Hi-sap: Secure and Scalable Web Server System for Shared Hosting Services

Daisuke Hara, Ryohei Fukuda, Kazuki Hyoudou,
Ryota Ozaki, and Yasuichi Nakayama

Department of Computer Science,
The University of Electro-Communications
Chofu, Tokyo 182–8585 Japan
hara-d@igo.cs.uec.ac.jp

Abstract. We propose *Hi-sap*, a Web server system that solves internal security problems in a server used for shared hosting services and that achieves high site-number scalability with little performance degradation. Customers are often exposed to internal attacks, i.e., malicious customers illegally access other customers' files. Existing approaches solve a portion of this problem, but they are not enough from the view point of performance, site-number scalability, or generality. The proposed system protects customers' files by isolating them in separate security domains, "partitions" that are unit of protection, using a secure OS facility. A default partition is a Web site, and each partition has a Web server instance that runs under the privilege of an individual user and serves files in the partition. Since the Web servers reuse server processes and can run without the burden of a security mechanism themselves, there is little performance degradation. In addition, since Hi-sap dynamically controls the number of Web servers, the number of partitions in a server is scalable. We implemented Hi-sap on a Linux OS and evaluated its effectiveness. Experimental results show that Hi-sap has up to 14.3 times the performance of suEXEC and achieves high scalability of 1000 sites per server.

Keywords: Security in a Server, Shared Hosting Service, Web Server Architecture, Site-number Scalability.

1 Introduction

More people are creating their own contents and publishing them on the Web as the Internet grows in popularity. Although there are various types of services for creating Web contents, many powerful Web publishers tend to use shared hosting services. In the services, service providers typically lease server resources for a monthly/yearly fee for use in building Web sites. Customers login to an assigned server with a given account and install favorite weblogs, wikis [1], and content management systems (CMSs) [2], etc. A customer can thereby publish

I. Tomkos et al. (Eds.): BROADNETS 2010, LNICST 66, pp. 119–137, 2012.
© Institute for Computer Sciences, Social Informatics and Telecommunications Engineering 2012

its contents more flexibly and powerfully. The shared hosting service's requirements for a Web server are security in the server, site-number scalability[1], and performance.

Security is one of the most important concerns on the Web. Vulnerabilities in Web browsers and Web servers are daily found, and both site administrators and site audiences face security risks. Common attacks on the Web, such as cross site scripting and cross site request forgeries, are conducted by external attackers. In addition, the customers are exposed to internal attacks by malicious customers in a shared server. Malicious customers who share a server can illegally access other customers' files on traditional Web servers and OSes. In the server configuration, the customers have to set access permissions[2] on their files so that Web servers can access them. This means that the files can be accessed illegally by malicious customers using command-line tools or through a Web server.

Scalability is also one of the most important concerns on the Web. As the number of Web sites grows, many server machines are required to house them. There are some hardware approaches to improve performance per unit area. For example, blade servers are optimized to minimize physical space and can reduce the server footprints at the data center. However, much more sites must be housed in a machine at shared hosting services. To achieve it, software approaches are also required. That means an innovative server software is desired.

Additionally, we take into consideration *generality*. Generality means a kernel modification is unnecessary and any programming language are supported. If a kernel is modified, it is difficult to keep OS version up to date because of the porting cost. If customers cannot use various programming language, it is inconvenient and does not attract many customers.

It is thought that there was no approach that took into account these requirements. Although existing approaches solve a portion of the security problem, they are not enough from the view point of performance, site-number scalability, or generality. An approach that uses suEXEC [3,4,5] and POSIX ACL [6] (suEXEC & POSIX ACL) has poor performance because they cannot be applied to server-embedded interpreters [7,8,9,10] that process requests for dynamic contents at high speed. Harache [11] performs poorly for server-embedded interpreters. In addition, virtual machines (VMs) or containers which have advanced due to increase hardware performance and the number of CPU cores are problematic for shared hosting services because they have low scalability or low generality; i.e., they typically require modifying the kernel.

To satisfy the requirements of shared hosting service, we propose a secure and scalable Web server system called "Hi-sap" [12] that solves the problems. Customer files are isolated in separate security domains, "partitions" that are unit of protection, using a secure OS [13] facility. A default partition is a Web

[1] Scalability of the number of sites in a server.

[2] To publish static files, for example HTML and image files, read permission must be granted to "other", which is defined by the UNIX permission model "owner/group/other". To publish CGI scripts, execution permission must also be granted.

site, and each partition has a Web server instance that runs under the privilege of an individual user and serves files in the partition. Since the Web servers reuse server processes and can run without the burden of a security mechanism themselves, there is little performance degradation. In addition, Hi-sap implements a Web-server-level scheduler called "content access scheduler", which dynamically controls the number of Web servers. The scheduler reduces memory consumption by partitions for which the Web server is not accessed by clients, so the number of partitions in each server is scalable.

The target of our system is shared Web hosting services in which a server houses from several hundred to one thousand sites. In contrast, VMs and containers target a server machine consolidation or virtual private server in which a server houses from several to dozen OSes or server software programs.

We implemented Hi-sap on a Linux OS and evaluated its effectiveness. Experimental results show that Hi-sap has up to 14.3 times the performance of suEXEC and achieves high scalability of 1000 sites per server.

The remainder of this paper is structured as follows. In section 2, we describe the background. In section 3, we describe the key aspects of our design. In section 4, we describe the implementation of Hi-sap on a Linux OS. In section 5, we describe our evaluation of the system. In section 6, we discuss benefit and limitation of the system. Finally, in section 7, we summarize this work and discuss future work.

2 Background

In this section, we describe shared hosting services, security threats in a shared server, and existing approaches.

2.1 Shared Hosting Services

Providers of shared hosting services lease server resources, such as computing power, network bandwidth, and data storage, to customers for a monthly/yearly fee. The customers login to an assigned server with a given account and build their Web sites on the server, which is usually shared with many other customers. The customers share the same OS image. The number of customers housed on a commodity server is generally from several hundred to one thousand.

The biggest concern of the service providers is the server footprints at the data center. To reduce operating expense, it is important to reduce the footprints. Recent virtualization technologies described later in section 2.3 and blade servers aim at doing this. The service providers therefore want to maximize the number of customers housed on a server. Their servers thus need to process requests for contents at high speed and use computation resources, for example CPU, memory, and disk, effectively.

An effective common approach to processing requests for contents at high speed is to use server-embedded interpreters for dynamic contents, which consume more computing resources than static contents. Dynamic contents are essential for a rich user experience on the Web. A common gateway interface (CGI)

has traditionally been used to generate dynamic contents. However, it is difficult for CGI to process dynamic contents at high speed because it requires a process termination, i.e., invoking fork() and execve() system calls, for each request. Therefore, server-embedded interpreters, such as mod_ruby [8], mod_perl [9], and mod_python [10], have been used as an alternative to CGI. The interpreters are contained in a server process and run as a part of the process. Although server-embedded interpreters are commonly used on the Web, it is difficult to use them for shared hosting services because of a security problem described later in section 2.2. Therefore, suEXEC and the PHP: Hypertext Preprocessor (PHP) safe mode [7] described later in section 2.3, 2.3 are still used in the service.

One way to use computation resources effectively is to assign resources to sites in proportion to the volume of access traffic at each site to reduce resource consumption by each site. The traffic distribution on the Web is known to follow Zipf's law [14]; i.e., while a few sites get a large amount of traffic, most sites get little traffic. Resource allocation for sites that are not accessed at all should be avoided.

The service providers additionally want to use general technologies to do maintenance easily. For example, they want to avoid modifications of kernel or server software.

2.2 Security Threats in a Shared Server

The sharing of a server by many customers has caused new security threats. The seed of them is the roughness of traditional OS access control; i.e., the file permissions are managed for only three classes, owner, group, and other.

The customer is given an account and assigned a user ID by the service provider for use in logging in to the assigned server. The "owner" of the customer's files is set to the user ID, and the files can be accessed only by the customer. Thus, permissions must be granted for the files to "group" or "other" so that the Web server, which is assigned a dedicated user ID[3], can access them (Figure 1 (0)). In this situation, the files can be illegally stolen, deleted, or tampered with by malicious customers that share the server by using command-line tools, for example *cp* and *rm* (Figure 1 (1)). They can additionally attack through the Web server (Figure 1 (2)). For example, a malicious CGI script that deletes an other customer's writable files[4] can run because the script runs under the privilege of the dedicated user, which can write the file.

We identified the factors in the security scenario for a server:

- Worth protecting: customer files
- Threat: stealing, deletion, or tampering with files by malicious customers that share a server
- Vulnerability: coarse-grained isolation of files in traditional OSes.

The *one-to-one* approach can be used to solve this security problem. In this approach, customer files are isolated in separate domains. Each domain has a

[3] e.g., *apache*, *www-data*, *www*.

[4] e.g., a log file, wiki's data file.

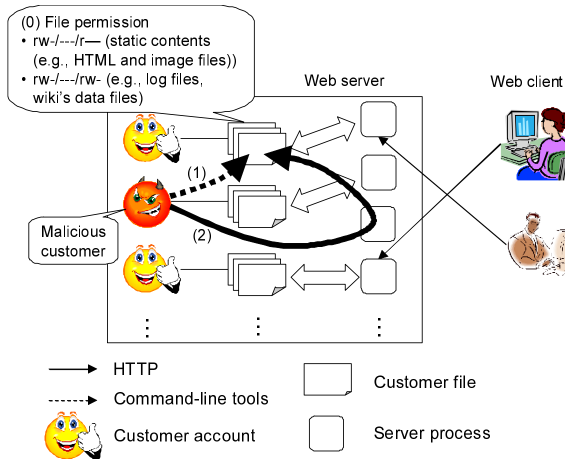

Fig. 1. Internal security treats at a shared hosting service

Web server instance that serves only files in the domain. There is also a reverse proxy server that dispatches requests from Web clients to the instance. However, this approach is unsuitable for shared hosting services because of its poor site-number scalability. This is because all instances run all the time though most sites get little traffic, as mentioned in section 2.1.

2.3 Existing Approaches and Their Limitations

Although there are approaches that solve the security problem, they have limitations.

Container and Virtual Machine. Containers [15,16,17,18,19] are OS-level virtualization methods. Multiple containers with server software programs can run concurrently in an OS (Figure 2 (1)). Each container has a different namespace. Assigning a container to every site creates high security in the server. However, using containers at shared hosting services is difficult because of their low scalability for the number of sites in a server. Although this mechanism can scale up to a few hundred sites, service providers require up to about 1000 sites. In addition, some containers, for example Linux-VServer [17], need to modify the kernel. Kernel modifications are dependent on the kernel version, so keeping them up to date generally requires significant porting [19]. If the porting is not done, the kernel's latest features and devices cannot be used.

In VMs [20,21,22,23,24,25], a hypervisor can run multiple OSes concurrently on the same server machine (Figure 2 (2)). Assigning an OS to every site also creates high security in the server. However, using VMs at shared hosting services is difficult because of the overhead involved. The utilization of computation resources for each site dramatically increases when this mechanism is used. This strongly affects the scalability of the number of sites in a server. For example,

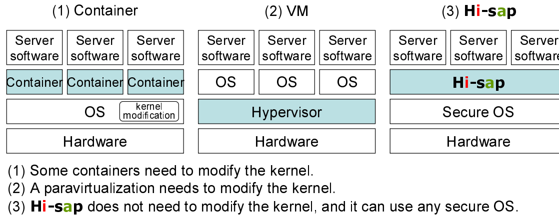

Fig. 2. Software stack of container, VM, Hi-sap

an OS that runs server programs on VMware ESX Server reportedly uses about 200 MB of memory [21]. That means about 200 GB of memory is required to provide 1000 sites. Although some mechanisms of memory sharing between VMs are proposed recently [22,23,24], no reports show that the number of sites in a server reaches up to 1000. Vrable *et al.* described that Xen could not run concurrently more than 116 VMs because of Xen's limitation [22]. Gupta *et al.* [23] described that their evaluations only used up to 6 VMs. In addition, paravirtualization [25] needs to modify the kernel, and it has low generality.

In Hi-sap, server software programs share a single namespace in an OS (Figure 2 (3)). Therefore, our system can control each server software program by using content access scheduler and achieves high scalability of the number of sites. It also achieves high generality because it does not need to modify the kernel, and it can use any secure OS, which is described later in section 3.1.

PHP Safe Mode. PHP [7] has a *safe mode*. This mechanism maintains a high level of security in a server by restricting the operations of PHP scripts.

- File handling is permitted only when the owner of the script is the same as the owner of the file that the script is about to handle.
- File handling is permitted only below specific directories.
- Environment variables that can be changed are restricted.
- Specific functions and classes are disabled.

However, this mechanism depends on the language processor and is not commonly used. There are also many cases when using this mechanism is difficult because of its restrictions.

Hi-sap supports any programming language because it provides a security mechanism outside the language processor.

suEXEC & POSIX ACL. The suEXEC program uses "setuid bit" to run CGI scripts under the privilege of an individual user different from a dedicated user. POSIX ACL provides access control for each user, unlike traditional access control, i.e., owner/group/other.

In this approach, first, read and execution permission for public access files is granted only to a dedicated user by using POSIX ACL. Files can therefore be

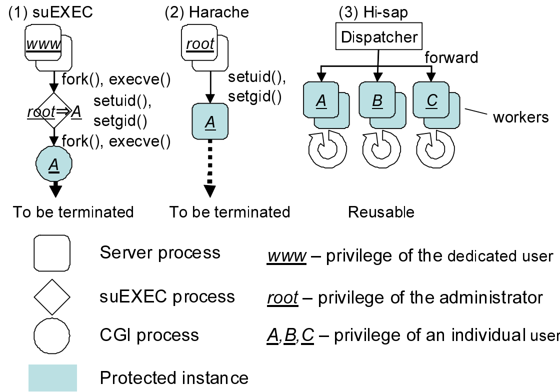

Fig. 3. Process composition of Web server systems

published without granting permission to "other". Second, CGI scripts run under the privilege of the site owner by using suEXEC. This approach can therefore prevent execution of cp and rm commands and prevent malicious customers that share the server from using CGI scripts to steal, delete, or tamper with files.

However, suEXEC cannot achieve the speed of server-embedded interpreters since it needs a process termination after each request (Figure 3 (1)). In addition, because server-embedded scripts that are executed by using server-embedded interpreters run under the privilege of the dedicated user, they cannot ensure the security in a server. suEXEC is therefore applied only to a CGI.

Harache. Our previously proposed Web server system, Harache [11], enables safe and convenient use of server-embedded programs. With Harache, each process of a Web server runs under the privilege of an individual user for every site. Harache therefore requires that permission be granted to only "owner" for any contents that include server-embedded scripts. Although Harache has up to 1.7 times the performance of suEXEC, it cannot achieve the speed of server-embedded interpreters since it needs a process termination after each HTTP session (Figure 3 (2)).

3 Design

We designed our proposed server system, Hi-sap, which can be used with UNIX-like OSes, with two goals in mind.

- High security with little performance degradation
- High site-number scalability in a shared server.

3.1 Security with Little Performance Degradation

Performance Degradation in Existing Systems. In general, a fundamental requirement for protecting files completely in a server is that the files are accessible only by the owner. This requires that processes that access the files have to be an owner of the files. In a shared server, Web servers have to run under the privilege of an individual user. However, changing the server privilege is problematic. Popular types of Web servers, such as Apache [3], consist of several server processes. Since the processes share the listen port (usually port 80) to accept requests from Web clients, the Web server system cannot control which of the requests a process grabs. Therefore, changing the privilege of a process, i.e., invoking setuid() system call, has to be done after a request is grabbed because the Web site (and in turn the customer files) about to be accessed by the request is not known until the request arrives. However, changing the privilege of a process is noninvertible because setuid() requires the administrator privilege. Consequently, the approaches that use an ordinary Web server system have a problem: the processes have to be terminated after finishing a request or an HTTP session (Figure 3 (1)(2)). For example, the CGI processes in the suEXEC & POSIX ACL approach and the server processes in Harache have to be terminated. Unsurprisingly, this degrades performance due to the increase in process terminations and activations. This means that improving security with little performance degradation requires a proactive privilege change ability.

The Hi-sap Approach. In the Hi-sap approach, the privilege of processes is changed in advance to avoid performance degradation (Figure 3 (3)). A "dispatcher" distributes requests to "workers" that run under the privilege of an individual user. The system protects customer files by isolating them in separate security domains, called partitions that are unit of protection, using a secure OS facility.

Partition: A partition in our system is a site or content. A default partition is a site. Each partition has a Web server instance (worker) that serves files in the partition. File permissions are granted for any files in a partition to only owner. A worker can therefore access only a specific dedicated partition (cannot access any other partitions) because workers run under the privilege of an individual user.

In the example shown in Figure 4, there are two sites (site *X* and *Y*). Site Y contains two contents (content *Y1* and *Y2*). A partition is assigned to site X, content Y1, and content Y2. The worker dedicated to processing requests for content Y1 (worker (B)) cannot access files on site X (Figure 4 (a)). Even if the files are on the same site, the worker cannot access them if they are in an other partition (Figure 4 (b)).

Since the Web servers reuse server processes and do not bear any overhead for security, there is little performance degradation.

Combination with Secure OS: If the privilege of the administrator account is appropriated due to a security hole or mis-configuration, there is no effect on the access control of Hi-sap, which uses individual user privileges.

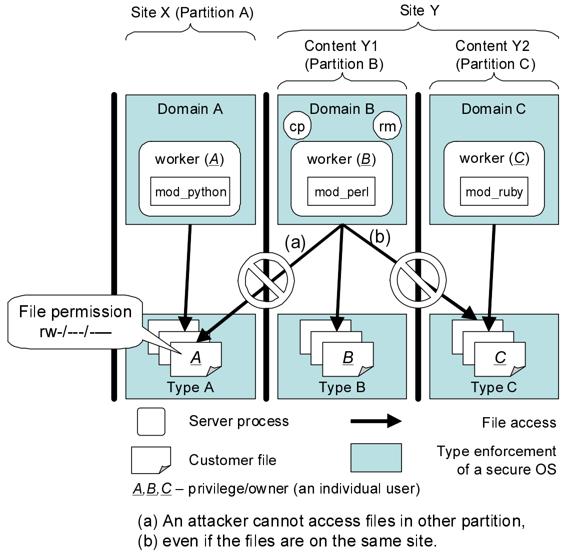

Fig. 4. Partitions

The Hi-sap enables access control in combination with a secure OS [13]. A secure OS enhances security features, e.g., mandatory access control (MAC) [26] and least privilege [27] security. The MAC mechanism enforces access control for all users and processes without exception. In the least privilege security model, a higher-than-needed privilege level is not granted to users and processes. These mechanisms isolate server software programs from the other server software programs in the same OS; i.e., cracking one server software program does not affect the other programs.

To prevent files from being stolen, deleted, or tampered with when the administrator account is appropriated, our system assigns "domain" of a secure OS to every worker and assigns "type" to every partition. In Figure 4, domain A, B and C are assigned to worker (A, B and C), and type A, B and C are assigned to files of partition A, B and C. If worker (B) is cracked and the privilege of the administrator account is appropriated, the attacker can access only partition B (content Y). Therefore, the system ensures the security for each partition by using a secure OS.

The system requires MAC and the least privilege security model; i.e., it can use any secure OS.

3.2 Scaling Number of Customers

Our content access scheduler is a Web-server-level scheduler that enhances the scalability of the number of partitions in a server. It controls the creation and termination of workers.

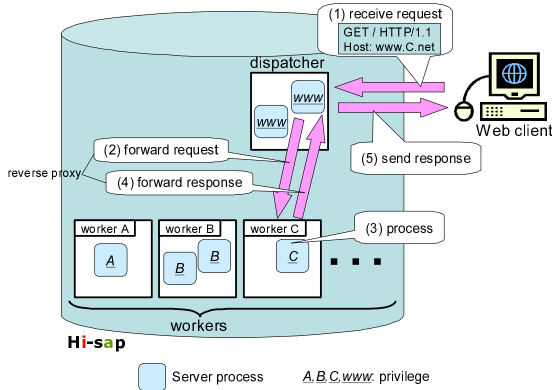

Fig. 5. Overview of Hi-sap architecture

In a Web server, memory utilization strongly affects scalability. Thrashing decreases the performance of Web servers dramatically [11]. Therefore, the system dynamically terminates workers not required to save memory resources. This means the system keeps only necessary workers. This scheduler works well because of Zipf's law described in section 2.1; i.e., while workers for a few sites that get a large amount of traffic are always active, workers for most sites that get little traffic are usually inactive.

The scheduler enables high scalability, in particular, by optimizing the algorithm used to create and terminate workers in accordance with the characteristics of the contents.

3.3 Hi-sap Architecture

An overview of the system architecture is shown in Figure 5. The system consists of a dispatcher and many workers. Each worker runs under the privilege of an individual user and processes requests for a specific dedicated partition. The dispatcher is a reverse proxy server and distributes requests to workers.

Secure OSes have trouble when transferring user privileges. If the policy of a secure OS permits workers that run under the privilege of the administrator account to transfer privileges to ordinary users, problems may occur if workers are appropriated. That means secure OSes are ineffective because a worker that is appropriated can transfer privileges to any ordinary user. Therefore, in our system, workers initially run under the privilege of ordinary users.

4 Implementation

We implemented Hi-sap on a Linux OS with SELinux [13]. The dispatcher was implemented as an Apache module, *mod_hisap*, on an Apache HTTP server (ver. 2.0.55) [3]. One thousand Apache HTTP servers (ver. 2.0.55) were used

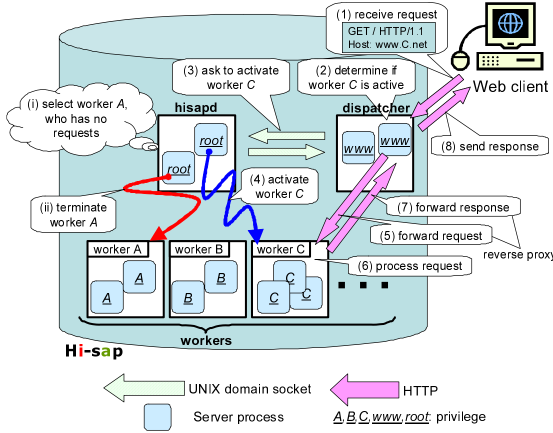

Fig. 6. Overview of Hi-sap request processing

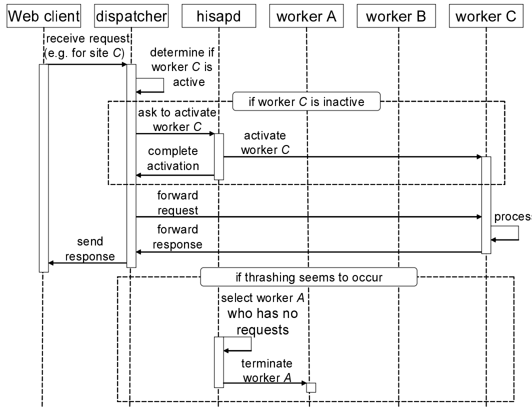

Fig. 7. Sequence of Hi-sap request processing

as workers. Each worker waited for requests at a unique port. The content access scheduler and other management facilities of the workers were implemented as a daemon, *hisapd*. An overview and sequence of request processing for the system is shown in Figure 6, Figure 7.

Our system has a simple and user-level implementation and does not need to modify the kernel, and it can use any secure OS. Therefore, our system can be easily ported to any UNIX-like OSes. In addition, our system can easily scale out because workers can be distributed to many server machines.

The details of the dispatcher and hisapd are as follows.

4.1 Dispatcher

If the dispatcher receives a request, e.g., for partition *C* in Figure 6, from a Web client (Figure 6 (1)), the dispatcher determines whether the dedicated worker for partition C is active (Figure 6 (2)). If the worker (worker C) is inactive, the dispatcher asks hisapd to activate it (Figure 6 (3)). The communication between the dispatcher and hisapd uses a UNIX domain socket. "Worker ID", the identifier of the requested worker, is then recorded in a dedicated log file, "worker request log". After hisapd activates the worker (Figure 6 (4)), the dispatcher forwards the request to the worker (Figure 6 (5)). The worker processes the request (Figure 6 (6)) and forwards the response to the dispatcher (Figure 6 (7)). The dispatcher sends the response to the Web client (Figure 6 (8)).

4.2 hisapd

As described in section 4.1, hisapd dynamically activates workers after receiving requests from the dispatcher.

There is also a procedure for worker termination. When thrashing seems to occur, hisapd terminates workers that have not been requested recently. The conditions under which hisapd judges thrashing seems to occur are as follows.

- A swap-in occurs.
- A swap-out occurs.
- Memory utilization is equal to or greater than 99%[5].

It checks for these conditions every five seconds[5]. When all conditions are met, hisapd starts terminating workers. It selects which workers to terminate on the basis of two conditions.

- The worker is active.
- The worker is not recorded in the most recent 10,000[5] requests in the worker request log.

The pseudo least recently algorithm is used to reduce the time for searching the worker request log. As illustrated in Figure 6, when thrashing seems to occur, hisapd selects worker *A* because it has not been requested recently (Figure 6 (i)) and terminates it (Figure 6 (ii)).

4.3 SELinux Configuration

The SELinux file context (FC) file defines the relationship between a file and the security context of SELinux. Each worker is installed at */vhosts/"Worker ID"/*. The */vhosts/"Worker ID"/bin/apachectl* scripts for starting and stopping workers are assigned the same security context. Other files are assigned a different security context in every partition because they are used while a worker is running.

[5] This value is adjustable.

5 Evaluation

We evaluated Hi-sap using the hardware configuration listed in Table 1.

Table 1. Hardware configuration of experimental environment

Network	
Switching Hub	Dell PowerConnect 2724 1000 BASE-T × 24

Client	
CPU	Intel Pentium III Xeon 500 MHz × 4
Memory	256 MB (swap 512 MB)
OS	Fedora Core 4 (Linux 2.6.14)
NIC	Intel PRO/1000XT (1 Gbps)

Server	
CPU	AMD Opteron 240EE 1.4 GHz × 2
Memory	4 GB (swap 8 GB)
OS	Fedora Core 4 (Linux 2.6.14)
NIC	Broadcom BCM5704C (1 Gbps)

5.1 Basic Performance

We evaluated the basic performance of Hi-sap when processing dynamic contents to determine its effectiveness. An Apache HTTP server ver. 2.0.55 (Apache), an Apache enabling suEXEC (suEXEC), and a one-to-one approach that described in section 2.2 were used for comparison. A one-to-one system enables access control in combination with a SELinux to isolate each worker. Although a one-to-one system is similar to our system, mod_hisap and hisapd were not installed. Therefore, all workers ran from beginning to end. Apache and suEXEC did not enable a SELinux. In our system, Apache, and one-to-one, a PHP script was executed by the server-embedded interpreter. In suEXEC, a PHP script was executed as a CGI. Our system, Apache, suEXEC, and one-to-one used the default configuration files. We used httperf benchmark ver. 0.8 [28] to measure performance.

We sent requests to the PHP script and measured the response throughput. The script calls *phpinfo()*, which displays the system information of the PHP language processor. The traffic generated by the script is 40 KB per request. As shown in Figure 8, the throughput with our system was, on average, 28.0% lower than with Apache and was a maximum of 56.5% lower. This was due to the overhead of the reverse proxy operation. However, the throughput was, on average, 10.2 times that with suEXEC and was a maximum of 14.3 times the throughput. It was, on average, 1.0% lower than with one-to-one and was a maximum of 2.6% lower. This was due to the overhead of mod_hisap and hisapd operation. Since this overhead is very low, this implementation is effective.

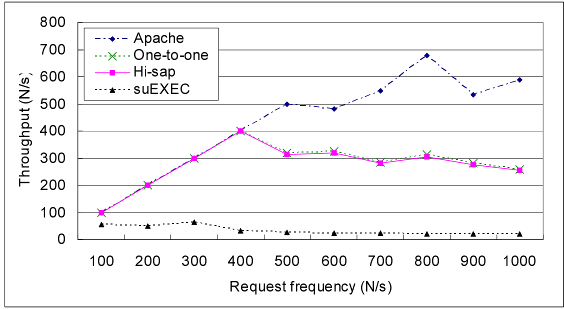

Fig. 8. Basic performance evaluation

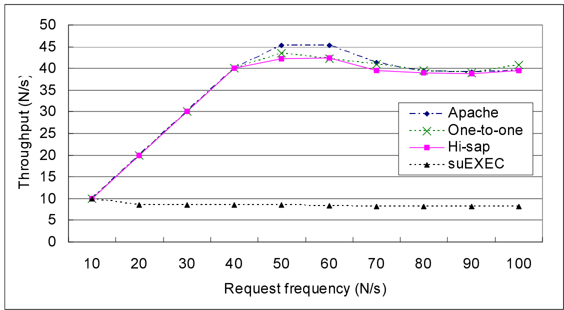

Fig. 9. Basic performance evaluation: Weblog

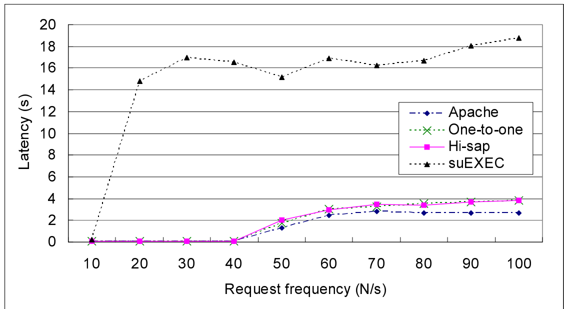

Fig. 10. Basic performance evaluation: latency

In addition, we performed the same experiment for a real application (Weblog). We used tDiary[6] ver. 2.0.2 written in Ruby. As shown in Figure 9, the throughput with our system was, on average, 2.0% lower than with Apache and

[6] http://www.tdiary.org/

was a maximum of 6.9% lower. It was, on average, 1.1% lower than with one-to-one and was a maximum of 3.7% lower. The reason for reducing performance difference between Apache and our system compared to the experiment using the PHP script is that the processing time for communication increased. As shown in Figure 10, the latencies of Apache, one-to-one, and our system were small. In contrast, the latency of suEXEC was very large, so suEXEC is not suitable.

This evaluation demonstrates the system has sufficiently high performance while ensuring security in a server.

5.2 Scalability

We evaluated the site-number scalability of Hi-sap when processing dynamic contents. The one-to-one approach was used for comparison. This experiment was designed to determine the effectiveness of the content access scheduler.

We sent 100 requests to a PHP script in each partition sequentially and measured the response throughput. The script was the same as that described in section 5.1. We used the Apache HTTP server benchmarking tool (ver. 2.0.41-dev). As shown in Figure 11, our system had substantially higher throughput

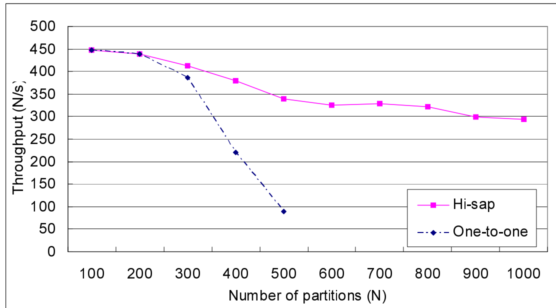

Fig. 11. Scalability evaluation

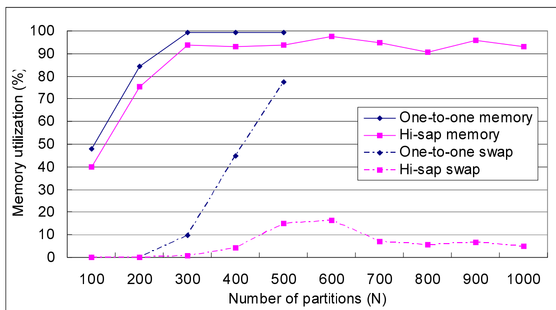

Fig. 12. Scalability evaluation: memory utilization

Table 2. Comparison of approaches (overall evaluation)

	Security in Server	Basic Performance	Site-number Scalability
Apache	poor	excellent	excellent
suEXEC & POSIX ACL	good	poor	excellent
One-to-one	good	excellent	poor
Hi-sap	**excellent**	**excellent**	**excellent**

than one-to-one from beginning to end. The reduction in throughput with our system as the number of partitions increased was lower than with one-to-one. With one-to-one, the OS crashed due to a memory shortage when the number of partitions reached about 600.

The change in memory utilization is plotted in Figure 12. The swap utilization of one-to-one increased dramatically as the number of partitions increased, which is the reason for the OS crashing. In contrast, our system does not use swap space as much because of the content access scheduler. In addition, although our system could avoid swapping by advancing the scheduling algorithms, it gave priority to immediate evaluation of the entire system.

This experiment demonstrates that the system has high site-number scalability.

5.3 Comparison of Approaches

As shown in Table 2, suEXEC & POSIX ACL had poor performance, and scalability of one-to-one was lower than that of others. Our system ranked high for all items and did not have any weak points. It is therefore the most effective.

6 Discussion

In this section, we discuss benefit, limitation, and target of the system.

6.1 Benefit of a User-Level Approach

In our system, server software programs of all sites share a single namespace in an OS. Computation-resource utilization of our system is much less than that of VMs and containers because all sites share computation resource except Web server processes. Therefore, it achieves high site-number scalability.

It also achieves high generality because it does not need to modify the kernel, and it can use any secure OS. As described in section 2.3, kernel modifications require significant porting effort. Our user-level approach enables compatible between different kernel versions for binary compatibility, and also compatible with other UNIX-like OSes for source compatibility.

6.2 Limitations

Network Isolation: VMs and containers provide a network isolation mechanism. A server software program on them is isolated at layer 2 or 3 from the others.

In our system, workers share a single IP address and use a unique port. SELinux can restrict access to each port. A worker therefore cannot sniff packets of the others.

Administrative Cost: Our system has many workers. Installation and maintenance costs increase in proportion to the number of workers that include a Web server instance and contents. To add or remove a worker, configuration of a dispatcher and SELinux is also required.

Because VMs and containers require installation of an OS or a container in addition to setup Web server instance and contents, administrative cost of our system is believed to be lower than that of VMs and containers.

Response Time for Request to Inactive Workers: Our system control the creation and termination of workers by using content access scheduler to achieve high site-number scalability. A response for a request to inactive workers thus takes a longer time than that to active ones because worker activation is required. However, this latency does not matter because running a server software program is very fast.

On the other hand, VMs and containers require to boot an OS (VM) or a container in addition to run a server software program. This takes a lot of time.

6.3 Target of Hi-sap

Our system is applicable to Web hosting services with following conditions.

- A dedicated OS for each Web site is unnecessary.
- The demand of communication confidentiality is not severe.
- The demand of real-time processing is not severe.

The target of our system is therefore shared Web hosting services in which a server houses from several hundred to one thousand Web sites.

If the demands are much severe, a dedicated server or a virtual private server is available.

7 Conclusion

This paper has three contributions. First, we have clarified security problems and requirements of shared hosting services. The requirements are security in the server, performance, and site-number scalability. Second, we have clarified that existing approaches and their limitations. It is thought that there was no approach that took into account these requirements. At last, we have designed

a secure and scalable Web server system for shared hosting services, and implemented it on a Linux OS with SELinux. Our evaluation results demonstrate our system qualitatively and quantitatively satisfies the requirements.

We plan to optimize the content access scheduler algorithm to avoid swapping and to enable more than 1000 sites to be housed. In addition, the concept of Hi-sap can be applied other daemons, for example mail servers and network file systems, which provide the service to many users in a server.

Acknowledgments. This work was supported in part by the Exploratory Software Project of the Information-technology Promotion Agency, Japan.

References

1. WikiWikiWeb, `http://c2.com/cgi/wiki?WikiWikiWeb`
2. Goodwin, S., Vidgen, R.: Content, content, everywhere...time to stop and think? The process of Web content management. IEE Computing & Control Engineering Journal 13(2), 66–70 (2002)
3. Apache HTTP Server, `http://httpd.apache.org/`
4. Neulinger, N.: CGIWrap: User CGI Access, `http://cgiwrap.sourceforge.net/`
5. Marsching, S.: suPHP, `http://www.suphp.org/`
6. Grunbacher, A.: POSIX Access Control Lists on Linux. In: Proc. FREENIX Track: 2003 USENIX Annual Technical Conference, pp. 259–272 (2003)
7. PHP: Hypertext Preprocessor, `http://www.php.net/`
8. mod_ruby, `http://modruby.net/`
9. mod_perl, `http://perl.apache.org/`
10. mod_python, `http://www.modpython.org/`
11. Hara, D., Ozaki, R., Hyoudou, K., Nakayama, Y.: Harache: A WWW Server Running with the Authority of the File Owner. J. IPS Japan 46(12), 3127–3137 (2005) (in Japanese)
12. Hara, D., Nakayama, Y.: Secure and High-performance Web Server System for Shared Hosting Service. In: Proc. the 12th International Conference on Parallel and Distributed Systems (ICPADS 2006), pp. 161–168 (2006)
13. Loscocco, P., Smalley, S.: Integrating Flexible Support for Security Policies into the Linux Operating System. In: Proc. FREENIX Track: 2001 USENIX Annual Technical Conference, pp. 29–40 (2001)
14. Classman, S.: A Caching Relay for the World Wide Web. In: Proc. the 1st International World-Wide Web Conference, pp. 69–76 (1994)
15. Kamp, P., Watson, R.: Jails: Confining the omnipotent root. In: Proc. the 2nd International System Administration and Networking Conference (2000)
16. Dike, J.: A user-mode port of the linux kernel. In: Proc. the USENIX Annual Linux Showcase and Conference (2000)
17. Linux-VServer, `http://linux-vserver.org/`
18. Linux containers, `http://lxc.sourceforge.net/`
19. Suranyi, P., Abe, H., Hirotsu, T., Shinjo, Y., Kato, K.: General Virtual Hosting via Lightweight User-level Virtualization. In: Proc. the 2005 International Symposium on Applications and the Internet (SAINT 2005), pp. 229–236 (2005)
20. Barham, P., Dragovic, B., Fraser, K., Hand, S., Harris, T., Ho, A., Neugebauer, R., Pratt, I., Warfield, A.: Xen and the Art of Virtualization. In: Proc. the 19th ACM Symposium on Operating Systems Principles (SOSP 2003), pp. 164–177 (2003)

21. Waldspurger, C.A.: Memory Resource Management in VMware ESX Server. In: Proc. the 5th Symposium on Operating Systems Design and Implementation (OSDI 2002), pp. 181–194 (2002)
22. Vrable, M., Ma, J., Chen, J., Moore, D., Vandekieft, E., Snoeren, A., Voelker, G., Savage, S.: Scalability, Fidelity and Containment in the Potemkin Virtual Honeyfarm. In: Proc. the 20th ACM Symposium on Operating Systems Principles (SOSP 2005), pp. 148–162 (2005)
23. Gupta, D., Lee, S., Vrable, M., Savage, S., Snoeren, A.C., Vahdat, A., Varghese, G., Voelker, G.M.: Difference engine: Harnessing memory redundancy in virtual machines. In: Proc. the 8th Symposium on Operating Systems Design and Implementation (OSDI 2008), pp. 309–322 (2008)
24. Milos, G., Murray, D.G., Hand, S., Fetterman, M.A.: Satori: Enlightened page sharing. In: Proc. the 2009 USENIX Annual Technical Conference (USENIX 2009), pp. 1–14 (2009)
25. Whitaker, A., Shaw, M., Gribble, S.: Denali: Lightweight Virtual Machines for Distributed and Networked Applications, University of Washington Technical Report, 02-02-01
26. McLean, J.: The algebra of security. In: Proc. 1988 IEEE Symposium on Security and Privacy, pp. 2–7 (1988)
27. Saltzer, J.H., Schroeder, M.D.: The protection of information in computer systems. Proc. the IEEE 63(9), 1278–1308 (1975)
28. Mosberger, D., Jin, T.: httperf—A Tool for Measuring Web Server Performance. In: Proc. the 1st Workshop on Internet Server Performance, pp. 59–67 (1998)