In this section we discuss library insertion (DLL injection) and filter driver injection. .

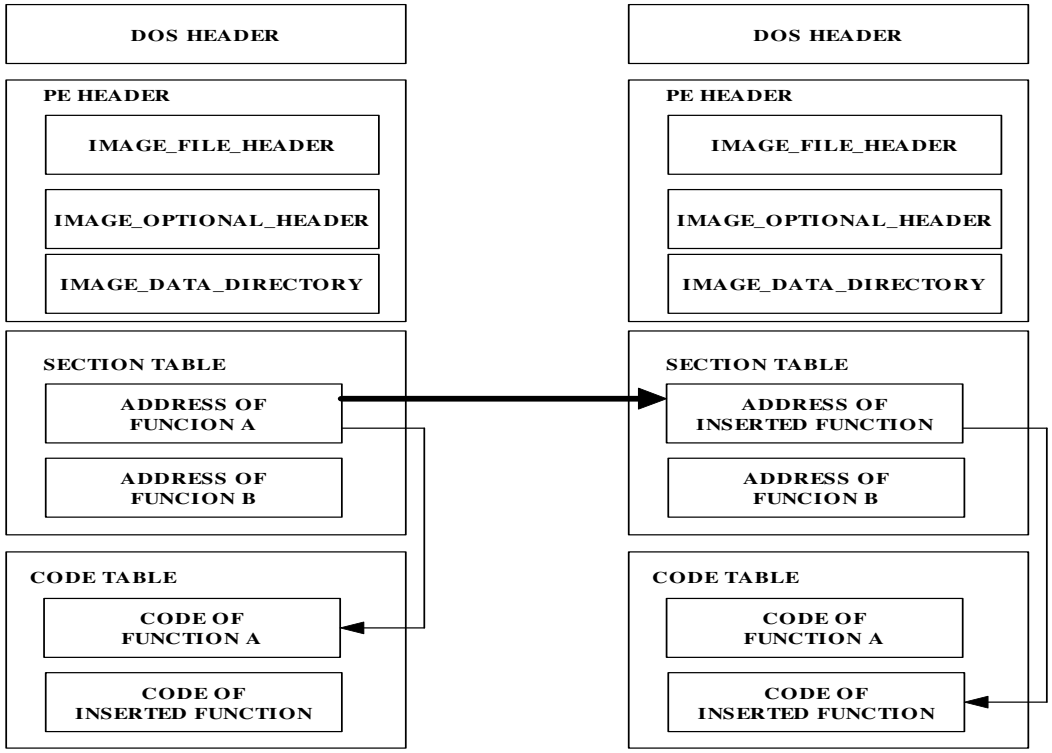

図 3 Dll injection by the modification of import section table. Address of function to hook is changed to address of inserted function. DLL injection is the insertion of original library on userland. We apply DLL injection for inspecting illegal resource access of malicious process. DLL injection is debugging technology to hook API call of target process. Windows executable applies some functions from DLL such as kernel32.dll. Executable has import table to use the linked DLL. This table is called as import section. Among some techniques of DLL injection, modifying import table is useful because this technique is CPU-architecture independent.

```
("kernel32.dll",
funcORG,
funcINSERT",
moduleHandler);
```

対象とするプロセスの構造などにより、実装方法はいくつか存在するが、大枠は以上で示した通りである。

4. 評 価 実 験

本節では、Windows OS の memory exploitation の検出を、提案手法によって行った際の結果について述べる。

4.1 Buffer overflow による shellcode の実行

図は、Buffer Overflow により、被害プロセスに GetProcAddress、LoadLibrary を実行させ、任意のシェルコードを実行した際のメモリ割り当て (mapviewofsection) の可視化した結果である。同実験によって、ＡＰＩ呼び出しの際の割り当てのアドレスがスパイクとして検出できることが明らかになった。

出力されたログは以下のとおりになる。

```
00000864        2.65054226      18;ZwOpenKey;pid;156;KeyHandle;1244276;
DesiredAccess;-2147483648;ObjectAttributes;1243472
00000865        2.65465522      16;MapviewOfsection;pid;156;bAddress;
1243176;pHanlde;-1;zeroBits;0;commitSize;0;
 sectionHandle;0;ViewSize;1243164;aType;0;w32Protect;4
00000866        2.65565228      16;MapviewOfsection;pid;644;bAddress;
1974765848;pHanlde;1484;zeroBits;0;commitSize;0;
 sectionHandle;0;ViewSize;8322728;aType;4194304;w32Protect;32
00000867        2.65628004      18;ZwOpenKey;pid;156;KeyHandle;1241752;
DesiredAccess;131097;ObjectAttributes;1241712

----> UNIT CONFLICT at   0.12 sec ----> 1275 [binary,1274.1,1270.1] $F.

-------- PROOF --------
Length of proof is 3.  Level of proof is 2.

---------------- PROOF ----------------

600 [] c(number(605),api(16),pid(644),adr(7535816)).
861 [] c(number(866),api(16),pid(644),adr(1974765848)).
1267 [] -c(number(x),api(y),pid(644),adr(z))|csrss(number(x),api(y),
        proc(csrss),adr(z)).
1268 [] -c(number(x),api(16),pid(y),adr(x1))|
-$GT(x1,1900000000)|map(number(x),api(16),pid(y),adr(x1)).
1269 [] -csrss(number(x1),api(x2),proc(csrss),adr(x3))|
-map(number(y1),api(16),pid(y2),adr(y3))| -$GT(y1,x1)|ok.
1270 [] -ok.
1271 [hyper,600,1267] csrss(number(605),api(16),proc(csrss),adr(7535816)).
1272 [hyper,861,1268,eval] map(number(866),api(16),pid(644),
        adr(1974765848)).
1274 [hyper,1272,1269,1271,eval] ok.
1275 [binary,1274.1,1270.1] $F.
```

図は、同実験によるシェルコード実行時のプロセスハンドラの取得を可視化したものである。マイナス方向にスパイクしているのは、一つには、シェルコード実行時に、プロセスハンドラの取得に失敗しているものであると推測される。

4.2 P2P と Web アプリケーションの実行

図６と７は、提案システム上でＰ２Ｐアプリケーションである Winny と、Web アプリケーションである FireFox を稼動させた際のメモリ挙動（割り当てアドレスとソケットバッファ）のを可視化したものである。Winny はハイポートを開放して、多数のコネクションを確立し、スレッドを多くきるため（図６右下)、メモリ挙動は図右側のような特徴が抽出される。FireFox は、対象的にマルチプロセスアプリケーションで、多くのＡＰＩを使うことが推測されるため、メモリ挙動は図左側のような特徴が抽出される。

5. まとめと今後の課題

近年のプロセッサ性能の急速な向上は、複数の OS を同時に稼動させる仮想化技術の実用性を高めた。仮想マシンモニタなどのシステムは、従来のデバッガやダンプツールと比較して、OS やプロセスの外部観測を容易にした。また、完全仮想化モードは、プロテクトモード以来のプロセッサ構造の革新と指摘されており、この仮想化技術が提供する外部観測性は、デバッグやソフトウェアのチェック、特に VMM（仮想マシンモニタ）に関してはセキュリティ機能の実装の観点から、注目されている。ここ数年のデバッグ技術の発達は目覚しいものがあり、仮想化技術の発展とあわせて、より詳細な観測と、高粒度なロギングが可能になってきている。特に、Microsoft Windows が提供するデバッギングＡＰＩ、ネットワークアプリケーション、そしてカーネルモジュールの開発のための環境も急速に整備されつつある。また、VMWare などが提供しているＡＰＩや、オープンソースの仮想マシンモニタの普及により、ホストＯＳのドライバと仮想マシンモニタ側のメモリ管理ユニットや割り込みハンドラが連携し、ゲストＯＳの情報をホストＯＳ側に通達できることになった。

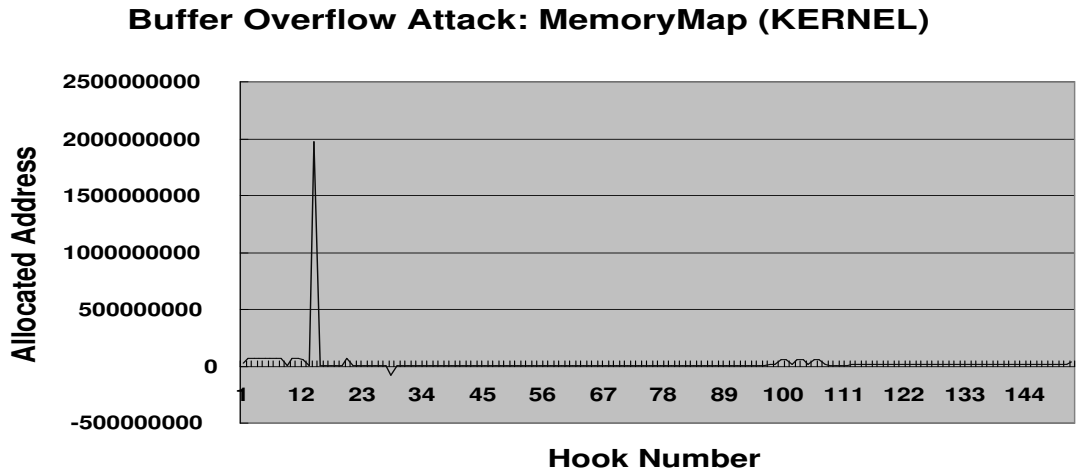

図 4 Detecting buffer overflow and shellcode execution by hooking memory map in kernel spike. With buffer overflow, there's one spike seen in this graph.

近年、ネットワークに接続される Windows OS のアプリケーションは質、量ともに急激な増加傾向にある。その結果、仮想化技術の普及とも相俟って、ネットワークアプリケーション間の依存関係は複雑化しており、定性的なログ解析によるネットワーク上のクライアントの稼動状況の異常を把握する際の情報処理付加が増加している。本論文では、仮想化技術を用いたメモリ上の異常挙動を把握するための可視化手法を提案した。

謝　　辞

本論文の執筆は、新世代セキュリティ研究開発事業「既存 OS に挿入可能な仮想マシンモニタによる異常挙動解析とデバイス制御の研究開発」の一環であり、協力して頂いている関係者各位に謝意を記す。

参 考 文 献

1) Kernel Based Virtual Machine, http://kvm.qumranet.com/kvmwiki
2) XEN virtual machine monitor, http://www.cl.cam.ac.uk/Research/
3) Paul Barham, Boris Dragovic, Keir Fraser, Steven Hand, Tim Harris, Alex Ho, Rolf Neugebauer, Ian Pratt, and Andrew Warfield, ”Xen and the art of virtualization ”, In Proceedings of the 19th Symposium on Operating System Principles(SOSP 2003), Bolton Landing, NY, October 2003.
4) Tal Garfinkel and Mendel Rosenblum, ”A Virtual Machine Introspection Based Architecture for Intrusion Detection ”, In the Internet Society ’s 2003 Symposium on Network and Distributed System Security (NDSS), pages 191.206, February 2003.
5) Nguyen Anh Quynh, Ruo Ando, and Yoshiyasu Takefuji：”Centralized Security Policy Support for Virtual Machine ”, USENIX, 20th Large Installation System Administration Conference, December 2006.
6) Reiner Sailer and Trent Jaeger and Enriquillo Valdez and Ramon Caceres and Ronald Perez and Stefan Berger and John L. Griffin and Leendert van Doorn, ” Building a MAC-Based Security Architecture for the Xen Opensource Hypervisor ”, in Proceedings of the 2005 Annual Computer Security Applications Conference (ACSAC), December 2005.
7) BD Payne, M Carbone, M Sharif, and W Lee. Lares, ”An Architecture for Secure Active Monitoring Using Virtualization ”, In Proceedings of the IEEE Symposium

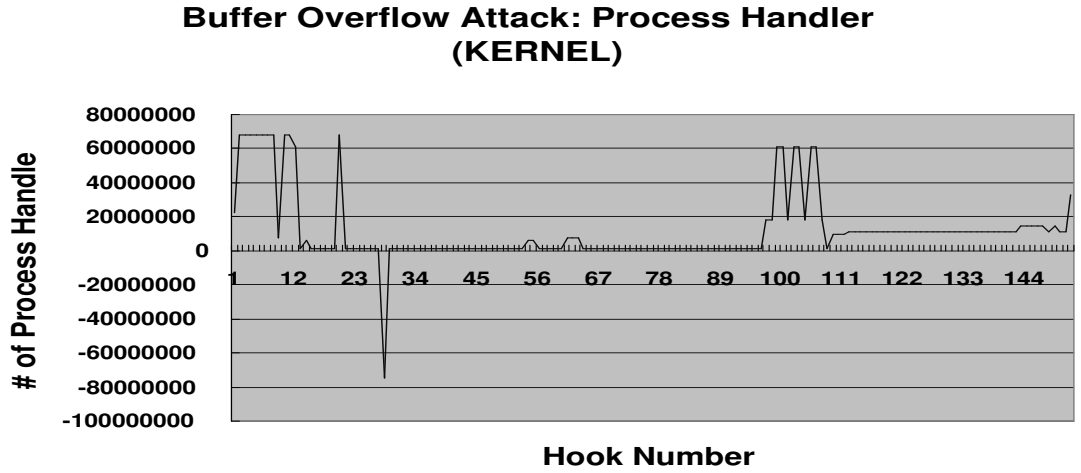

図 5 Sequence of the number of process handler allocated in buffer overflow attack. With bufferover overflow and illegal code execution, the number of process handler goes down after the spike shown in figure 4.

on Security and Privacy (Oakland 2008), May 2008.
8) BAN Tao, ANDO Ruo, KADOBAYASHI Youki, "Monitoring and Analysis of Network Traffic in P2P Environment", Journal of the National Institute of Information and Communications Technology, Vol.55 Numbers 2/3 2008, ISSN 1349-3205, 2008.11.
9) 安藤類央、門林雄基、篠田陽一、「Code injection 検出のための VMM スナップショット機能の強化」、情報処理学会コンピュータセキュリティ研究会第４２回研究報告２００８年7月２５日
10) Ruo Ando, Youki Kadobayashi, Youichi Shinoda, "Incident-driven memory snapshot for full-virtualized OS using interruptive debugging techniques", International Journal of Security and Its Applications, vol.2, No,3,pp41-48, 2008 July, ISSN 1738-9976
11) 安藤類央、門林雄基、篠田陽一、「Memento Project: 仮想化技術を用いた OS 内部観測による解析用データセットの構築と公開」、情報処理学会 コンピュータセキュリティ研究会 第４５回研究報告 ２００９年5月２９日
12) 安藤類央、Nguyen Anh Quynh、須崎有康、「Windows のメモリ挙動モニタと Libvirt によるゼロデイ攻撃の検出システムの構築」, 暗号と情報セキュリティシンポジウム（SCIS2009）2009 年 1 月 21 日
13) 安藤類央、Nguyen Anh Quynh、須崎有康、「仮想マシンモニタへのインシデント通知のための Windows OS 仮想化とデバッグ機構の修正」、情報処理学会 コンピュータセキュリティシンポジウム２００８ ２００８年１０月９日
14) 須崎有康、Nguyen Anh Quynh、安藤類央、「ゼロデイアタックに対処するためにデバイス制御を行う仮想計算機」、情報処理学会 コンピュータセキュリティシンポジウム ２００８ ２００８年１０月９日
15) 安藤類央、門林雄基、篠田陽一、「Code injection 検出のための VMM スナップショット機能の強化」、情報処理学会 コンピュータセキュリティ研究会 第４２回研究報告 ２００８年7月２５日

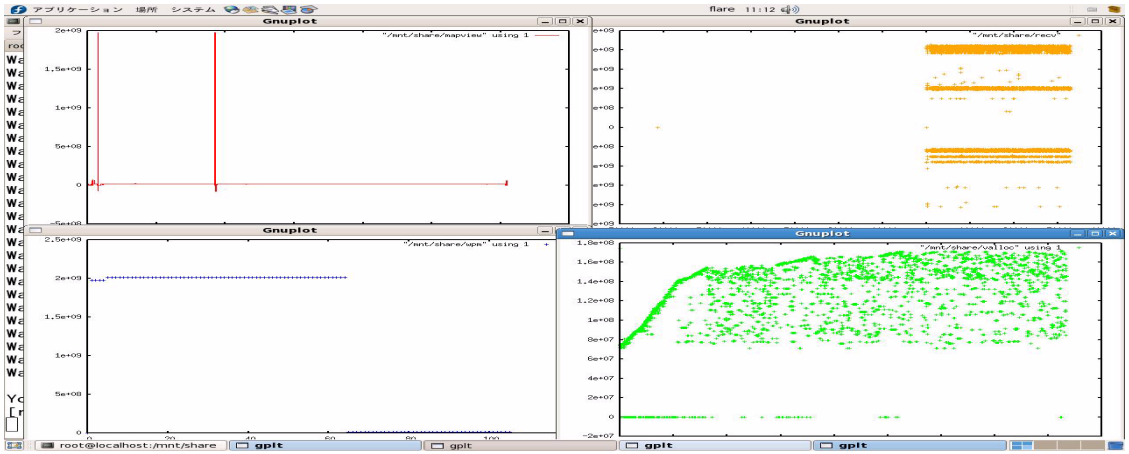

図 6 The visuzalization of the memory state of Windows OS running P2P application. On right side of the screen, many plots according to events hooked is shown which makes it eash to distinguish P2P application behavior.

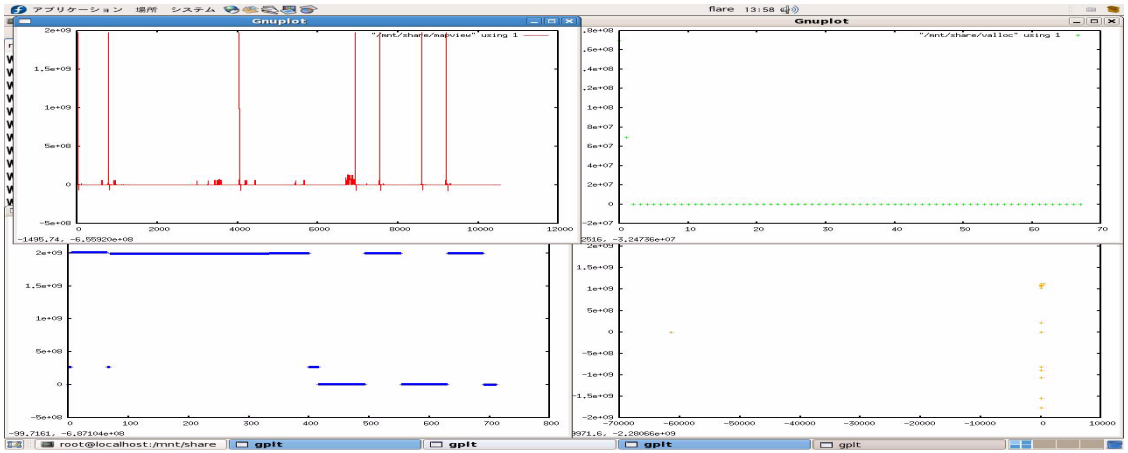

図 7 The visualization of the memory state of Windows OS running Web application. On left side of the screen, memory mapping and writing in user space has been occrred more frequently.