HyperSpector: Virtual Distributed Monitoring Environments for Secure Intrusion Detection

Kenichi Kourai
Department of Mathematical and Computing Sciences
Tokyo Institute of Technology
2-12-1 Ookayama, Meguro-ku, Tokyo, Japan
kourai@is.titech.ac.jp

Shigeru Chiba
Department of Mathematical and Computing Sciences
Tokyo Institute of Technology
2-12-1 Ookayama, Meguro-ku, Tokyo, Japan
chiba@is.titech.ac.jp

ABSTRACT

In this paper, a virtual distributed monitoring environment called *HyperSpector* is described that achieves secure intrusion detection in distributed computer systems. While multiple intrusion detection systems (IDSes) can protect a distributed system from attackers, they can increase the number of insecure points in the protected system. HyperSpector overcomes this problem without any additional hardware by using virtualization to isolate each IDS from the servers it monitors. The IDSes are located in a virtual machine called an IDS VM and the servers are located in a server VM. The IDS VMs among different hosts are connected using a virtual network. To enable legacy IDSes running in the IDS VM to monitor the server VM, HyperSpector provides three inter-VM monitoring mechanisms: *software port mirroring*, *inter-VM disk mounting*, and *inter-VM process mapping*. Consequently, active attacks, which directly attack the IDSes, are prevented. The impact of passive attacks, which wait until data including malicious code is read by an IDS and the IDS becomes compromised, is confined to within an affected HyperSpector environment.

Categories and Subject Descriptors

D.4.7 [**Operating Systems**]: Organization and Design—*distributed systems*

General Terms

Design, Security

Keywords

virtual machine, virtual network, inter-VM monitoring, distributed IDS

VEE'05, June 11-12, 2005, Chicago, Illinois, USA.

1. INTRODUCTION

As distributed computer systems become larger and more complex, self-protection against attacks is becoming an indispensable attribute. To achieve self-protection, distributed systems have to monitor themselves, analyze the information obtained, plan a defense against detected attacks, and execute the plan [12]. Of these four steps, we focused on self-monitoring, which is the important first step. One type of self-monitoring system is the distributed IDS (DIDS), a security system designed to detect suspicious activity with a system [20]. The DIDS uses multiple IDSes to protect a distributed system. The information they obtain is collected by a designated host to analyze the characteristics of an attack. The results are then shared among the IDSes to enable them to work together effectively.

Needless to say, the DIDS itself must not make the monitored distributed system less secure. However, the DIDS can have insecure points that must be protected against attacks. Attacks against an IDS are classified as active or passive. Active attacks are those in which the attacker takes actions designed to directly compromise an IDS. Passive attacks are those in which the attacker generates data that contains malicious code and waits until an IDS becomes compromised when it reads the data. Passive attacks against a DIDS are difficult to prevent because it must check all data as part of its function. Therefore, secure intrusion detection requires (1) protecting the IDSes from active attacks and (2) preventing passive attacks against the IDSes from affecting the protected system.

One approach is to use hardware. For example, in isolated monitoring, network-based IDSes (NIDSes) are located in hosts physically separated from the server hosts and are connected to a physically independent network. To enable monitoring of network packets sent from and to the server hosts, the NIDS hosts are connected to the server hosts via a network switch with the port mirroring feature. This protects the NIDS hosts from active attacks and confines the impact of passive attacks to within the NIDS hosts because the NIDSes are isolated from the servers. However, the cost is high because a large amount of hardware is needed, e.g. NIDS hosts, a switch for connecting them, and a switch with the port mirroring feature. In addition, it is difficult to apply this approach to host-based IDSes (HIDSes) in terms of security and efficiency.

Our proposed virtual distributed monitoring environment, *HyperSpector*, achieves isolated monitoring without any ad-

ditional hardware, even for legacy HIDSes. HyperSpector isolates the IDSes from the monitored servers by using virtual machines (VMs) and a virtual network. The IDSes for each host are located in a VM (IDS VM) and the servers are located in a server VM, instead of being located in different hosts. The IDS VMs are connected using an independent virtual network. To isolate multiple DIDSes from each other, multiple HyperSpector environments can also be created.

To enable legacy IDSes running in the IDS VM to monitor the separated server VM securely, HyperSpector provides three inter-VM monitoring mechanisms in a virtual machine monitor: *software port mirroring*, *inter-VM disk mounting*, and *inter-VM process mapping*. Software port mirroring enables the IDS VM to capture packets sent from and to the server VM. Inter-VM disk mounting virtually mounts the file system of the server VM on the IDS VM to check the integrity. Inter-VM process mapping allows the IDS VM to probe the behavior of the processes in the server VM.

HyperSpector protects an IDS from active attacks. The server VM prevents an attacker from compromising the IDS VM if a server in the server VM has been compromised. The virtual network connecting only the IDS VMs prevents an outside attacker from sending illegal packets to the IDS VMs. For passive attacks against an IDS, HyperSpector confines the impact to within one HyperSpector environment. The IDS VM prevents an attacker from interfering with the server VM. The virtual network prevents the attacker of an IDS VM from sending illegal packets outside the HyperSpector environment. With multiple HyperSpector environments, passive attacks against one DIDS do not affect the other DIDSes. In addition, the inter-VM monitoring mechanisms do not degrade security because they allow only the IDS VM to monitor the server VM.

The rest of this paper is organized as follows. Section 2 describes the type of attacks against IDSes and previous architectures for secure intrusion detection. Section 3 presents HyperSpector, which can isolate IDSes from servers by using virtual machines and a virtual network, and describes the inter-VM monitoring mechanisms. Section 4 explains the details of our implementation of HyperSpector. Section 5 describes our experiments for determining the security provided by using HyperSpector and the overhead it imposes. Section 6 discusses related work and Section 7 concludes the paper with a brief summary and a look at future work.

2. OBSTACLES TO SECURE INTRUSION DETECTION

To secure a distributed computer system against attack by using a DIDS, each IDS must resist attacks against itself because if an IDS is compromised, the attacker can attack the rest of the host without being detected. Even if the attacker cannot take over high privileges, the attacker can still steal sensitive information from the server monitored by the IDS. If the attacker then restarts the IDS after invalidating some policies, the administrator will have trouble detecting the penetration.

2.1 Types of Attacks

Attacks against IDSes are classified as either active or passive. In active attacks, the attacker takes actions directly against an IDS. IDSes can be attacked via a compromised server in the same host (local active attacks). In many cases, it is easier to attack servers than IDSes because servers must accept requests from anonymous users. Therefore, the first target of an attacker is usually a server. Once an attacker compromises a server, the next target is usually the IDSes monitoring the server because if the attacker fails to stop the monitoring, the attack will soon be detected. To stop the IDSes, the attacker may try to terminate the IDS processes or may rewrite policy files used by IDS programs so as to invalidate the IDS functions.

Also, IDSes can be attacked via the network from the Internet or compromised server hosts (remote active attacks). When multiple IDSes are used, each one communicates with various hosts to facilitate cooperation. For example, an IDS may receive commands or monitoring policies from a central server or may transfer its log to a central server. In some configurations, the network ports used for these communications may be open to the outside. In Counterpane [4], the distributed computer system needs to communicate with an external security operation center. In Big Brother [18], the vulnerability allowed a remote attacker to read arbitrary files [5]. Even if the IDSes do not communicate with the outside, the network ports are open to the whole distributed system at least. Therefore, if a server in the system is compromised by an attack, the network ports of the IDSes can be attacked via the compromised server.

Passive attacks, on the other hand, are attacks in which the attacker carefully crafts data so as to compromise the IDSes and waits until an IDS reads the data for monitoring purposes and thereby becomes compromised. A typical example of this type of attack is one designed to exploit buffer overflow vulnerabilities. For NIDSes, the attacker can send carefully crafted packets to servers so that the NIDSes are compromised when they analyze the packets [3]. Such attacks against NIDSes can be done both from the Internet and via hosts compromised in a distributed system. For HIDSes, the attacker can create a file with a carefully crafted name so that the HIDSes are compromised when they examine the file [6]. Passive attacks are more difficult to protect against than active attacks because IDSes must, in the course of their operation, read data from external sources in order to monitor the servers. Therefore, it is important to confine the impact of such attacks.

2.2 Isolated Monitoring

Isolated monitoring can be used to resist such attacks. In this approach, multiple NIDSes are located in Figure 1. Each NIDS is located in a host different from the one it is monitoring. Although the number of NIDS hosts may be less than that of the server hosts, many NIDS hosts are needed in a large distributed system. The server and NIDS hosts are connected to a front-end switch with the port mirroring feature and the switch is connected to the Internet. The server hosts are connected to normal ports, and the NIDS hosts are connected to mirroring ports. The port mirroring feature duplicates packets sent from and to the server hosts and forwards them to the appropriate NIDS hosts. In Cisco switches, this feature is called the switched port analyzer. This feature enables the NIDSes to monitor packets sent from and to the server hosts from remote hosts. In addition, each NIDS host has an extra network interface card (NIC) and is connected to a back-end switch.

With such isolated monitoring, local active attacks against

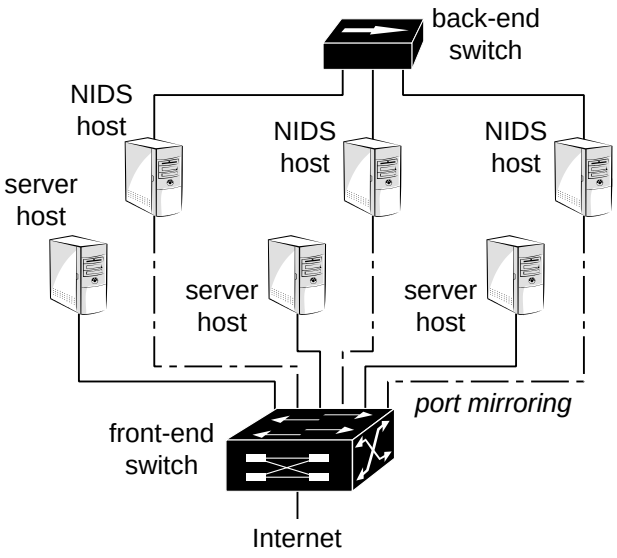

Figure 1: Isolated monitoring using NIDSes.

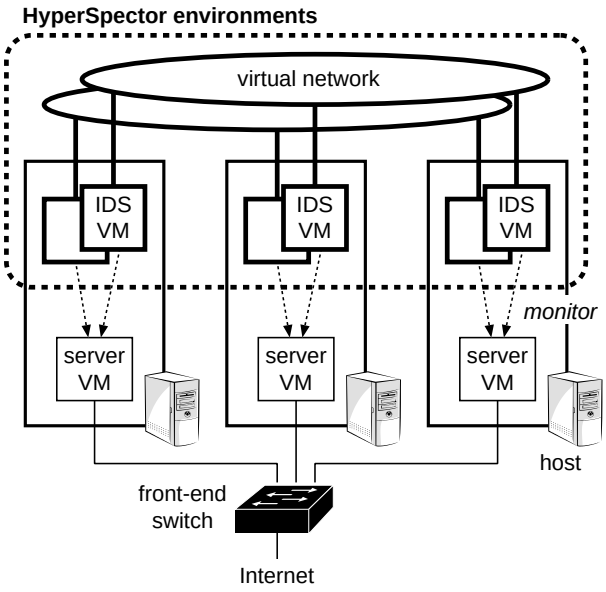

Figure 2: HyperSpector architecture.

NIDSes via compromised servers are prevented since servers cannot access the file systems and processes of NIDS hosts directly. NIDS hosts are also protected from remote active attacks. An attacker cannot directly send packets to NIDS hosts through the mirroring ports in the front-end switch because the mirroring ports support only monitoring. Moreover, the NIDS hosts can communicate with the other NIDS hosts only through the back-end switch, so an attacker cannot send packets to the network of NIDS hosts. The impact of a passive attack is confined within the affected DIDS. Even if an NIDS host is compromised, it cannot attack the server hosts and hosts in the Internet because it cannot send packets to a mirroring port in the front-end switch, which is connected to the servers and the Internet. Active attacks are thus prevented and the impact of passive attacks is confined.

However, this isolated monitoring increases the amount of hardware such as an NIDS host and a network switch. As the amount of hardware increases, the management cost also increases. Moreover, a costly switch with the port mirroring feature is needed, and half of the network ports are reserved for monitoring. Although it is possible to use an inexpensive dumb hub, which enables packet monitoring, it causes a performance bottleneck due to the shared bus architecture. In addition, if there are multiple DIDSes and the administrator would like to localize the impact of passive attacks to only one, even more hardware is needed to locate each NIDS in a different host.

Moreover, it is difficult to use isolated monitoring when there are legacy HIDSes. Legacy HIDSes cannot run in hosts separated from the server hosts because they are designed to monitor file systems, processes, the operating system, etc. in the same host. To enable HIDSes to monitor server hosts from other hosts, the Backdoors architecture has been proposed [2]. Using Backdoors, a distributed computer system can be protected by locating HIDSes like NIDSes in Figure 1. While Backdoors does not need the port mirroring feature in the front-end switch, a programmable NIC is needed for each server host to transfer monitored data.

Although the Backdoors architecture isolates the HIDSes from the servers, it is not secure enough because HIDS hosts can suffer remote active attacks. Since they must be connected to normal ports in the front-end switch to receive monitored data from the server hosts, illegal network packets can be delivered to HIDS hosts from, for example, the Internet. Also, if an HIDS host suffers a passive attack and becomes compromised, it can be used to attack server hosts and hosts in the Internet. Moreover, to monitor server hosts from HIDS hosts, the server hosts must export monitoring interfaces to the network, making them insecure. In addition, the performance of the HIDSes can be degraded when receiving a large amount of data such as the contents of all files from server hosts.

3. HYPERSPECTOR

Our proposed virtual distributed monitoring environment, *HyperSpector*, overcomes the disadvantages of isolated monitoring. Using virtual machines and a virtual network, it can perform isolated monitoring without any additional hardware. With its secure monitoring mechanisms, HyperSpector achieves the same level of security as hardware-supported isolated monitoring, even when legacy HIDSes are being used, as long as HyperSpector itself does not have any vulnerabilities.

3.1 Architecture

As illustrated in Figure 2, HyperSpector runs the IDSes and servers for each host in isolated VMs. HyperSpector can also run each IDS in a different VM so that one IDS does not affect the other IDSes in the same host. These VMs run on top of the base system, which does not use network. Each VM provides independent network system space, file system space, and process space and is isolated from the other VMs and the base system. Our VM for IDSes is called an *IDS VM*, and our VM for servers is called a *server VM*. An IDS VM can monitor the server VM in the same host without using a physical network. Unlike with isolated monitoring, extra network ports in the front-end switch are not required

for this. Moreover, the NIDSes do not also need a costly front-end switch with the port mirroring feature. Even if the IDSes need to monitor a large amount of data sent from and to the server VM, the data can be transferred from the server VM to the IDS VM efficiently without passing through the front-end switch.

A virtual network is used to connect the IDS VMs among different hosts. It is constructed above the base network, such as a LAN, and requires no extra hardware. If one host contains multiple IDS VMs, multiple virtual networks are constructed for each IDS VM. The virtual network is isolated from the server VMs. Illegal packets received from outside the virtual network are discarded by message authentication. The IDS VMs cannot send packets outside the virtual network and cannot change their network configuration so as to communicate with the other hosts without using the virtual network.

A HyperSpector environment consists of the IDS VMs among different hosts and the virtual network connected them. It is used for running a DIDS and independent of the rest of the system. To isolate multiple DIDSes from each other, multiple HyperSpector environments can be created.

3.2 Attack Resistance

HyperSpector protects IDSes from active attacks. First, the server VM prevents an attacker from attempting local active attacks against the IDS VM or the base system in the same host after compromising a server. An attacker cannot access file systems and processes outside the server VM even if root privileges are obtained in the server VM. Second, remote active attacks against the IDS VM are prevented by the virtual network, which connects only the IDS VMs. The virtual network denies access from any server VM and from any host outside HyperSpector. Therefore, the IDS VM cannot receive illegal network packets from outside the virtual network.

HyperSpector confines the impact of passive attacks to within one HyperSpector environment. Even if an attacker succeeds in a passive attack against an IDS, the IDS VM prevents the attacker from interfering with the file systems and processes of the server VM and the base system in the same host. However, the attacker can attack IDS VMs on the other hosts in the same HyperSpector environment because the compromised IDS VM can communicate with them using the virtual network. The attacker cannot attack hosts outside the HyperSpector environment because only the virtual network connecting the IDS VMs can be used. HyperSpector prevents the attacker from sending packets without using the virtual network. In addition, the attacker cannot extend the virtual network to connect it to the attacker's host. In any case, if an IDS VM is compromised by a passive attack, the administrator can restart the HyperSpector environment including it easily since it is isolated from the rest of the system.

The impact of passive attacks can be mitigated by using multiple HyperSpector environments. For example, the administrator can configure two HyperSpector environments, one for the NIDSes and one for the HIDSes. Even if an attacker manages to compromise a server by an active attack and one of the HyperSpector environments by a passive attack, the IDSes in the uncompromised environment can continue intrusion detection.

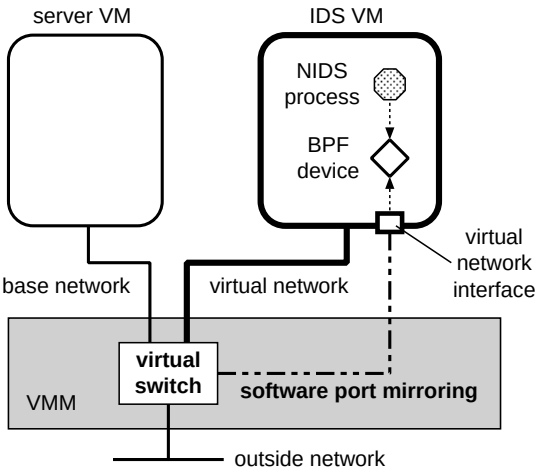

Figure 3: Network monitoring using the virtual switch with the software port mirroring feature.

3.3 Inter-VM Monitoring Mechanisms

As described above, HyperSpector completely isolates the IDSes and servers by using virtual machines and a virtual network. To enable legacy IDSes in the IDS VM to monitor the servers, mechanisms are needed that provide interfaces to legacy IDSes and provides secure monitoring between the IDS and server VMs. HyperSpector satisfies both requirements by providing three inter-VM monitoring mechanisms, which are called software port mirroring, inter-VM disk mounting, and inter-VM process mapping, in a virtual machine monitor (VMM).

3.3.1 Software Port Mirroring

As illustrated in Figure 3, the server VM network, the IDS VM network, and the outside network are connected to the *virtual switch* in the VMM. When the virtual switch receives a network packet, it delivers the packet to the appropriate VM. The virtual switch does not deliver packets sent between the IDS VM and the server VM.

The software port mirroring feature in the virtual switch enables the IDS VM to capture all the packets that the server VM sends and receives. In addition to being connected to a normal port in the virtual switch, the IDS VM is connected to a mirroring port via a *virtual network interface*. All the packets sent from and to the server VM are duplicated and forwarded to the mirroring port in the virtual switch. They are then transferred to the virtual network interface in the IDS VM. The IDS VM captures the forwarded packets using a legacy packet filter device, such as a Berkeley packet filter (BPF). For efficiency, if the IDS VM is not monitoring the virtual network interface, packets are not forwarded. If multiple IDS VMs are using their virtual network interfaces, packets are forwarded to those IDS VMs. Instead of using software port mirroring, the network interface of the IDS VM can be set to the promiscuous mode. However, doing so imposes higher overhead because the IDS VM receives not only packets sent from and to the server VM but also those sent from and to the IDS VMs.

Software port mirroring allows the IDS VM to only monitor the server VM. A virtual network interface cannot be created in the server VM, so the server VM cannot capture

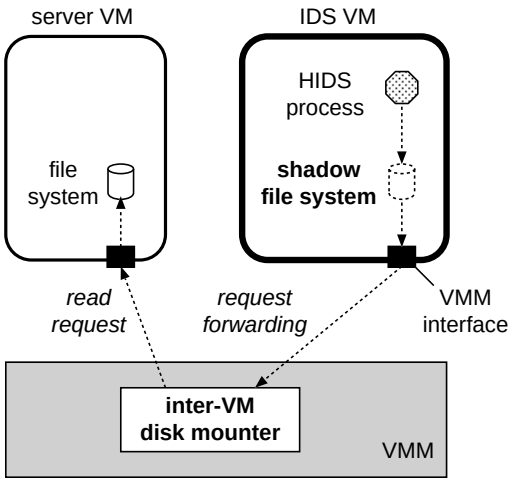

Figure 4: File system monitoring using the inter-VM disk mounter.

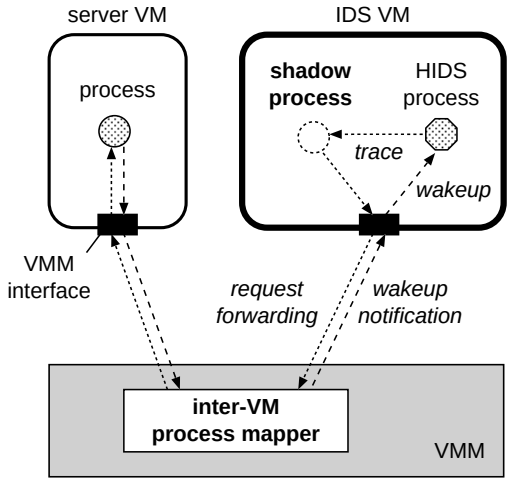

Figure 5: Process monitoring using the inter-VM process mapper.

packets sent from and to the IDS VM. Moreover, the mirroring ports in the virtual switch can only be used for forwarding packets sent to the normal ports; all packets directly sent to the mirroring ports are discarded by the virtual switch. Therefore, even if an IDS VM is compromised by a passive attack, the attacker cannot send packets to the server VM via a mirroring port. On the other hand, if the IDS VM is compromised, the attacker can view sensitive information in the server VM by using this monitoring mechanism. However, even if the attacker manages to do this, the information cannot be sent outside the HyperSpector environment due to the use of virtual machines and a virtual network.

3.3.2 Inter-VM Disk Mounting

Inter-VM disk mounting enables the IDS VMs to check the integrity of the file system of the server VM. The IDS VM cannot locally mount the disk of the server VM directly because the disk used by the server VM is not accessible to the IDS VM. Also, the IDS VM cannot mount the file system of the server VM using the network file system such as NFS because the IDS VM cannot communicate with the server VM directly. We thus developed an *inter-VM disk mounter* running outside the VMs. The mounter virtually mounts the file system of the server VM on the IDS VM as a *shadow file system*. The IDS VM can mount the shadow file system on an appropriate directory.

The inter-VM disk mounter mediates between the monitored file system of the server VM and the shadow file system of the IDS VM, as illustrated in Figure 4. When the IDSes in the IDS VM access files or directories in the shadow file system via legacy system calls such as `read`, the IDS VM forwards the access request to the mounter using its *VMM interface*. The mounter forwards the request to the VMM interface of the server VM. The server VM handles the request and returns the response to the IDS VM.

Inter-VM disk mounting allows the IDS VM to only monitor the file system of the server VM. A mount request to the IDS VM from the server VM is denied by the mounter. Requests that will result in modification of the file system such as write requests are denied even if they are issued by the IDS VM. Therefore, the attacker of a compromised IDS VM cannot rewrite the file system of the server VM via the shadow file system.

3.3.3 Inter-VM Process Mapping

Inter-VM process mapping enables the IDS VM to probe the processes in the server VM, performing process tracing and getting process status. The IDS VM cannot monitor the processes in the server VM directly because the two VMs have different process spaces. We thus developed the *inter-VM process mapper* running outside the VMs. The mapper maps the processes in the server VM to the IDS VM as *shadow processes*. The IDS VM allocates local identifiers to the shadow processes and the mapper manages the mapping between the process identifiers in the IDS VM and the server VM.

The inter-VM process mapper mediates between the monitored processes in the server VM and the shadow processes in the IDS VM, as illustrated in Figure 5. For example, if an IDS sends a trace request to a shadow process via a legacy interface such as the `ptrace` system call or the proc file system, the IDS VM forwards the request to the mapper using the VMM interface. The mapper translates the process identifier into the corresponding one in the server VM and sends the translated request to the VMM interface of the server VM. The server VM handles the request and returns the response to the IDS VM. If a traced process in the server VM issues a system call, the process stops the execution and notifies an IDS process in the IDS VM. This notification is forwarded to the VMM interface in the server VM, to the inter-VM process mapper, and to the VMM interface in the IDS VM. The IDS VM looks up the IDS process tracing the corresponding shadow process and wakes it up so that the IDS process can obtain information on the system call issued by the target process.

Inter-VM process mapping allows the IDS VM to only monitor the processes in the server VM. Shadow processes are not created in the server VM. Access requests to process identifiers that are not registered in the inter-VM process mapper are denied. The IDS VM is allowed to only read the registers and memory of the shadow processes, so the attacker of a compromised IDS VM cannot rewrite the

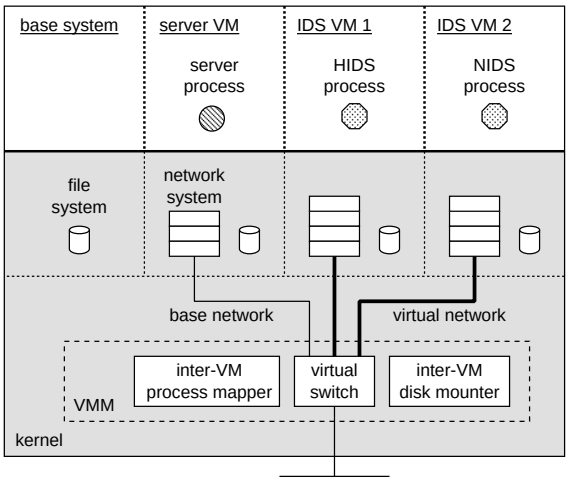

Figure 6: Host implementation of HyperSpector.

memory of the processes in the server VM. In addition, for process tracing, the IDS VM is allowed to control the processes in the server VM through the corresponding shadow processes. An IDS can stop and continue the execution of a traced process. To avoid an illegally long stop of a target process, the mapper forces execution to continue if a stopped process is not continued by the IDS within a certain time. This timeout mechanism mitigates the impact of denial of service attacks by compromised IDSes.

4. IMPLEMENTATION

We have implemented HyperSpector in our Persona operating system, which is based on FreeBSD 4.9. As illustrated in Figure 6, each host in HyperSpector consists of a base system, IDS VMs, a server VM, and a VMM. In our implementation, we assume (1) that the kernel is not compromised and (2) that the base system, which is not affected by the VMs and is not connected to any network, is not compromised. In this section, we first describe the implementation of the IDS and server VMs and then describe the implementation of the inter-VM monitoring mechanisms in the kernel-level VMM.

4.1 IDS VM

The IDS VMs were implemented by extending the *portspace* [15]. The portspace is not a heavy-weight VM that emulates hardware; rather, it provides only an independent namespace for a file system, a network system, and processes. Such virtualization makes the IDS VMs secure enough because we assume that the kernel is developed carefully so that it cannot be compromised. Here, we give an overview of the extended portspace. Further details of the implementation of the original portspace are described in [15].

4.1.1 Network Virtualization

A virtual network is implemented using IPsec tunneling. To allow only a specific IDS VM to use certain IPsec tunnels, an IDS VM provides an independent network system that includes tunnel interfaces, IPsec databases, a routing table, and protocol control blocks. The tunnels can be accessed only through the tunnel interfaces bound to a specific IDS VM. The databases store the tunnel information used by the IDS VM. The routing table contains network routes used only in the virtual network to which the IDS VM is connected. The protocol control blocks bind sockets to arbitrary ports to run new network services in the IDS VM without interfering with the other IDS VMs, the server VM, and the base system even if they use the same IP address.

Among these network elements of an IDS VM, the tunnel interfaces and the IPsec databases can be configured only in the base system so that a fixed configuration of the virtual network is enforced even if the IDS VM or the server VM is attacked. To specify the target IDS VM from the base system, the administrator uses the `setportspace` system call. This call allows the process issued and its descendants to configure the tunnel interfaces and the IPsec databases of the specified IDS VM. For convenience, we developed a utility that uses this system call and configures a virtual network easily.

4.1.2 File System Virtualization

Subdirectory `/.filespace/<id>` (`<id>` identifies an IDS VM) is dedicated to an IDS VM to provide an isolated file system. This subdirectory is translated to the root directory of the IDS VM by using the chroot mechanism. Persona allows an IDS VM to use the file system of the base system by using the union file system so that an IDS VM can use standard binaries and libraries without installation. The union file system enables a subdirectory to be mounted above the existing file system in such a way that both directory trees remain visible. Persona mounts `/.filespace/<id>` of the base system on `/` of the corresponding IDS VM by using the union mount mechanism. Since mounting a subdirectory on `/` is not allowed in the original FreeBSD implementation, we modified the mechanism so that such mounting is allowed only for the IDS VM. We also modified the mechanism so that the mount operation described above is applied only to the target IDS VM, not to the whole system.

This virtualization using the union file system achieves both security and convenience. Since processes in the IDS VM always write files to `/.filespace/<id>`, the IDS VM cannot affect the base system. The processes can read a file from the file system of the base system as long as the file is not overwritten in the IDS VM. This does not degrade the security level of the IDS VM due to our assumption that the file system of the base system is not compromised.

4.1.3 Process Virtualization

The processes in the IDS VM are mainly regular FreeBSD processes. Each has an independent address space and a unique identifier in the host. They use a virtualized file system and a virtualized network, and they can see only the processes inside the same IDS VM. This is achieved by limiting the interactions between processes in the host. A process can send messages to and receive messages from only processes running in the same IDS VM, using inter-process communication, shared memory, and signals.

The administrator enters a particular IDS VM from the base system by using Persona's `swportspace` system call. This call moves the process issued to the specified IDS VM once the process is authenticated. This call is not exploitable by attackers because it can only be issued from the base system, which we assume cannot be compromised from the VMs in the same host or from outside the host.

4.2 Server VM

The server VM is also implemented based on the portspace. While the virtualization of the file system and processes is the same as for the IDS VM, the server VM neither virtualizes the network system nor uses a virtual network. Instead, it uses the base network so that it can provide services to the public Internet. All the packets not received by any virtual networks are delivered to the server VM by the virtual switch.

4.3 VMM

The VMM contains the virtual switch, the inter-VM disk mounter, and the inter-VM process mapper. The virtual switch sends the packets from the VMs to the outside network directly. It also delivers packets from the outside network to the appropriate VM based on the security parameter index in the IPsec header. The value of the index is unique to each virtual network. If a packet has no IPsec header, it is delivered to the server VM. Since the virtual switch does not rely on the IP address for routing, all the VMs in the same host can use the same IP address. This enables the administrator to create multiple HyperSpector environments without having to assign a new IP address to each VM in the same host. To achieve software port mirroring in the virtual switch, the Persona kernel maps the network interface of the server VM to the virtual network interface of the appropriate IDS VM. If an NIDS in the IDS VM opens a BPF device to that interface, Persona copies the packets sent from and to the server VM to that interface. In other words, as long as a BPF device is not opened, no packets are copied.

In our implementation, the inter-VM disk mounter in the VMM, the shadow file system of the IDS VM, and the monitored file system of the server VM reside in the same kernel. Therefore, inter-VM disk mounting is implemented efficiently. When an HIDS process in the IDS VM reads data from a file in the shadow file system, the data is read from the disk of the server VM into buffer cache in the kernel. Since the buffer cache is shared between VMs, the IDS VM can use it without any copying. Overhead is generated only by indirect accesses to the monitored file system via the shadow file system, which is automatically mounted on `/.serverfs` in the IDS VM.

Like inter-VM disk mounting, inter-VM process mapping is also implemented efficiently because the inter-VM process mapper in the VMM, the shadow processes in the IDS VM, and the monitored processes in the server VM reside in the same kernel. Therefore, the process information is shared between VMs, and an HIDS process can get the contents of the registers and memory of a monitored process via the corresponding shadow process without any copying. In addition, all processes except the shadow processes have unique identifiers in the host, and each shadow process uses the same process identifier as the corresponding process in the server VM. It is therefore not necessary for the inter-VM process mapper to translate process identifiers between VMs.

5. EXPERIMENTS

We performed experiments to evaluate the secureness of HyperSpector and to measure the overhead imposed. We used two HyperSpector environments, as illustrated in Figure 7. For host A and the external host, we used PCs, each with a single 3.0 GHz Pentium 4 processor. For host B, we used a PC with a single 2.8 GHz Pentium 4 processor. Each of these three PCs had 1 GB of memory and an Intel Pro/100+ NIC. The PCs were connected via a 100Base-T Ethernet switch.

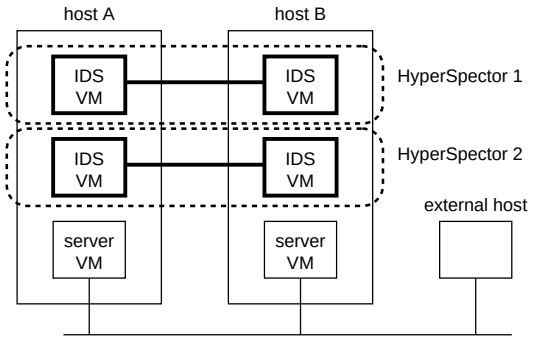

Figure 7: System configuration for experiments.

Table 1: Results of port scanning.

source	target		
	external host	server VM	IDS VM 1
external host	–	http	unreached
server VM	–	http	unreached
IDS VM 1	unreached	unreached	ftp
IDS VM 2	unreached	unreached	unreached

5.1 Secureness

To evaluate the secureness of HyperSpector, we examined the impact of active and passive attacks against IDSes.

5.1.1 Network Security

To evaluate the resistance to attacks made through the network, we scanned the network ports using the nmap port scanning utility [10]. The inetd daemon with ftp service enabled was executed in every IDS VM, while the httpd server was being run in every server VM. First, active attacks were initiated directly from outside by performing port scanning from the external host on the two IDS VMs in HyperSpector 1. Second, active attacks via a compromised server were initiated by performing port scanning from the server VM of host A on the two IDS VMs in HyperSpector 1. Third, after an IDS VM of host A in HyperSpector 1 was compromised by a passive attack, active attacks were initiated by performing port scanning from that IDS VM on the external host, the two server VMs, and the other IDS VM in HyperSpector 1. Finally, active attacks between IDS VMs in the two HyperSpector environments were initiated by performing port scanning from the IDS VM of host A in HyperSpector 2 on the two IDS VMs in HyperSpector 1.

Table 1 shows the results of the port scanning. IDS VM 1 means the IDS VMs in HyperSpector 1. The results of port scanning from the external host and the server VM show that the IDS VMs were not compromised by active attacks because the attacker could not reach the ftp service provided by IDS VM 1. The results of port scanning from IDS VMs 1 and 2 show that the attacker of the compromised IDS VM could not attack other than the IDS VMs in the same HyperSpector environment because IDS VM 2 could

not reach any services provided by the external host, the server VM, or IDS VM 1.

In addition, to evaluate the secureness of software port mirroring, we attempted to capture network packets using the tcpdump utility in the server VM. We found that tcpdump in the server VM could not capture any packets from the IDS VMs. This means that the server VM cannot sniff communication in HyperSpector environments.

5.1.2 File System Security

To evaluate the resistance to attacks against file systems, we checked the independence of file systems when we destroyed the file system of the server VM and two IDS VMs in host A. For the server VM, we deleted all files and directories in the server VM and then compared the file system with those of the IDS VMs and the base system before and after the deletion. For the IDS VMs, we performed the deletion and compared the file system with those of the server VM, the other IDS VM, and the base system. The number of deleted files and directories was 204,706. In both experiments, no files or directories were deleted in other than the VM where the deletion was performed. To evaluate the secureness of inter-VM disk mounting, we attempted to delete files and directories of the shadow file system in an IDS VM, but were unable to delete any files or directories. These results show that the server VM and the IDS VMs cannot change file systems in other VMs.

5.1.3 Process Security

To evaluate the resistance to attacks against processes, we examined process visibility using the ps command. We executed inetd in the two IDS VMs and httpd in the server VM in host A. In the server VM, only the httpd processes were visible. This means that the attacker of a compromised server VM cannot access processes in the IDS VMs. In each IDS VM, not only the inetd process but also the httpd process were visible. The latter are designed to be visible to enable the IDS VM to monitor the server VM. To evaluate the secureness of inter-VM process mapping, we first sent a signal to the httpd of a shadow process from the IDS VM by using the `kill` system call. Next, we attempted to modify the httpd process by using the `ptrace` system call. Both attempts were denied, meaning that the server and IDS VMs cannot attack processes in other VMs.

5.2 Overhead

HyperSpector incurs overhead due to monitoring of the server VM by the IDS VMs. We measured this overhead for three legacy IDSes.

5.2.1 Snort

Snort [19] is a typical NIDS for detecting network-level attacks. Snort sniffs network packets and compares them with attack patterns to detect attacks.

We measured the performance of Snort running in the IDS VM of host A. We used the drop rate of network packets as the performance indicator. We used Snort version 2.0.1 and 1,779 Snort rules, which are included in the Snort package. To impose various loads on Snort, we sent 1-byte UDP packets from the external host to host A at various rates. For comparison, we measured the performance of Snort running in the base system of host A. As shown in Figure 8, when the rate was about 150,000 packets per second, the drop rate of Snort running in the IDS VM was about 3%. As the network load is higher, the drop rate in the IDS VM was higher than in the base system by 0.5 to 0.7%.

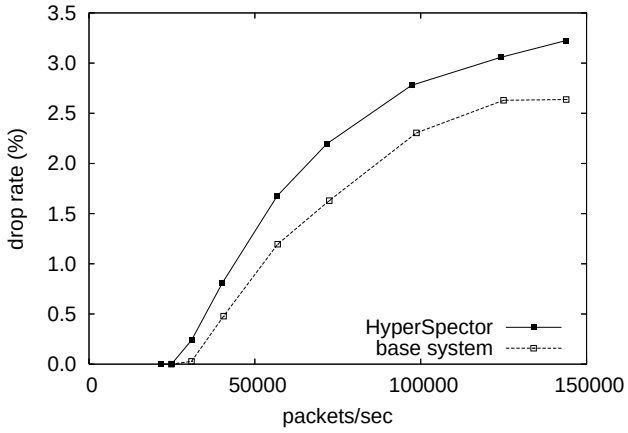

Figure 8: Packet drop rate with Snort.

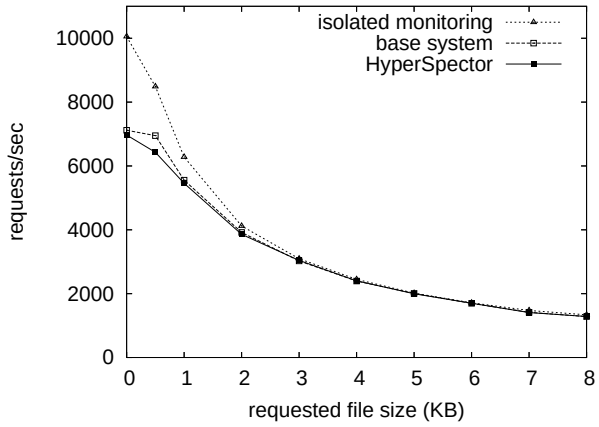

Figure 9: Performance of thttpd with Snort.

Next, we measured the performance of the thttpd 2.23 beta1 web server [17] when it was monitored by Snort in various configurations. We located thttpd and Snort (1) in the base system of host A, (2) in the server and IDS VMs of host A, respectively, using HyperSpector, and (3) in host A and host B, respectively, based on isolated monitoring. To measure the performance of thttpd with Snort, we used the ApacheBench benchmark [1] in the external host. The results are shown in Figure 9. Comparing HyperSpector with the base system, the overhead was 7.5% at maximum and decreased as the requested file size increased. Comparing HyperSpector with isolated monitoring, thttpd running in HyperSpector slowed down by 30% when the requested file size was zero. This is because thttpd in isolated monitoring does not suffer any overhead due to Snort. However, when the requested file size exceeded 2 KB, the overhead was less than 7%. This means that the tradeoff between HyperSpector and isolated monitoring is additional hardware and performance.

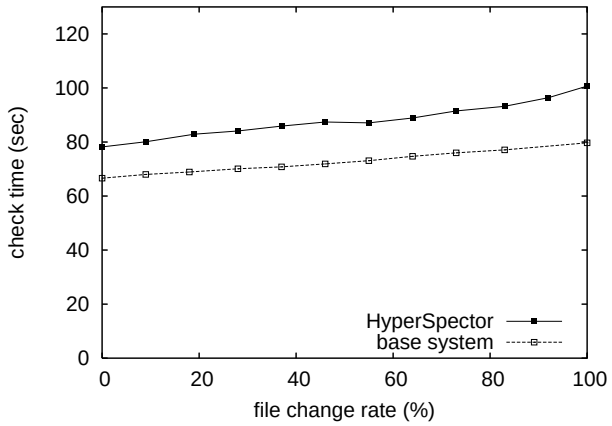

Figure 10: Time for integrity checking in Tripwire.

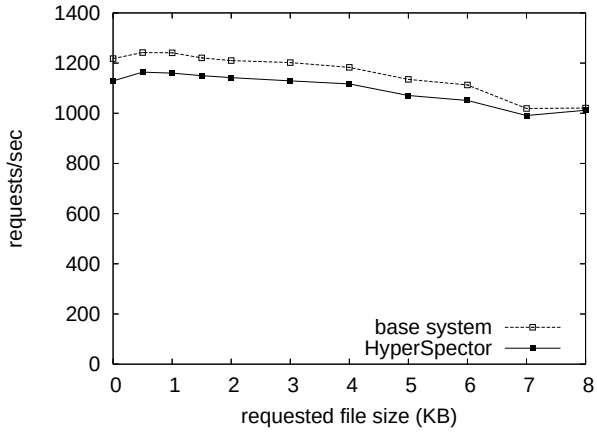

Figure 11: Performance of thttpd traced by truss.

5.2.2 *Tripwire*

Tripwire [14] is a typical HIDS for checking the integrity of file systems. Tripwire stores the state of the file system in a database before attacks and periodically compares it with the current state.

First, we measured the overhead due to initializing the Tripwire database for the file system of the server VM in host A. The measurement was done for the shadow file system, on which the file system of the server VM is virtually mounted. For comparison, we also measured the time for database initialization using the file system of the base system. We used Tripwire version 2.3.1 and 133 rules. The number of files and directories examined based on the rules was 54,885. The initialization time was 48 seconds for the base system and 53 seconds for HyperSpector. Comparing these two, the overhead of HyperSpector was 10.4%.

Next, we measured the time for integrity checking of file systems in the base system and HyperSpector of host A. Figure 10 shows that the overhead due to HyperSpector was 17 to 26%. It increased with the file change rate. This overhead is considered to be for the union file system used in the server VM for ease of use.

5.2.3 *Truss*

Truss is a standard command that traces system calls issued by processes. Strictly speaking, truss is not an IDS because it does not detect intrusions, but we used it to estimate the overhead of tracing system calls between VMs. In this experiment, the thttpd web server was run in the server VM of host A, and truss was executed in an IDS VM of host A. Truss traced the system calls issued by the thttpd. To measure the performance of thttpd as traced by truss, we used the ApacheBench benchmark in the external host. For comparison, we also measured the performance when both thttpd and truss were executed in the base system. As shown in Figure 11, the overhead for tracing the system calls was 0.8 to 7.3%.

6. RELATED WORK

Several architectures that locate servers in a VM and IDSes in the base system have been proposed for isolating servers from IDSes in a host [8, 11]. ReVirt [8] enables an IDS in the base system to record the execution of the VM at the hardware level. Livewire [11] enables IDSes in the base system to monitor the state of the operating system in the VM in addition to the state of the VM. In these architectures, the VM protects IDSes running in the base system from local active attacks via compromised servers. However, these architectures do not assume the DIDS, where multiple IDSes communicate with each other. IDSes running in the base system can be compromised by remote active attacks. Moreover, if an IDS is compromised by a passive attack, the attacker can attack IDSes and servers in other hosts in addition to compromising the VM.

For secure distributed computing, virtual machines and virtual networks should be combined. Figueiredo et al. proposed an architecture that uses both to isolate each execution environment in Grid computing [9]. It enables untrusted programs to be executed securely and provides independent virtual machines and a virtual network for Grid users, but does not provide any monitoring mechanisms. The personal network we previously reported uses virtual machines and virtual networks to enable the user to securely use multiple networks in the user's host [15]. Unlike HyperSpector, the personal network does not provide monitoring mechanisms, and it allows the user to extend the personal network freely.

Isolated monitoring separates servers from IDSes by using additional hardware. Isolated monitoring for NIDSes is achieved by using network taps, instead of a switch with the port mirroring feature. A network tap is hardware inserted in a network cable between a server and a switch. It splits and copies signals in the cable and sends them to NIDS hosts. The pros and cons of using network taps are almost the same as those of using a switch with the port mirroring feature. As described in Section 2.2, Backdoors [2] enables HIDSes to monitor a server host from another host. The key feature of Backdoors is that the operating system in a server host is not responsible for transferring monitored data to an HIDS host. Monitored data is transferred by a programmable NIC in the server host even if the operating system is compromised.

As another monitoring architecture for NIDSes, the administrator can locate the NIDS host at the edge of the distributed system. The NIDS host captures all packets sent from and to the distributed system. However, since the

NIDS host is exposed to the Internet in this architecture, an attacker can easily compromise it.

Many commercial IDSes, such as Counterpane [4], divide their functions between monitoring and analyzing. Only monitoring is done in the target distributed system; the log data are transferred to a security operation center. The center analyzes the data and alerts the distributed system if an intrusion is detected. Although these IDSes outsource the difficult analysis of intrusion detection, they do not make the monitoring process itself secure.

To implement the IDS and server VMs, we extended the portspace; however, VMs that have already been developed are available to which we can add the inter-VM monitoring mechanisms and enforce the use of a virtual network. FreeBSD jail [13] extends the chroot mechanism so that it can virtualize a network space and a process space to some extent. Cloneable network stacks [22] provide independent network stacks from the network interface layer to the application layer and provide independent file systems based on chroot. Zap [16] introduces a pod abstraction, which provides a virtualized view of the operating system to a group of processes. These VMs are light-weight like our IDS and server VMs. On the other hand, VMware [21] and UML [7] are heavy- and middle-weight VMs that can run different operating systems. Although these VMs can prevent attacks to their operating system kernels, their overheads for program execution and inter-VM monitoring are larger.

7. CONCLUSION

We have proposed a virtual distributed monitoring environment called HyperSpector. HyperSpector isolates an IDS from the servers it monitors without using any additional hardware, even for legacy HIDSes. To isolate IDSes and servers inside a host, the IDSes are located in an IDS VM and servers are located in a server VM. To isolate the networks used by the IDSes and servers, the IDS VMs are connected to a virtual network. The IDS VM can monitor the server VM using inter-VM monitoring mechanisms: software port mirroring, inter-VM disk mounting, and inter-VM process mapping. Using such virtualization technologies, HyperSpector protects a DIDS from active attacks and confines the impact of passive attacks to within the affected HyperSpector environment.

One of our future directions is to support active monitoring such as integrity checking by port scanning. Active monitoring requires taking actions against the server VM, e.g. by sending it probe packets, but it is difficult to distinguish these actions from attacks. We plan to develop a filter that allows only strictly restricted interactions between the IDS VM and the server VM.

Also, it is important to protect the HyperSpector environment from denial of service attacks. If an attacker sends a flood of network packets to a server VM, an NIDS in the IDS VM of the same host can become overloaded and almost disabled. As a result, the attacker can compromise the server VM without the intrusion being detected. To prevent such attacks, sufficient resources must be allocated to the IDS VM to enable it to run the IDSes properly under heavy loads.

Another direction is to automatically detect when a HyperSpector environment has been compromised by a passive attack. Although HyperSpector can confine the impact of passive attacks to within one HyperSpector environment, a compromised environment cannot guarantee the correctness of intrusion detection. We should be able to detect the misbehavior of the environment by monitoring the resource usage.

8. REFERENCES

[1] Apache HTTP Server Project. Apache HTTP server benchmarking tool. `http://www.apache.org/`.

[2] A. Bohra, I. Neamtiu, P. Gallard, F. Sultan, and L. Iftode. Remote repair of operating system state using Backdoors. In *Proceedings of the 1st IEEE International Conference on Autonomic Computing*, pages 256–263, 2004.

[3] CERT. Multiple vulnerabilities in snort preprocessors. CERT Advisory CA-2003-13.

[4] Counterpane Internet Security, Inc. Counterpane. `http://www.counterpane.com/`.

[5] CVE. CAN-1999-1462. `http://www.cve.mitre.org/`.

[6] CVE. CAN-2004-0536. `http://www.cve.mitre.org/`.

[7] J. Dike. A user-mode port of the linux kernel. In *Proceedings of the 4th Annual Linux Showcase & Conference*, 2000.

[8] G. Dunlap, S. King, S. Cinar, M. Basrai, and P. Chen. ReVirt: Enabling intrusion analysis through virtual-machine logging and replay. In *Proceedings of the 5th Symposium on Operating Systems Design and Implementation*, pages 211–224, 2002.

[9] R. Figueiredo, P. Dinda, and J. Fortes. A case for Grid computing on virtual machines. In *Proceedings of the 23rd IEEE Conference on Distributed Computing Systems*, pages 550–559, 2003.

[10] Fyodor. The network mapper. `http://www.insecure.org/nmap/`.

[11] T. Garfinkel and M. Rosenblum. A virtual machine introspection based architecture for intrusion detection. In *Proceedings of the Network and Distributed Systems Security Symposium*, pages 191–206, 2003.

[12] P. Horn. Autonomic computing: IBM perspective on the state of information technology. `http://www.research.ibm.com/autonomic/`, 2001.

[13] P. Kamp and R. Watson. Jails: Confining the omnipotent root. In *Proceedings of the 2nd International SANE Conference*, 2000.

[14] G. Kim and E. Spafford. The design and implementation of Tripwire: A file system integrity checker. In *Proceedings of the 2nd ACM Conference on Computer and Communications Security*, pages 18–29, 1994.

[15] K. Kourai, T. Hirotsu, K. Sato, O. Akashi, K. Fukuda, T. Sugawara, and S. Chiba. Secure and manageable virtual private networks for end-users. In *Proceedings of the 28th Annual IEEE Conference on Local Computer Networks*, pages 385–394, 2003.

[16] S. Osman, D. Subhraveti, G. Su, and J. Nieh. The design and implementation of Zap: A system for migrating computing environments. In *Proceedings of the 5th Symposium on Operating Systems Design and Implementation*, pages 361–376, 2002.

[17] J. Poskanzer. Tiny/turbo/throttling HTTP server. `http://www.acme.com/software/thttpd/`.

[18] Quest Software. Big Brother systems and network monitor. `http://www.quest.com/bigbrother/`.

[19] M. Roesch. Snort – lightweight intrusion detection for networks. In *Proceedings of the 13th USENIX System Administration Conference*, pages 229–238, 1999.

[20] S. Snapp, J. Brentano, G. Dias, T. Goan, T. Grance, L. Heberlein, C. Ho, K. Levitt, B. Mukherjee, D. Mansur, K. Pon, and S. Smaha. A system for distributed intrusion detection. In *Proceedings of the COMPCON*, pages 170–176, 1991.

[21] VMware, Inc. VMware. `http://www.vmware.org/`.

[22] M. Zec. Implementing a clonable network stack in the FreeBSD kernel. In *Proceedings of the USENIX 2003 Annual Technical Conference*, pages 137–150, 2003.