3-4 Towards Traceable Overlay Network over Virtualized Systems

ANDO Ruo

We have been researched the traceability of overlay network including P2P network and virtualized systems. For this purpose, we have developed monitoring system for P2P network and virtual machines running on hypervisor. As one of the design goals, we have aimed to enhance the system about fine-grained and large-scale monitoring for tracing information leaks and malware behavior. In this research, we have cooperated with software developer and SIer concerning information security and evaluated our systems on testbed of NICT.

Keywords
Virtual network monitoring, Virtualization technologies, Overlay network, Information leaks, Testbed

1 Introduction

As networks are increasingly widespread and cloud computing and Pear to Pear (P2P) network technologies are more broadly utilized, we have been using virtual networks composed of virtual machines more frequently. Our medium-term plan in the second term has focused on improving the accuracy of intrusion detection and enhancing monitoring networks as part of research and development of traceable network technology. In our medium-term plan of the previous term, we mainly researched the monitoring of virtual machines and P2P networks to make our traceable network monitoring technologies more accurate and to expand our monitoring ranges.

One of the reasons triggering the current rapid penetration of cloud computing is the advancement of virtualization technology. Irrespective of service types (e.g. SaaS, PaaS, and IaaS), we can define cloud computing as a service enabling the utilization of virtual machines. Based on this understanding, we have been engaged in research on probing technology since making new types of systems including cloud computing more secure requires the monitoring of virtual machines.

Furthermore, information leakage on overlay networks including P2P networks began to be noticed as one of the most important social issues around 2006, during the medium-term plan period of the previous term. This prompted us to develop traceability technologies of this event as a way to research traceable networks. We also enhanced the capability of analysis processing since the data obtained from P2P networks for tracking became very large.

Generally speaking, systematic monitoring and intrusion detection technologies based on virtualization technology is called virtual machine introspection. An Intrusion Detection System (IDS) deployed on virtual machines keeps monitoring objects and probes separated by virtualization technology, which enables new features not available through conventional IDSes. It has been pointed out that conventional Network Intrusion Detection System (NIDS) is harder to be detected by attackers but their data is less accurate and that Host-based Intrusion Detection System (HIDS) is more visible for attackers but their data is more accurate. A system design is being proposed so that we can leverage the characteristics of Virtual Machine Monitor (VMM) and make the maximum use of these contradicting features. Advantages available through the use of VMM when we create an IDS can be summarized as Isolation (i.e. probes are placed separated from the guest OS), Inspection (i.e. detailed data including events made available in the kernel space), and Interposition (i.e. enabling system calls, interruptions, and I/O requests to VMM to be hooked and intermediate actions to be inserted at the same time). Thus, we have created a system to enable intrusion detection based on these features through our R&D efforts.

In P2P, one computer serves both as server and client, and information can be easily exchanged among users and it is also resistant to changes such as the rapid increase of network traffic, and thus P2P is rapidly becoming widely used. On the other hand, infringement of copyright through P2P networks and information leaks due to virus infection via P2P are becoming issues. While P2P software is often released for free, there are more cases where the provision of upgrading and modifications is stopped during the stages of formation and development of development communities, compared with commercial software. Development is sometimes stopped due to various circumstances. This means that even when new vulnerabilities or attacks on overlay networks that are comprised of P2P applications become evident, corresponding modifications and patches are not always prepared. It is therefore necessary in P2P network monitoring to analyze software that comprise the virtual network, extract protocols and data structures and conduct monitoring. During the first half of the medium-term plan period, we conducted research and development of a system to monitor a wide-area virtual network through the analysis of P2P software.

2 Development of virtualization system monitoring technology

The rapid improvement in processor performance in recent years has enhanced the practicability of virtualization technology that can operate multiple OS simultaneously. Systems such as virtual machine monitors have facilitated external monitoring of OS and processes, compared with traditional debugger and dump tools. It has been also pointed out that the complete virtualization mode is the greatest innovation in the processor structure since the protect mode. The capability of external monitoring offered by this virtualization technology is attracting attention with regard to debugging and software checking, and particularly regarding VMM, from the perspective of implementing security functions. In particular, because it is impossible to predict the timing of occurrence of security incidents or estimate processing time, it is important for the target to be monitored for prevention to send a request for processing to the prevention system asynchronously. By using virtualization technology, such processing becomes possible[1]. In addition, when there are multiple OS that are the targets to be monitored for prevention, it becomes possible to configure security policies and control access in an integrated manner by virtualizing and consolidating such OS in one physical machine[2].

Debugging technology, which has experienced significant development during the past few years, coupled with the development of virtualization technology, has enabled more detailed monitoring and highly granular logging. In particular, there has been rapid improvement in the Debugging API provided by Microsoft Windows and network applications as well as the environment for the development of kernel modules. The dissemination of API provided by VMWare and others and open-source virtual machine monitors has enabled the linkage between host OS drivers and memory management units and interrupt handlers on virtual machine monitors, and the communication of information of the guest OS to the host OS. This has led to research on the enhancement of debugging and malware analysis functions utilizing virtualization technology[3].

In recent years, Windows OS applications that are connected to networks are demonstrating an increasing tendency both in terms of quality and quantity. As a result, coupled with the dissemination of virtualization technology, dependency among network applications is becoming more complex, and there is an increased information processing load when assessing abnormalities in the client operation status on networks through qualitative log analysis. This paper proposes the method of monitoring virtual network applications utilizing virtualization technology.

2.1 Types of virtual machine monitors and modification methods

Virtual machine monitors are categorized into two types: one is the type that constructs hypervisors by itself such as XEN[4]; and the other is the type that constructs them inside the host OS (Linux) such as the kernel virtual machine (KVM).

The first type of virtual machine monitor such as KVM uses the host OS as the hypervisor. In this method, the virtual memory of the guest OS that is placed within the kernel space of the host OS is transferred into the user space or the kernel space. This method uses the QEMU interface. Since KVM is taken into the Linux kernel, the latest functions can be used in the management OS, but it is not stable yet because a dedicated API has not been prepared. Therefore, the proposed system was implemented for the transfer of values using debug handlers and transfer of character strings using shared memories.

In the second type of virtual machine monitor such as XEN, its own bootloaders and hypervisors are prepared. In this case, it is either quasi-virtualization or complete virtualization. As this paper deals with the complete virtualization of Windows, when acquiring snapshots, the QEMU interface is used, and not the XEN interface. As is the case with KVM, there is a method of transferring API factors, ASCII codes and others via the vCPU register. XEN is characteristic in that an API is in place, and it is possible to take advantage of the API to analyze the Windows status using memory snapshots, or transfer log information. It is also possible to capture hardware interrupts ahead of the Windows OS.

In both the first and second types of virtual machines, the method of capturing the access and status transitions to hardware or virtualized hardware, and monitoring the behavior of the virtualized machine can be applied. This method is called virtual machine introspection, the details of which are proposed in [5].

2.2 Modification method of virtual Windows OS

The system integration system monitors proposed in this paper enables monitoring that is more granular than monitoring tools provided by Microsoft, with API hooks by DLL Injection, filter drivers and rewriting of system tools, among others. Figure 1 shows the outline of an integration system monitor, which explains how the mechanism for debugging and filtering provided by Windows OS is utilized to intercept access by various resources (memory, sockets, files, registries) to be recorded in the log. Hooks such as the execution of branch instructions and hardware interrupts are dealt with through modification by the virtual machine.

2.2.1 Kernel API Hook by filter drivers

The filter driver of Microsoft Windows is a software module whose active introduction and utilization began with Windows XP. The filter driver, which is located between the I/O

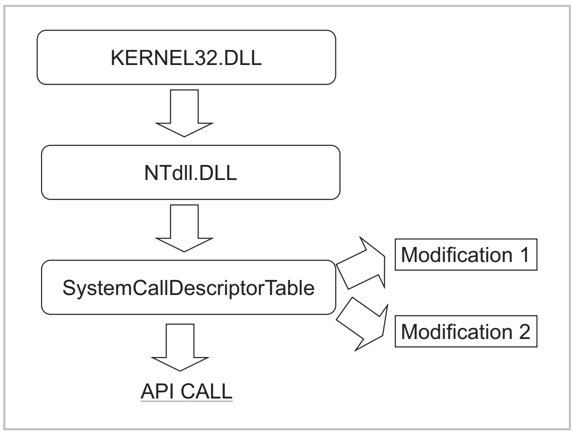


Fig.1 *Monitoring API call sequence by modifying system call table in kernel space*

manager and kernel driver, is a software that is invoked utilizing the existing function provided by Windows in the vicinity of the function driver (existing device driver), which adds, modifies and debugs new functions. By using the filter driver, a Native API Hook can be implemented. The Native API Hook is used to detect events in kernel mode. In user mode, the issuance of API calls is communicated into the kernel space via Ntdll.DLL. In Windows, the API operated in kernel mode is called Native API, which is controlled by a structure called SystemServiceDescriptorTable. Therefore, it is necessary to modify SystemServiceDescriptor to implement the Native API Hook.

Modification 1 is a method used to rewrite the system call table using InterlockExchange. Modification 2 is a method used to rewrite the system call table when loading the driver to pass through the hook function. The modification of the system call table makes it possible to capture the memory access of Windows OS and visualize memory behaviors, among others[6].

2.2.2 API Hook by filter manager

Although this is also influenced by the development environment and the operating situation, the file I/O hooks using normal filter driver sometimes do not operate in a stable manner. Filter driver is a file system filter driver which is newly provided by Microsoft. Filter driver is designed with a focus on appropriately controlling the order and other aspects regarding large numbers of invoked interrupts and API, and streamlining and supporting development by third parties. Figure 2 shows the outline of the function of filter manager. Filter manager serves as an intermediary between the kernel driver (function driver) and filter driver, simplifies the implementation of filter driver, and stabilizes the functions. Filter driver is implemented via filter manager provided by Microsoft, and enables lightweight intermediate processing[7].

2.2.3 API Hook by DLL Injection

DLL Injection is executed when a given function is stored in a library and a specific API call is issued, and is used at the time of debugging of software when a new function is to be added to an existing application. Given the request from the development side, a technology called DLL Injection is also sometimes used, in which a DLL (unique API) is executed by another process at a given timing. The methods of DLL Injection include types in which the functions provided by Microsoft Windows are used, the functions of debugging mechanisms are used, remote threads are used, and the import section of modules is modified. This paper explains the method in which the import section of modules is modified, which made it possible to track software behavior and

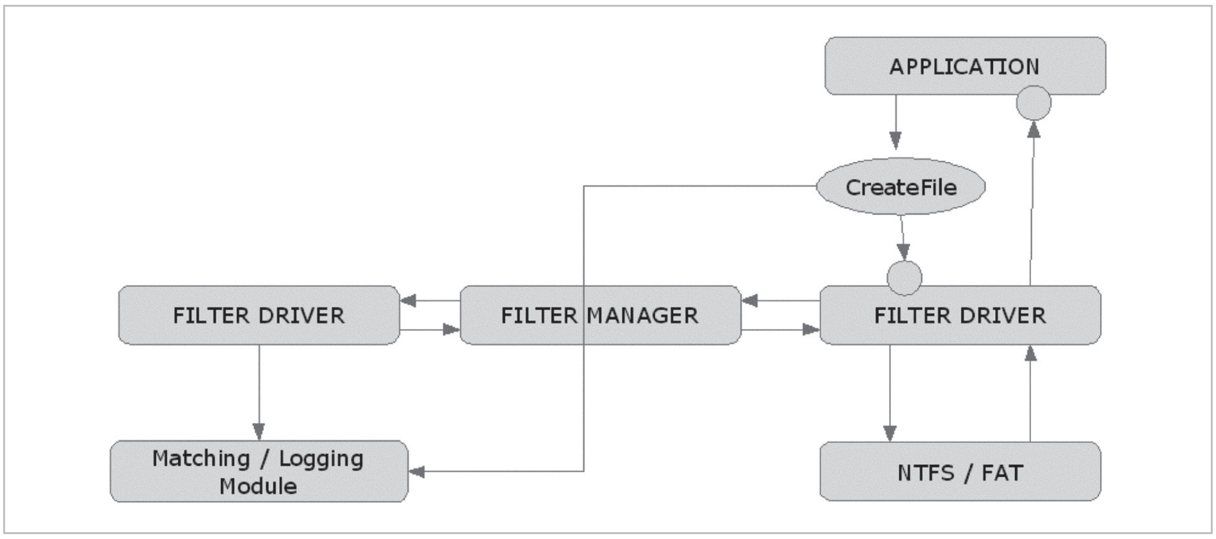


Fig.2 *Logging file access of Windows OS using filter driver*

prevent problematic traffic before its occurrence or arrival, by hooking the data transmitting and receiving functions of the target software and inserting appropriate operation there. As shown in Fig. 1, we injected DLL by modifying the import table. The import table (import section) is a data structure (header) located inside the section table, where the list of DLL addresses necessary before the execution of the program and addresses of symbols inside the DLL to be used is stored. By rewriting the DLL addresses to be hooked into given DLL addresses, software operation can be modified. This method does not depend on the specification of a specific CPU and there is no issue of synchronization of threads, and is therefore often utilized as a very flexible method. The process consists of the following: i) preparing the DLL to be injected, ii) acquiring import table address of the target software, iii) looking for function addresses to be hooked, and iv) rewriting discovered addresses into the addresses for the DLL (function) to be injected. The function form is as follows:

```
void ReplaceIATTableInP2Psoftware
("kernel32.dll",
funcORG,
funcINSERT",
moduleHandler);
```

There are several implementation methods that vary depending on the structure of target process, but the outline is as shown above.

2.3 Output in the proposed system

This section explains visualization as result of clustering the output log of the above-mentioned virtual machine monitoring system using a self-organizing map. The details of capturing registry access of Windows OS and the tabulation process are proposed in [8].

2.3.1 Acquired data and algorithm

This section shows the results of visualizing behavior data of the monitored virtual machine. A registry access log that was made observable was used to visualize the virtual machine. An example of output of the log is shown below.

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

The top line consists of eight values.

1 System time
2 Process information of invoker
3 Name of invoked registry function
4 Return value of function
5 Registry key
6 Registry value
7 Setting value
8 Acquired value

When quantifying registry access logs, we generated a matrix by counting the frequency of occurrence of words in the function name of registry key and registry.

In addition, this study uses a self-organizing map to visualize registry access data. Self-organizing map is one of the unsupervised learning algorithms, and is characteristic in that cluster numbers are not given, unlike k-means method or other methods. The merit of a self-organizing map lies in that it can learn without changing the topology among data. When the competition layer is set in two dimensions, it is possible to visualize while maintaining the phase relation among data input in multiple dimensions.

The behavior expression is as shown below. In the same way as other neural network algorithms, the weight among neurons is modified using the difference between input vector and weight vector.

$$Wv(t+1) = Wv(t) + F(t)a(t)(D(t) - Wv(t))$$

Here "W" indicates the weighting coefficient matrix among nodes, "D" the input vector, and a(t) the learning coefficient. In a self-organizing map, the best matching unit (BMU) is determined against each node. The F(t) in the above expression is neighborhood radius and changes according to the distance from the

BMU. For the threshold value that terminates the learning process, the number of times of learning is used. The details of the visualization of a self-organizing map are explained below.

2.3.2 Visualization of virtual machine behaviors

This paper has evaluated our proposed method by using a self-organizing map to visualize virtual machines whose status is similar to that of virtual machines experiencing security incidents.

Our evaluation experiment categorizes virus and infection behaviors into installations and transmissions (i.e. sending packets). We have prepared data that are similar in their behaviors (i.e. data observed in case applications and device drivers are installed and P2P applications and a Web browser are executed) to categorize and identify behaviors to be monitored. The following are the behavior patterns analyzed by our method.

```
<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
```

Figures 3–5 show the visualization results based on different scenarios. Figure 3 (Pattern I) visualizes and identifies malware infections and non-malicious software installations. Malware installations are located in the center of Fig. 3 and software installations, including installations of applications and device drivers, are located on the right-hand side of the Fig. 3. Figure 4 (Pattern Ⅱ) visualizes and identifies malware infections and the execution of two network applications (i.e. a P2P application and a Web browser). These three cases are separated and placed in three locations on Fig. 4. Malware infections and communication application executions have been visualized and identified. The communication applications include P2P applications and a Web browser, whose activities are located in the center and on the left hand side of Fig. 4. Malware infections are located on the right hand

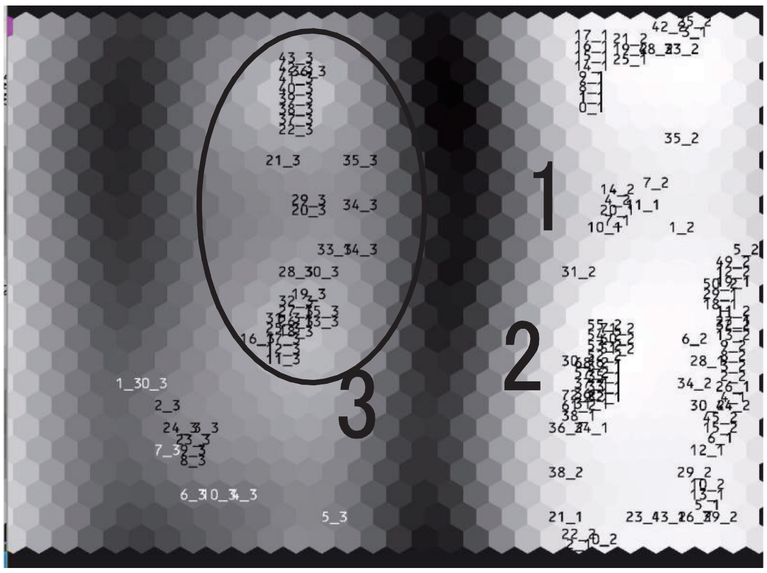


Fig.3 Clustering and visualization of 3 states of Windows OS (installing application, installing device driver and malware infection). States caused by malware are distributed on the left side (**3**) of the figure.

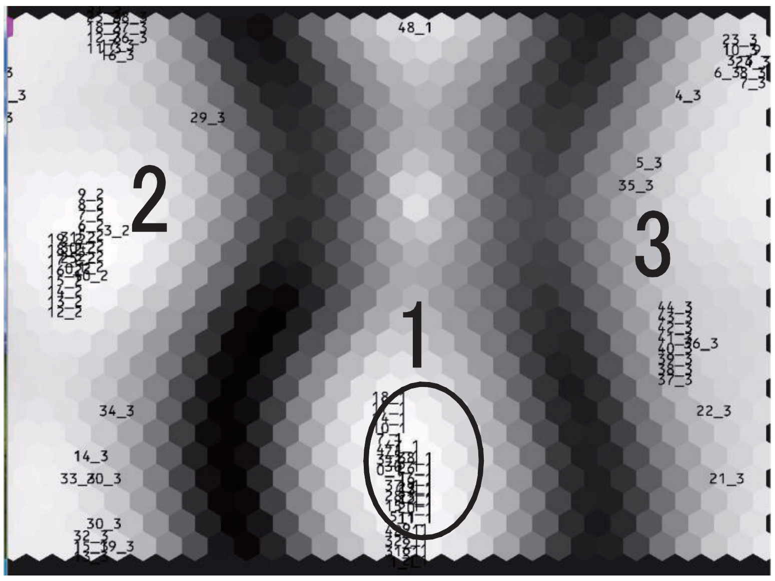


Fig.4 Clustering and visualization of 3 states of Windows OS (P2P application, Web browser (IE) and malware infection). States caused by malware are distributed on the right side (**3**) of the figure which is properly isolated from other two states.

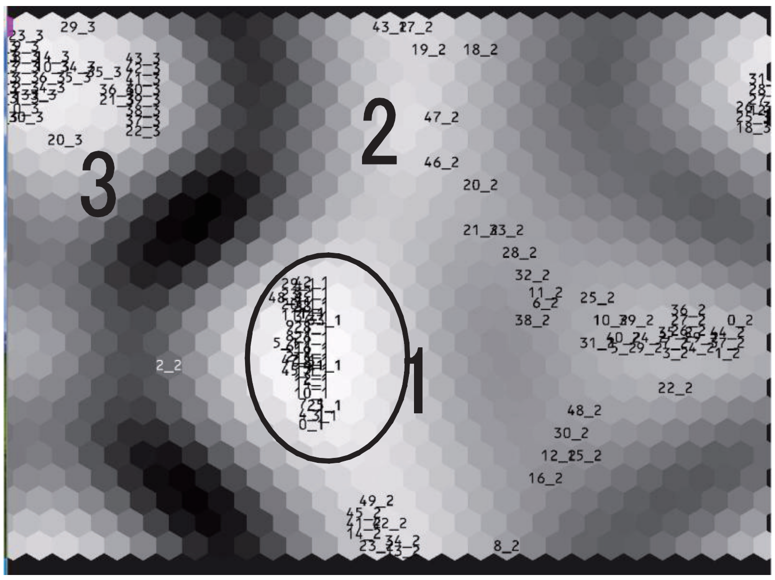


Fig.5 Clustering and visualization of 3 states of Windows OS (P2P application, installing device driver and malware infection). States caused by malware are distributed on the left side (**3**) of the figure which is properly isolated from other two states.

side of Fig. 4. Figure 5 (Pattern Ⅲ) categorizes and visualizes (**1**) the execution of P2P applications, (**2**) the installations of device driver software, and (**3**) malware infections. **1** and **2** are located on the right hand side of center and malware infections are located on the left hand side of Fig. 5.

3 Developing the monitoring technology for overlay networks

P2P networks, where one machine functions both as a server and a client, lead to information leaks and pose many issues to our society. File distribution through P2P networks has been studied in recent years, but we still do not have enough information on in-depth research and lack disclosed information on large-scale file distribution covering foreign countries. As a more widespread usage of P2P software leads to more issues in our society, we need to leverage the visualization of network status to correctly understand its impact. P2P software is often released and used for free, but its information is not disclosed or readily available. This makes it impossible for us to understand how file distribution is done across networks, complicating our response to security incidents. Large-scale P2P networks contain a large volume of data to be monitored, requiring us to design our system to virtualize nodes to be monitored for the enhanced consolidation of our monitoring process and to prevent the resource consumption by the analysis system from hampering or confusing the operations of the monitoring system. This paper illustrates how we have enhanced the capabilities to process a large volume of data based on a large-scale P2P network monitoring system using virtualization technology. We have deployed a protocol using virtual registers as we transfer monitored data to a virtual machine monitor and the host operating system (i.e. VM introspection). This has reduced the frequency of file I/O and memory access within the virtual monitoring system[1]. This mechanism enables the virtual monitoring system to satisfy measurement problem restrictions, allowing us to make the maximum use of resources for network monitoring.

3.1 Structure and characteristics of P2P software

P2P networks, where one computer functions as a server and a client, enable simple communication between users and are highly resistant to changes including a rapid increase in network traffic. These features have rapidly accelerated the utilization of P2P. On the other hand, copyright infringement through P2P networks and information leaks triggered by virus infections on P2P networks are causing many issues. P2P software is frequently released for free, while it often ceases to be upgraded or modified, unlike commercial software, as developer communities are established and developed. Its development may be halted due to various issues. This can make fixes and patches unavailable as we detect new vulnerabilities and attacks against overlay networks created by P2P applications.

In general, P2P software is categorized into three types. What is unique about P2P is that its nodes directly exchange data and files with each other. The first generation P2P (also known as pure P2P) enables a server to control which nodes on networks contain which files. The second generation P2P allows nodes to communicate data with each other without servers. The third generation P2P software is equipped with a caching feature to enhance anonymity.

P2P software is typically composed of four modules: node management, query management, key management, and task management. The critical information required to track P2P software behaviors is key tables, node tables, transmitted file tables. We used these data deployed on virtual memory to hook and capture as well as analyze decoding and filtering API calls used to send packets. The hooking methodology is described in Chapter **2**.

P2P applications are designed to be used by multiple users, making it difficult for us to easily rewrite their programs. In addition, upgrades and patches are not guaranteed for them and development can be halted due to various reasons. As we use software with these characteristics in a safe manner, it is valid for us to modify it using the debugging technology explained in this paper. The second important characteristic of P2P applications is that we cannot capture network topologies created by them. This is because specific probe systems placed on P2P networks can be targeted for attacks and broadcasting is not used for the same reason. This makes it difficult to monitor based on servers and requires us to obtain the information on process memory on each probe so that we can capture topologies and traffic flows. To meet this objective, a system is proposed to modify P2P applications by using various technologies, including DLL Injection explained in Chapter **2**, and analyze their traffic[9].

3.2 Monitoring and processing P2P network traffic

P2P network traffic is more likely to fluctuate and harder to predict than routine and Web applications. We also have to deal with rapid increases in volume within a short period of time and add storage and processors. To counter this situation, the analysis system should be placed outside of virtual nodes (i.e. on the host OS). If the analysis system is placed inside nodes, we have not yet come up with features to dynamically add storage and processors. This paper proposes a system transmitting logs from virtual probes to the host OS through virtual registers and enabling the host OS directly controlling physical machines to enhance the analysis process.

Large-scale P2P networks contain a large volume of data to be monitored, requiring us to virtualize nodes to be monitored for the enhanced consolidation of the monitoring process. However, the consolidation does not intend to reduce the traffic triggered by each probe and increasing volumes of monitoring data enhance loads on the analysis process. Thus, placing monitoring and analysis features on the same probe can lead to the analysis process load impacting the monitoring feature.

Based on the P2P traffic monitoring conditions explained in Section **3.1**, this paper proposes a method separating the monitoring system from the analysis system by virtualization technology. This will enable us to flexibly change storage and processor capacities of the host OS in line with an increase or change in output logs from the monitoring system. A detailed methodology on using VM monitoring technology (i.e. VM Introspection) to process large-scale and large-volume traffic data for P2P networks is proposed by [10].

3.3 Creating probes with virtual machines

Figure 6 illustrates a system transmitting

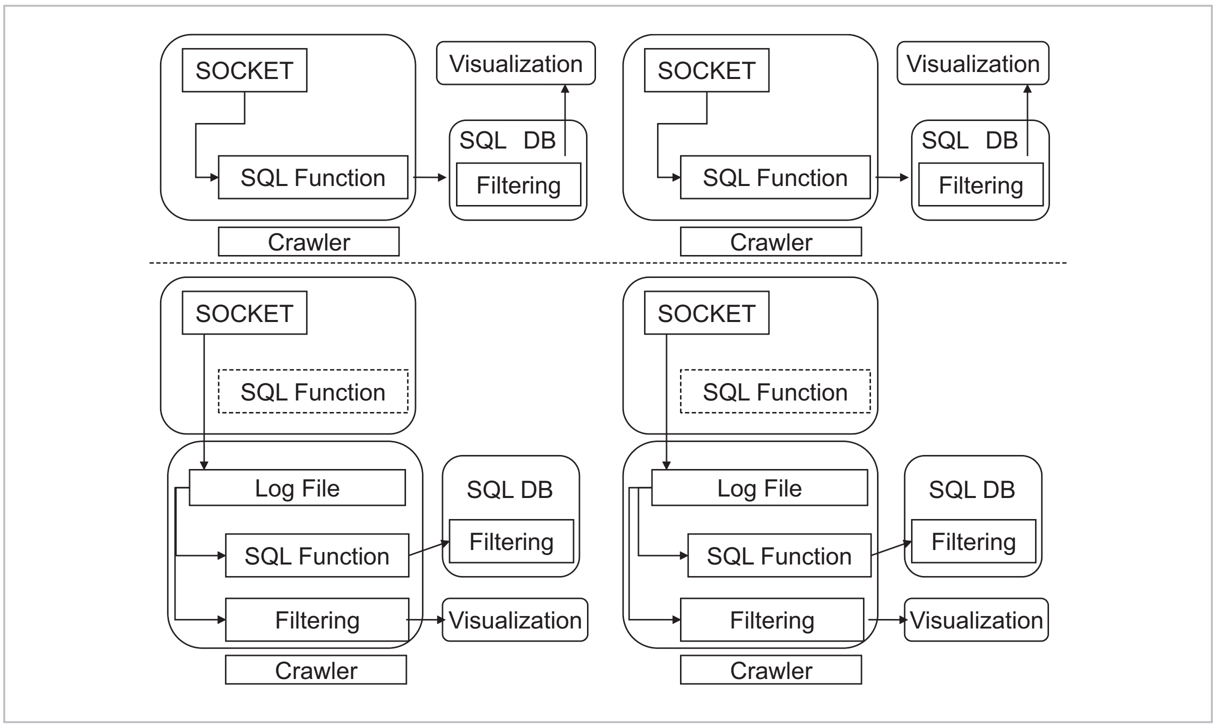


Fig.6 *System architecture for logging and storing large scale data of P2P monitoring*

the probes output by the guest OS through virtual registers to the host OS. It converts obtained monitoring information into numerical data at the time of P2P communication events (e.g. socket access and file access), places it in the context of virtual CPUs, and moves the control to virtual machine monitors. It obtains the context of virtual CPUs within the handlers of virtual machine monitors or the host OS and recreates it as output logs. Using virtual registers enables us to minimize the memory or file access by the monitoring system of the virtual OS and transmit monitoring data to the host OS (i.e. physical machines) we can configure in the system[11].

Large-scale P2P networks provide a large volume of data to be monitored, requiring monitoring nodes to be virtualized for the enhanced consolidation of the monitoring process. The proposed system enables provisioning for the monitoring process whose traffic logs dynamically fluctuate and are hard to predict.

3.4 Visualization of P2P network traffic

This section explains how we have monitored large-scale traffic and represented the results of stored data as explained in Section **3.3**. It also describes how we have categorized the detected nodes and focuses on the detection and visualization of super nodes.

3.4.1 Leveraging Keyhole Markup Language (KML)

KML is an XML-based markup language developed to visualize 3D geographical information. KML enables us to visualize points, lines, images, polygons, and other signs on a map and to share geographical information with other users using Google Earth. Currently KML 2.2 is adopted as a standard by Open Geospatial Consortium, Inc. (OGC). Our visualization system converts monitored data into a KML-based file format to enable data visualization based on Google Earth[12].

Figure 7 visualizes P2P (i.e. Winny) networks monitored in a specific period based on passive monitoring. We have color-coded detected nodes based on their size, converting their IP addresses into spatial information (i.e. longitude and latitude) and visualizing the converted information by storing it into a KML file. We can also create animated images

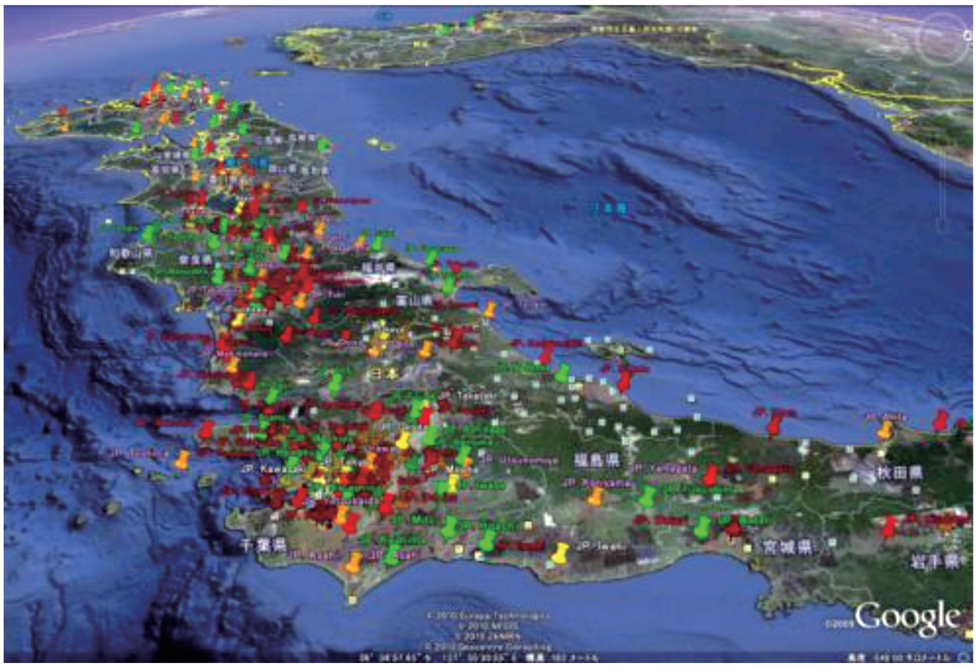


Fig.7 *Plots are colored according to the scale of node*

by successively showing images from multiple P2P networks using the TimeSpan tag.

Figure 8 shows the visualized image of P2P network nodes around Koganei City. The detected number of P2P network nodes reaches from hundreds of thousands of nodes to millions of nodes, making it impossible for us to visually check the node distribution of each area by creating a nation-wide visualized image. Thus, our visualization system enables us to confirm the node distribution of each area through its scaling feature.

3.4.2 Detecting a super node and visualizing nodes

In general, P2P networks include nodes where file distribution information is consolidated. Detecting and preventing the propagation of leaked files on P2P networks requires us to detect these nodes with consolidated information. Figure 9 shows the layered structure of super nodes and other nodes located on Winny networks. As leaked files are stored on super nodes, propagation speed will rapidly accelerate. Thus, our P2P monitoring system has an algorithm to detect super nodes out of the data containing detected nodes.

Generally speaking, pure P2P networks create large-scale super nodes, where information, files, and queries are converged. Many super nodes are linked to other multiple nodes. Thus, super nodes not only consolidate information on networks but can rapidly propagate their stored files. To counter this, our information leak monitoring and tracking system detects super nodes distributed on networks and rapidly captures network distribution status as well as prevents future information leaks more effectively.

Figure 10 visualizes a super node located in Tokyo. It shows many nodes color-coded based on their size (i.e. red, green, and orange) linked from the hub. This super node consolidates distribution information. It is critical for us to detect and monitor a super node so that

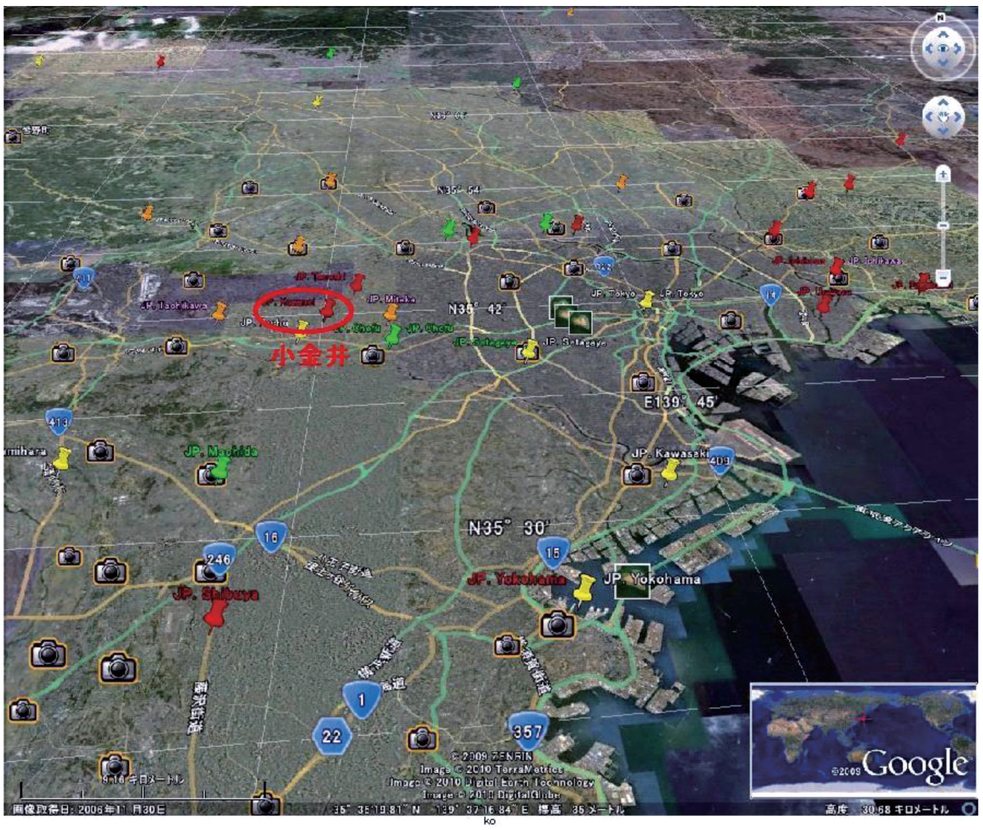


Fig.8 Visualization of P2P networks around Koganei City (sample)

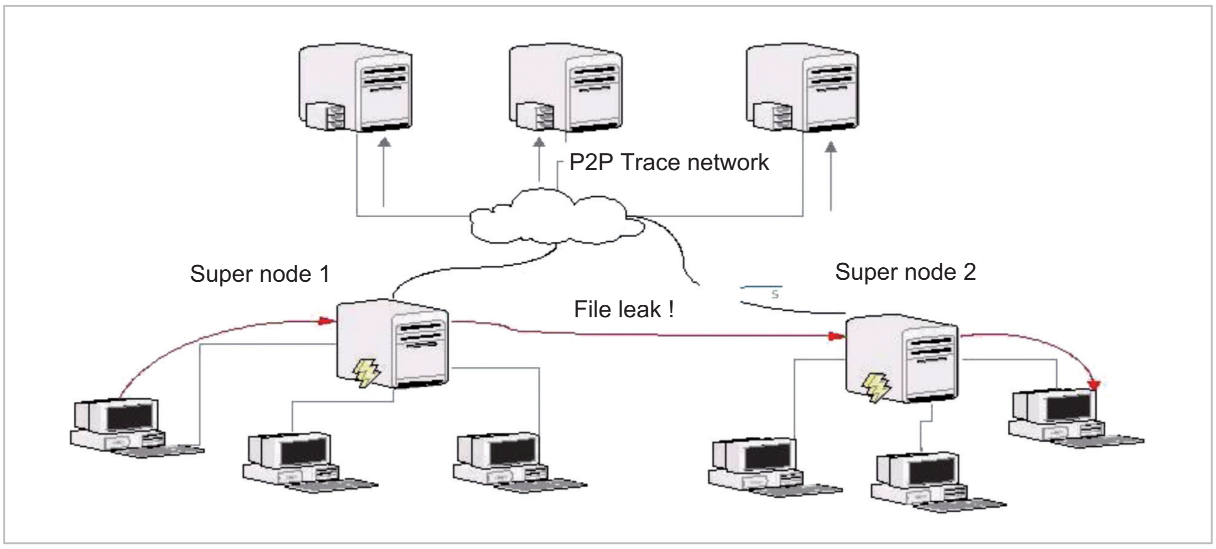


Fig.9 *File leakage over super node*

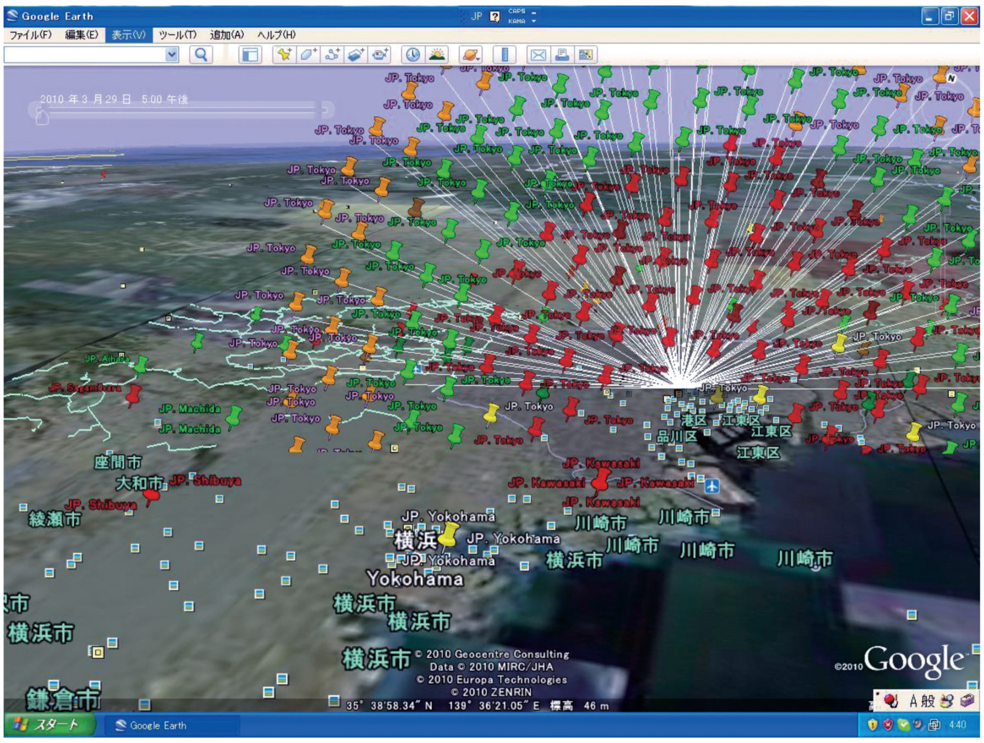


Fig.10 *Visualization of super node in Tokyo*

we can monitor information leaks in a timely manner.

4 Conclusion

As networks are increasingly widespread and cloud computing and P2P network technologies are more broadly utilized, we have been using virtual networks composed of virtual machines more frequently. Our medium-term plan in the second term has focused on improving the accuracy of intrusion detection and enhancing monitoring networks as part of research and development of traceable network technology. In our medium-term plan of the previous term, we mainly researched the monitoring of virtual machines and P2P networks to make our traceable network monitoring technologies more accurate and to expand our monitoring ranges.

One of the reasons triggering the current rapid penetration of cloud computing is the advancement of virtualization technology. Irrespective of service types (e.g. SaaS, PaaS, and IaaS), we can define cloud computing as a service enabling the utilization of virtual machines. Based on this understanding, we have been engaged in research on probing technology since making new types of systems including cloud computing more secure requires the monitoring of virtual machines.

Furthermore, information leakage on overlay networks including P2P networks began to be noticed as one of the most important social issues around 2006, during the medium-term plan period of the previous term. This prompted us to develop traceability technologies of this event as a way to research traceable networks. We also enhanced the capability of analysis processing since the data obtained from P2P networks for tracking became very large.

Generally speaking, systematic monitoring and intrusion detection technologies based on virtualization technology is called virtual machine introspection. An IDS deployed on virtual machines keeps monitoring objects and probes separated by virtualization technology, which enables new features not available through conventional IDSes. It has been pointed out that conventional NIDSes are harder to be detected by attackers but their data is less accurate and that HIDSes are more visible for attackers but their data is more accurate. A system design is being proposed so that we can leverage the characteristics of VMM and make the maximum use of these contradicting features. Advantages available through the use of VMM when we create an IDS can be summarized as Isolation (i.e. probes are placed separated from the guest OS), Inspection (i.e. detailed data including events made available in the kernel space), and Interposition (i.e. enabling system calls, interruptions, and I/O requests to VMM to be hooked and intermediate actions to be inserted at the same time). Thus, we have created a system to enable intrusion detection based on these features through our R&D efforts.

The protocol analysis and virtualization technologies applied to these monitoring systems will be further utilized for the safety analysis of cloud computing environments, the monitoring of outside environments, and the recurrence verification on our Testbeds, all of which are going to be conducted by the Security Architecture Laboratory based on the medium-term plan of the current project period.

References

1 Ruo Ando, Youki Kadobayashi, and Youichi Shinoda, “Asynchronous Pseudo Physical Memory Snapshot and Forensics on Paravirtualized VMM Using Split Kernel Module,” ICISC 2007, The 10th International Conference on Information Security and Cryptology, November 29–30, Seoul, Korea.

2 Anh-Quynh Nguyen, Ruo Ando, and Yoshiyasu Takefuji, “Centralized Security Policy Support for Virtual Machine,” LISA 2006: 79–87.

3 Ruo Ando, Youki Kadobayashi, and Yoichi Shinoda, “Incident-driven checkpointer on full-virtualized VMM using Kernel Virtual Machine,” IPSJ Computer Security Symposium 2007, Oct. 2007.

4 Paul Barham, Boris Dragovic, Keir Fraser, Steven Hand, Timothy L. Harris, Alex Ho, Rolf Neugebauer, Ian Pratt, and Andrew Warfield, “Xen and the art of virtualization,” SOSP 2003, pp. 164–177.

5 Tal Garfinkel and Mendel Rosenblu, “A Virtual Machine Introspection Based Architecture for Intrusion Detection,” In the Internet Society’s 2003 Symposium on Network and Distributed System Security, NDSS 2003, pp. 191–206.

6 Ruo Ando, Nguyen Anh Quynh, and Kuniyasu Suzaki, “A visualization of memory state transition of Windows OS exploiting virtual machine introspection,” IPSJ SIG technical reports 2009-OS-112.

7 Ruo Ando, Nobuko Inoue, and Kuniyasu Suzaki, “An implementation of secure access control on Windows OS using filter driver,” IPSJ Computer Security Symposium 2009, Oct. 2009.

8 Ruo Ando, Kazushi Takahashi, and Kuniyasu Suzaki, “A SOM based malware visualization system using resource access filter of virtual machine,” The 2011 International Conference on Computers, Communications, Control and Automation (CCCA 2011), Hong Kong, China, February 20–21, 2011.

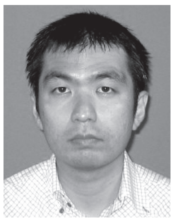
9 Ruo Ando, Youki Kadobayashi, and Yoichi Shinoda, “A large scale P2P network monitoring using virtual register based introspection,” IPSJ Computer Security Symposium 2010, Oct. 2010.

10 Ruo Ando, Hideo Toyama, and Youki Kadobayashi, “Tracing behavior of P2P software using DLL injection,” IPSJ SIG technical reports 2007-CSEC-36.

11 Ruo Ando, Youki Kadobayashi, and Yoichi Shinoda, “Blink: Large-scale P2P Network Monitoring and Visualization System Using VM introspection,” NCM 2010: 6th International Conference on Networked Computing and Advanced Information Management, August 16–18, Seoul, Korea, 2010.

12 Ruo Ando, Hideo Toyama, Youki Kadobayashi, and Yoichi Shinoda, “P2P Network geographical visualization using passive monitor and Google Earth,” IPSJ SIG technical reports 54, 1, 2010-05-20.

(Accepted June 15, 2011)



ANDO Ruo, *Ph.D.*

Senior Researcher, Security Architecture Laboratory, Network Security Research Institute

Network Security, Software Security