Secure and High-performance Web Server System for Shared Hosting Service

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

Yasuichi Nakayama
Department of Computer Science
The University of Electro-Communications
Chofu, Tokyo 182-8585 Japan
yasu@cs.uec.ac.jp

Abstract

We developed Hi-sap, a web server system that ensures the security in a server and has high performance when processing dynamic content. In existing servers, server embedded programs cannot be used safely in large-scale environments like a shared hosting service. These problems occur because server processes run under the privilege of an identical user. For example, server embedded interpreters are commonly used to improve performance in processing dynamic content, like weblogs and wikis. However, other customers that share the same server can steal, delete, and tamper with data files of weblogs and wikis. To solve these problems, we designed a new web server system, Hi-sap. In the system, web objects that are stored in a server are divided into partitions. Server processes run under the privilege of different users in every partition. We implemented Hi-sap on a Linux OS and tested the effectiveness of the system. Experimental results show that Hi-sap has high performance and scalability.

1. Introduction

More people are creating their own websites as the Internet grows in popularity. Shared hosting services, where many customers share a server, are widely used.

With existing servers, server embedded programs, for example server embedded interpreters [1, 2, 3, 4] or WebDAV [5], cannot be used safely and conveniently in large-scale environments like shared hosting services. These problems occur because server processes that load server embedded programs run under the privilege of a identical user (dedicated user\(^1\)). When server embedded interpreters that improve performance in processing dynamic content like weblogs, wikis [6], and content management systems (CMSs) [7] are used, other customers that share the same server can steal, delete, or tamper with data files of weblogs

\(^1\)Dedicated to run server processes: apache, www-data, www, etc.

server.

In addition, when WebDAV is used, the owner of created files is the dedicated user. Therefore, ordinary users that actually own files typically cannot edit these files directly.

To solve these problems, we designed and implemented a secure and high-performance web server system, Hi-sap [8]. In the system, web objects\(^2\) that are stored in a server are divided into partitions, for example sites and content. Server processes run under the privilege of different users for every partition. Thus, the system can prevent theft, deletion, and tampering when server embedded interpreters are used. Ordinary users can directly edit files that are created by way of WebDAV. Therefore, the system can solve the problems of using server embedded programs.

Also, we propose a web-server-level scheduler named Content Access Scheduler to enhance the scalability of the number of partitions in a server. We implemented Hi-sap on a Linux OS and tested the effectiveness of the system. Experimental results show that Hi-sap has high performance and scalability.

The remainder of this paper is structured as follows. In section 2, we describe the background of this work. In section 3, we describe the key aspects of our design. In section 4, we describe an overview of our implementation of Hi-sap on a Linux OS. In section 5, we present an evaluation of the system. In section 6, we describe the related work and discuss realization approaches of a server system. Finally, in section 7, we conclude.

2. Background

In this section, we describe the security in servers, server embedded interpreters, our previous work, Harache, and shared hosting services.

\(^2\)Sets of public access files, directories, HTTP environment variables, etc.
2.1. Security in Servers

Existing servers run under the privilege of a dedicated user. Thus, it is required to grant read and execution permission to an “other” defined by the UNIX permission model, “owner/group/other”. Even if HTTP authentication is used to protect content against outside attacks, internal customers that share the same server can steal, delete, or tamper with these content (Figure 1).

It is required to use both suEXEC [9, 10, 11] and POSIX ACL [12] (hereinafter, “suEXEC & POSIX ACL”) to solve these problems. First, read and execution permission of public access files are granted to only the dedicated user by using POSIX ACL. Therefore, content can be published without granting any permission to an “other”. Second, CGI scripts run under the privilege of the site owner by using suEXEC. Therefore, suEXEC & POSIX ACL can prevent malicious CGI scripts of other customers that share the same server from stealing, deletion, or tampering with content.

2.2. Server Embedded Interpreters

Recently, dynamic content like weblogs, wikis, and CMSs are widely used. A CGI has been used to execute dynamic content. However, processing dynamic content at high speed is difficult for a CGI because it requires fork() and execve() for every request. Therefore, server embedded interpreters have been used as an alternative to CGI. They have server processes including interpreters of language processors. Server processes are used for many requests without termination. PHP [1] scripts are usually executed by a server embedded interpreter not as a CGI. Similarly, server embedded interpreters are widely used as an alternative to CGI, such as Ruby, Perl, and Python [2, 3, 4].

However, server embedded interpreters cannot be used together with suEXEC. suEXEC has server processes terminate after each request because it is a CGI. Because embedded scripts that are executed by server embedded interpreters run under the privilege of the dedicated user, it cannot ensure the security in a server (Figure 2).

2.3. Harache

We previously proposed a web server, Harache [13, 14]. It allows safe and convenient use of server embedded programs. In Harache, server processes run under the privilege of different users for every site (Figure 3). Therefore, Harache must grant permission of any content that includes embedded scripts only to an “owner”. However, Harache cannot fully use the increased speed of server embedded interpreters because server processes terminate after each session.

Hi-sap allows high security in servers in the same way as Harache. In addition, it has high performance when processing dynamic content by fully using the increased speed of server embedded interpreters.

2.4. Shared Hosting Service

In shared hosting services, customers subscribe to hosting service providers for every site. They can store content on the disk space of the server by paying a small monthly fee.

---

3Data files of weblogs and wikis additionally require write permission.
The biggest problem for hosting service providers is server footprints at the data center. Many customers must be accommodated in a server to make an acceptable profit. However, it is general that the amount of the data transfer is limited in shared hosting services. There are many personal sites that the amount of access is little. There could even be sites that are not accessed at all.

Therefore, the number of customers accommodated on a server must be maximized by minimizing the computation resources for sites that are not accessed at all.

In shared hosting services, a charging system exists according to the amount of the data transferred. Dynamic content that transfers little data and uses a lot of CPU power and memory is executed for a nominal fee. This is not fair, because the dynamic content can cause harmful effects for other sites that share the same server.

3. Design

In this section, we describe the design of our proposed server system, Hi-sap. It can be used with UNIX-like OSes.

3.1. Design Principle

The design principles of the system are as follows.

- High security: Server processes run under the privilege of different users for every partition.
- High performance: The system pools server processes that run under the privilege of the different users.
- High scalability: Server processes are created and terminated dynamically.

First, web objects that are stored in a server are divided into partitions. Each partition is a site, a content, and a QUERY_STRING\(^4\) according the division method. To have high security in a server, server processes run under the privilege of different users for every partition in the same way as Harache.

Second, the system pools server processes that run under the privilege of the different users. This is different from Harache whose server processes terminate after each session. Therefore, the system uses the increased speed of server embedded interpreters.

Third, the system creates and terminates server processes dynamically. As described in section 2.4, the scalability of the number of partitions in a server is an important factor in shared hosting services. In a web server, memory use strongly influences scalability [13]. Therefore, in the system, we tried to save memory with a web-server-level scheduler, Content Access Scheduler. Because it creates and terminates server processes dynamically, the system can have high scalability of the number of partitions in a server.

3.2. Architecture

The system brings access control into operation with a secure OS [15]. If the privilege of the administrator account is taken over because of a security hole or a misconfiguration, the access control of Hi-sap with different user privilege has no effect. Every web object is stolen, deleted, or tampered with by outside attackers. To prevent these incidents, the system ensures the security for each partition by using a secure OS.

An overview of the architecture of the system is shown in Figure 4. The system consists of a dispatcher and many workers. Each worker runs under the privilege of a different 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 in a transition of user privilege. If the policy of secure OSes permits workers that run under the privilege of the administrator account to transfer to the privilege of ordinary users, then the security will have no effect. Therefore, in our system, workers run under the privilege of ordinary users initially.

3.3. Content Access Scheduler

We propose Content Access Scheduler, a web-server-level scheduler to enhance the scalability of the number of partitions in a server. It controls the creation and termination of workers. Its scheduling principles are as follows.

\(^4\)An argument given to a CGI script and an embedded script.
Workers are created when a request occurs. Running workers are terminated when the server is in a high-load state.

The scheduler can allow high scalability, in particular by optimizing the scheduling algorithm for content.

3.4. Partitions

As described in section 2.4, customers subscribe to hosting service providers for each site. Also, current charging systems in shared hosting services do not take into account computation resource use.

In our system, the basic partition is a site. In Figure 5, worker A is dedicated to process requests for website A. Similarly, worker B is dedicated to website B, worker C is dedicated to website C. Because server processes run under the privilege of different users for each site in the system, the computation resource use of each site can be easily measured. Also, the system can bring limitations of the computation resources for each site into operation.

In addition, the system has the partition extend a content and a QUERY_STRING. In Figure 6, website A in 5 is divided into content. Worker A is dedicated to process requests for a wiki and a weblog, and worker A2 is dedicated to a CMS.

The measurement of the computation resource use and the limitations of computation resources can be computed for each content. Therefore, even if a server process that processes a request for a content runs out of control, other content in the site is not affected. Also, QUERY_STRINGs are used for page switching and operations, for example browse, edit, preview, and save, in weblogs and wikis. The system provides these fine-grained access controls at the web server level.

4. Implementation

In this section, we describe the implementation of Hi-sap based on the design of the previous section. We implemented the system on a Linux OS. The dispatcher was implemented as an Apache module, mod_hisap on Apache HTTP Server ver. 2.0.55 [9]. 1,000 Apache HTTP Server ver. 2.0.55 were used as workers. Each worker waits for requests at a unique port. Also, Content Access Scheduler and other management facilities of workers were implemented as a daemon, hisapd. An overview of a request processing of the system is shown in Figure 7. The details of the dispatcher and hisapd are as follows.

4.1. Dispatcher

When the dispatcher receives a request, for example for partition C in Figure 7, from a browser (Figure 7 (1)), it confirms whether the dedicated worker for partition C is active (Figure 7 (2)). If the worker (worker C) is inactive, the dispatcher asks hisapd to activate it (Figure 7 (3)). The communication method between the dispatcher and hisapd is a UNIX domain socket. Then Worker ID, an identifier of the requested worker, is recorded in a dedicated log file, Worker Request Log. After hisapd activates the worker (Figure 7 (4)), the dispatcher forwards the request to the worker (Figure 7 (5)). The worker processes the request (Figure 7 (6)). Then the dispatcher receives a response from the...
worker (Figure 7 (7)) and sends the response to the browser (Figure 7 (8)).

4.2. hisapd

As described in section 4.1, hisapd dynamically activates workers after requests from the dispatcher. This is the algorithm of Content Access Scheduler for worker activation.

In addition, an algorithm exists for worker termination. When thrashing occurs, hisapd terminates workers that have not been requested recently. Thrashing decreases the performance of web servers dramatically [13]. The conditions for which hisapd judges that thrashing occurs are as follows.

- A swap-in occurs.
- A swap-out occurs.
- Memory use is 99% or more.

hisapd checks for these conditions every 5 seconds. When all of the conditions are met, hisapd terminates workers. Also, the conditions for which hisapd chooses workers to terminate are as follows.

- The worker is active.
- The worker is not recorded in the most recent 10,000 requests of the Worker Request Log.

Pseudo LRU is used to reduce the search time for the Worker Request Log. In Figure 7, hisapd chooses worker A (Figure 7 (i)) and terminates it (Figure 7 (ii)).

5. Evaluation

In this section, we present the results of our evaluation experiments of Hi-sap. The hardware configuration of the experimental environments is shown in Table 1.

5.1. Basic Performance Evaluation

We evaluated the basic performance of Hi-sap in processing dynamic content to determine its usefulness. An Apache HTTP Server ver. 2.0.55 (Apache) and Apache that enables suEXEC (suEXEC) were used for comparisons. In the system and in Apache, a PHP script is executed by the server embedded interpreter. However, in suEXEC, a PHP script is executed as a CGI. The system, Apache, and suEXEC used the configuration files by default. We used a httperf benchmark ver. 0.8 [16] to measure the performance.

We sent requests to a PHP script and measured the response throughput. The script calls phpinfo() that displays the system information of the PHP language processor. The traffic of the script is 40 KB per request. The results are shown in Figure 8. The x-axis shows the request frequency, and the y-axis shows the throughput. The system loses an average of 28.0% of the throughput relative to Apache. However, the system has high throughput relative to suEXEC. The overhead of the system is because of a reverse proxy. Therefore, this implementation is effective.

This experiment demonstrates the system has high performance while ensuring security in a server.
Table 1. Hardware configuration of experimental environments

<table>
<thead>
<tr>
<th></th>
<th>Network</th>
</tr>
</thead>
<tbody>
<tr>
<td>Switching Hub</td>
<td>DELL PowerConnect 2724</td>
</tr>
<tr>
<td></td>
<td>1000 BASE-T ×24</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th></th>
<th>Client</th>
</tr>
</thead>
<tbody>
<tr>
<td>CPU</td>
<td>Intel Pentium III Xeon 500 MHz ×4</td>
</tr>
<tr>
<td>Memory</td>
<td>256 MB (swap 512 MB)</td>
</tr>
<tr>
<td>OS</td>
<td>Fedora Core 4 (Linux 2.6.14)</td>
</tr>
<tr>
<td>NIC</td>
<td>Intel PRO/1000XT 1 Gbps</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th></th>
<th>Server</th>
</tr>
</thead>
<tbody>
<tr>
<td>CPU</td>
<td>AMD Opteron 240EE 1.4 GHz ×2</td>
</tr>
<tr>
<td>Memory</td>
<td>4 GB (swap 8 GB)</td>
</tr>
<tr>
<td>OS</td>
<td>Fedora Core 4 (Linux 2.6.14)</td>
</tr>
<tr>
<td>NIC</td>
<td>Broadcom BCM5704C 1 Gbps</td>
</tr>
</tbody>
</table>

5.2. Scalability Evaluation

We evaluated the scalability of Hi-sap in processing dynamic content. The One-to-one approach was used for comparison. One-to-one uses networks with a reverse proxy, and has a dispatcher and many workers that are dedicated to process requests for each partition. Although One-to-one is similar to our system, mod_hisap and hisapd are not installed. All of the workers run from beginning to end. This experiment is intended to determine the effectiveness of Content Access Scheduler.

We sent 100 requests to a PHP script on each partition sequentially and measured the response throughput. The script was the same as that described in section 5.1. We used Apache HTTP server benchmarking tool ver. 2.0.41-dev included with Apache. The results are shown in Figure 9. The x-axis shows the number of partitions in the server, and the y-axis shows the throughput. Our system had substantially higher throughput than One-to-one from beginning to end. Also, the throughput decrement in the system due to an increase in the number of partitions was low. For One-to-one, the OS crashed due to a memory shortage when the number of partitions was 600.

The change of the memory use in the experiment is shown in Figure 10. The x-axis shows the number of partitions in the server, and the y-axis shows the memory use. The swap use of One-to-one dramatically increased due to an increase in the number of partitions. This is the reason of the crash of the OS. However, our system does not use swap space as much because of Content Access Scheduler.

This experiment demonstrates the system has high scalability for the number of partitions in a server.
Table 2. Comparison of approaches

<table>
<thead>
<tr>
<th></th>
<th>Security in a Server</th>
<th>Basic Performance</th>
<th>Scalability</th>
<th>Generality</th>
</tr>
</thead>
<tbody>
<tr>
<td>Apache</td>
<td>very bad</td>
<td>very good</td>
<td>good</td>
<td>good</td>
</tr>
<tr>
<td>suEXEC &amp; POSIX ACL</td>
<td>good</td>
<td>very bad</td>
<td>good</td>
<td>good</td>
</tr>
<tr>
<td>Sandbox / VM</td>
<td>very good</td>
<td>very good</td>
<td>bad / very bad</td>
<td>good</td>
</tr>
<tr>
<td>PHP safe mode</td>
<td>good</td>
<td>very good</td>
<td>good</td>
<td>very bad</td>
</tr>
<tr>
<td>Apache perchild MPM</td>
<td>good</td>
<td>–</td>
<td>bad</td>
<td>good</td>
</tr>
<tr>
<td>Harache</td>
<td>good</td>
<td>bad</td>
<td>good</td>
<td>good</td>
</tr>
<tr>
<td>Hi-sap</td>
<td>very good</td>
<td>good</td>
<td>good</td>
<td>good</td>
</tr>
</tbody>
</table>

6. Related Work

In this section, we describe the related work about the security in a server and discuss approaches for creating a server system.

6.1. Sandboxes and VMs

Many sandboxes [17] and virtual machines (VMs) [18] have been proposed. These mechanisms isolate server software and OSes running server software from the rest of a server machine.

If a sandbox or a VM is assigned to every partition, it has high security in a server. However, the computation resource use per partition dramatically increases by using these mechanisms. This strongly influences the scalability of the number of partitions in a server.

6.2. Language Processor

PHP [1] has a safe mode. This mechanism tries to create high security in a server by restricting the operations of PHP scripts. The restriction items are as follows.

- File handling is permitted only when the owner of the script is 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. Also, this mechanism is hard to use.

However, our system provides a general security mechanism.

6.3. Apache

perchild MPM is included in Apache HTTP Server ver. 2.0 [9]. In this mechanism, the user account and group account that executes server processes can be set for each site. Although this mechanism may create high security in a server, no reports show that it runs stably, and its development ended while still in the experimental phase. Also, dedicated server processes must be created initially for every site. This influences the scalability of the number of partitions in a server.

6.4. Discussion of Approaches

A comparison of the approaches of the different server systems is shown in Table 2. Normal Apache cannot have high security in a server. suEXEC & POSIX ACL has very low performance. Sandboxes and VMs have low scalability. PHP safe mode has low generality. Apache perchild MPM has low scalability and its performance is unknown because no reports show whether it runs stably. Harache has high security in a server, high scalability, and high generality. However, its performance is low.

Our system gets high marks in all items and does not have any weak points. Therefore, it is the most effective.

7. Conclusion

We designed Hi-sap, a secure and high-performance web server system, and we implemented it on a Linux OS. We described the security in a server and shared hosting services and mentioned their problems. We evaluated our system. Our results demonstrate the system has an advantage over other approaches in performance and scalability.

Acknowledgments

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

References