Confining the Insider Threat in Mass Virtual Hosting Systems

Marco Prandini, Eugenio Faldella and Roberto Laschi

DEIS, Università di Bologna, Viale Risorgimento 2
Bologna, Italy

Abstract. Mass virtual hosting is a widespread solution to the market need for a platform allowing the inexpensive deployment of web sites. By leveraging the ever-increasing performances of server platforms, it is possible to let hundreds of customers share the available storage, computing, and connectivity facilities, eventually attaining a satisfying level of service for a fraction of the total cost of the platform. Since the advent of dynamic web programming, however, achieving a sensible tradeoff between security and efficiency in mass hosting solutions has become quite difficult. The most efficient and widespread solution, in fact, foresees the execution with undifferentiated rights of code belonging to different customers, thus opening the possibility of unauthorized access of one customer to the others' data. This paper illustrates a possible solution to this problem, based on the integration of Mandatory Access control techniques within the web server. The proposed solution guarantees robust isolation between resources belonging to different subjects, without introducing a sensible increase in resource utilization.

1 Introduction

The traditional approach to hosting sites on a server relies on the classical access control schemes provided by modern operating systems. In a scenario where the server is called to handle only static websites, these schemes offer both a sufficient level of separation between customers, by authenticating them when they manage their files on their filesystem share, and a satisfactory level of efficiency, by serving all of the contents through a single, optimized set of processes. The latter detail explains the notion of virtual hosting: in a system working as described, not even the web server process is dedicated to a specific customer; instead, a single program leverages the capabilities of the HTTP/1.1 protocol [1] to distinguish between sites with different names even if all the requests come to a single address. However, this approach is unsuitable for the hosting of dynamic sites, i.e. when the customers are allowed to upload programs that run on the server in order to generate content on-the-fly, rather than static files which are sent to the browsers as they are. This scenario raises the problem of implementing an effective access control scheme also in the web server process, possibly without compromising the resource sharing efficiency.

In the following section, we analyze the possible approaches to the implementation of a secure virtual hosting system, discussing their security and efficiency characteristics. Subsequently, we illustrate how the integration of Mandatory Access Control systems within the most efficient solution can lead to an optimal balance between security and performance. Finally, we present a prototype implementing the proposed solution, based on the ubiquitous Linux/Apache/PHP platform and on the prominent SELinux security module.

2 Performance and Security Issues in Mass Virtual Hosting

The main concern in a web hosting server regards scalability, i.e. the capability to power a large number of sites without causing unacceptable performance degradation. The obvious method to pursue this result is minimizing the amount of resources needed just for making each site work, leaving as many resources as possible free for serving the actual requests hitting the sites. In a multitasking operating system, several kinds of resources must be allocated for a web server process to work: memory, file descriptors, IP addresses, processor time, and bandwidth are just the key examples. In order to minimize the static allocation of these resources to a single website, they have to be shared between all.

Sharing is possible only with some degree of cooperation between the processes associated to the different sites, which could mean lack of isolation between the resources belonging to different customers. A viable solution, then, must implement an efficient sharing model but also execute the processes related to each site according to the least privilege principle. The most commonly adopted solution, however, almost completely fails to provide this kind of guarantee: there is a single process serving every virtual site by spawning sub-processes or threads. In order to limit the effects of bugs or intrusions, this process runs as an unprivileged user, and consequently it cannot assume different identities for the different sites. The chosen user, then, must have read access to every site's files, and consequently a dynamic page belonging to a hosted site, when invoked, can perform local operations with enough privileges to read other sites' sensitive data.

In the following, more sophisticated solutions are described, showing how higher security can be attained at the cost of lower efficiency. The analysis of the possible solutions will make reference to Apache [2], the world's most used web server [3]. It exhibits a modular structure [4, 5] allowing to extend the core functionalities so as to support almost any need.

2.1 Virtual Machines

Serving different sites through separate virtual machines (VMs) offers the highest possible degree both of mutual isolation between sites and of protection of the host system from intrusions exploiting the web server processes. The price to pay is sensible resource duplication, because in this case an entire "guest" operating system is run within a "host" platform. The performance impact in terms of CPU usage is moderate (in the order of 2-3% for the widespread VMware [6] and Xen [7] platforms), but the memory footprint even of an idle VM is quite high: in our tests, each VMware-based site absorbs nearly 70 Mbytes just for making its VM work, requiring as much memory as roughly 35 web server processes would.

2.2 Separate Processes within the Host System

A good level of isolation can also be obtained by serving the different sites through separate process trees, each one running under its own identity. The overall security level of this solution is lower than what can be achieved with VMs, because the effective separation both between the different processes, and between each process and the host system is enforced by one operating system, with visibility of every resource. Any kind of attack directed to a web server process, which succeeds in granting the attacker a privilege escalation, allows breaking all the barriers.

As previously noted, the memory occupation gain over the VM solution is sensible. However, a single process tree serving all the virtual sites is much more effective at pooling available resources, for two main reasons. First, a simple test performed on a stock distribution shows that in every new, master Apache process takes up approximately 2.5 Mbytes more than a child process spawned by an existing master. Second, spare child processes are needed in order to achieve acceptable performance during peak times. On a single-process-tree system a child process can be allocated on any site, while, when separate process trees are used, each site has to keep its share of spare sub-processes. Their total number consequently increases for each added site.

2.3 Identity Change of the Sub-processes

In order to get a more efficient resource pooling, it is possible to envisage a mode of operation where a child sub-process, spawned from a single master to serve a request, assumes the specific identity related to the corresponding site (in the Apache project, an implementation of this model is available as mod_perchild; unfortunately, it is not functional). This approach solves the first of the two inefficiencies associated to the previous solution. However, after completing its duty, a sub-process can be reused for the same site only, because it cannot switch to a different identity. Eventually, this approach is slightly less secure than running separate processes, because of possibility of attacking the master process running with administrative privileges, and only marginally more efficient, because it does not solve the problem of the excess of spare sub-processes.

2.4 Execution of Dynamic Content Outside of the Web Server Process

The generation of dynamic contents, also called server-side programming, is actually the only activity performed by the web server which needs strict access control. It is commonly handled in two different ways, with peculiar reflexes on security.

CGI (Common Gateway Interface) [8] is the oldest model for the implementation of dynamic pages on web servers. Basically, as its name suggests, it defines a standard interface between the web server process and an external program it invokes.

The web server acts almost as a pure gateway, passing all the data received from the browser to the external program, and passing the program's output back.

By adding an intermediate set-user-id wrapper program it is easy to execute the CGI programs with the site-specific credentials, instead of changing the credentials of the web server process. A comparison between this mediated CGI execution and a solution based on changing the identity of the web server sub-processes shows that, on the efficiency side, the first solution has the advantage of exploiting undifferentiated (thus reusable) Apache processes, while on the security side it introduces another potentially exploitable component, that is the wrapper executing with root privileges. This problem is mitigated by the fact that, conversely to Apache, the wrapper is an extremely simple piece of code, which should be easily security-tested.

The vast majority of dynamic pages are, however, mostly composed of fixed HTML elements, with some dynamically-generated information inserted in between. CGI programming can be uselessly cumbersome in this scenario: the program has to deal with countless details, which often prevail over its specific function. For this reason, server-side scripting has known a great success as a newer model for the implementation of dynamic pages. It works by tightly integrating the language used to program dynamic behavior with the web server, making the latter able to parse a page while serving it. When, at some point in a page, the server recognizes the special tags marking embedded instructions, it executes them and inserts their results at the same point within the data flow. From the security point of view, isolation is obviously more difficult when this approach is chosen over CGI. Being the scripting engine actually part of the web server, isolation between virtual sites could only be attained by separating the web server processes, by means of one of the aforementioned techniques (i.e. different servers, different virtual machines, or per-child user id assignment). Some third-party addition pursues the goal of getting the best of both worlds, that is, being able to develop sites according to the simpler server-side scripting model, and executing the resulting code outside of the web server, like a CGI program. We describe one of these additions in section 4, showing our prototype of a secure hosting server.

3 MAC/TE Applications to Hosting

In addition to the illustrated analysis of the possible security scenarios, we should note that any approach based on a simple identity change (of the process in charge of generating a dynamic page) leaves many potential security problems open. The discretionary access control model implemented on Linux leaves many options available to a malicious customer for trying to compromise the server's security or simply misuse the server resources.

Several freely available systems implement various kinds of “hardening” techniques in the Linux operating system, making it an ideal choice for building a secure, efficient and economical hosting platform. The functionalities added by these systems range from enhanced access control models like Mandatory Access Control (MAC) to code execution checks.

Among the most important projects, we can recall:

- **RSBAC** – Amon Ott began to develop the Rule Set Based Access Control project [9] in 1996, with the goal of providing a Linux implementation of the Generalized Framework for Access Control (GFAC) by Abrams and LaPadula [10]. It implements MAC functionalities and several other security modules.
- **LIDS** – The Linux Intrusion Detection System [11] is an integration to the Linux kernel exclusively aimed at implementing MAC functionalities.
- **grsecurity** – This project, started by Brad Spengler around the year 2002 [12], implements MAC as Role-Based Access Control (RBAC), and several other functionalities aimed at containing the effects of compromised processes on the system.
- **SELinux** – The Security-Enhanced Linux project [13] is developed by the National Security Agency (NSA) and released as open-source. It implements MAC as Role-Based Access Control (RBAC) and other sophisticated security models.

Each of the listed projects would be worth a deep analysis. In this work, we chose to build a prototype based on SELinux for two main reasons. First, it is the most theoretically sound and carefully validated one, thanks to the NSA leading the project. Second, it is the only one that provides an Application Programming Interface (API) allowing SELinux-aware programs to fully exploit the potential of its features, whereas the other projects provide only a static, configuration-based behavior.

3.1 SELinux Basic Concepts

The foundation of SELinux is the Flask architecture [14], the result of a decade of research on the subject of MAC lead by the National Security Agency (NSA), the Secure Computing Corporation, and University of Utah.

The origin of the MAC concept, in turn, can be traced back to the research activity undertaken in the seventies by Bell and LaPadula, with their formal description of the Multi-Level Security System (MLS) [15]. Since then, access control is regarded as a key property in systems security, and finding an efficient and effective implementation for the MAC model has been an important task. On a MAC-based system, the security manager can finely tune which resources are available to which users and processes, and enforce the chosen policies in a way that cannot be circumvented, not even by the system privileged account.

After the conclusion of the aforementioned theoretical studies, the NSA proceeded to implement the Flask architecture within the Linux operating system [16], releasing the first prototype of SELinux in year 2000 as open-source. The project is undergoing continuous development since then.

The architecture of Flask, and thus of SELinux, encompasses two components. The Security Server contains the definition of every security policy, and takes decisions accordingly. The Object Managers (one for each OS subsystem) enforce the policies by querying the Security Server for each relevant action.

Four different models cooperate to define the access control mechanism implemented by the Security Server [17]: Type Enforcement (TE) [18], Role Based Access Control (RBAC) [19], User Identity (UI) and, rarely used, Multi-Level Security (MLS).

The key component for the proposed application is TE. According to the TE model, each subject on the system has an associated security attribute called domain, and similarly each object on the system has a type. Interactions between subjects, or actions performed by subjects on objects are controlled by an access control matrix stating the rights of a given domain when dealing respectively with another domain or a type. Each subject/object is associated with a label called security context composed of three attributes (user, role and type/domain). The Security Server works under a closed-world policy, meaning that permissions not explicitly granted are denied, so a security context is effectively taken into account for mapping the associated subject on its domain only when the subject is invoked by the listed user, acting under the specified role. After this mapping has taken place, the Security Server bases its security decision (i.e. decides whether granting the subject access to the object or not) on the third field of the context only (type being a synonym of domain for subjects). The Security Server makes also another kind of decision, labeling decisions, choosing the proper security context to be assigned to an object, typically at its creation time. The configuration of SELinux, according to the described models, encompasses two actions: the labeling of each object with the right security context, and the definition of type enforcement policies.

3.2 Enhancing the Security of CGI Execution

Many Linux distributions already leverage the MAC layer enforced by SELinux in order to confine Apache (as well as the other services) to a specific domain. Without further configuration actions, any process spawned by Apache will run within the same domain, thus inheriting the same capabilities. As already noted, this is not desirable, since in general dynamic page generation requires much lower privileges than the web server. Enforcing a precise and effective limitation of the capabilities associated to the sub-processes is particularly important for the wrapper-based models of CGI execution, which momentarily gain unrestricted powers (with respect to the standard Linux access control model) in order to eventually switch to the identity associated with the virtual site.

To achieve both proper privilege reduction and isolation between virtual sites, it is possible to define a different domain for each virtual site, and induce a domain transition from the Apache starting domain to the site-specific one. The transition can be handled in three different ways: two requiring the modification of either the module component of the wrapper subsystem (the one which, tightly integrated within Apache, makes it aware of the wrapper) or the actual external wrapper program, in order to make them SELinux-aware, and one exploiting the configurable SELinux automated labeling capabilities.

3.2.1 Domain Transition Induced by the Module

A suexec-like module, when loaded, becomes part of the running Apache process, and handles the creation of the child wrapper process when needed. A simple modification to the module code allows to invoke the SELinux API function setexeccon() just before child creation. This function sets the context that SELinux will create the child processes within, and thus allows to directly start the wrapper process within the domain associated to the virtual site to be served. The context is reset to its original value after the external execution has ended, allowing correct reuse of the Apache process for other requests.

This is probably the optimal strategy, security-wise, because the wrapper never runs in any other domain than the site-specific one. It requires modifying a code portion which runs within the Apache process, so theoretically, if its implementation is flawed, it could add a vulnerability to the whole server. However, the patch is usually so small that a thorough security revision should not be absolutely difficult.

3.2.2 Domain Transition Induced within the Wrapper

The setcon() function, available within the SELinux API as well, allows to design an alternative transition model. When called from within the wrapper, it causes the context transition of the process, which leaves the generic Apache domain to enter the site-specific domain, much like the setuid() and setgid() system calls cause the process to assume the identity related to the virtual site to be served.

This approach has the only, negligible advantage of modifying a component which is not part of the main Apache process. However, it requires passing the final domain (read from the Apache configuration file) to the wrapper process through the environment, thus complicating the code and making more correctness checks necessary.

3.2.3 Policy-driven Domain Transition

It is possible to configure SELinux, by means of the domain_auto_trans macro, for executing a program within a specified domain instead of inheriting the domain of the caller. The domain is determined within the policy file in function of the type label associated to the executable file. By exploiting this feature, the wrapper process can execute the correct domain transition without introducing changes to either the module or the wrapper itself. There is a drawback: since a file can only be labeled with a single type, in order to attain transitions to different domains (one for each virtual site) it is necessary to create a separate, uniquely labeled copy of the wrapper executable for each one.

3.3 Enhancing the Security of PHP Execution

PHP [20] is probably the most commonly used language for the implementation of dynamic web sites [21]. It can be deployed within Apache either as a module implementing the server-side scripting functionality, or as a CGI program. The former approach is usually preferred, but the peculiar property of supporting both models with very small changes to pages can be leveraged to attain both the security advantages of the CGI model and the practicality of server-side scripting at the same time. The suPHP project [22] pursues this goal. Its structure, similarly to suexec, is composed of two parts: a small Apache module which exposes the same functionalities of the full PHP module to the web server, and a wrapper program which invokes the actual PHP interpreter in CGI mode. These components take care of invoking the PHP interpreter so that (1) PHP pages written according to the server-side scripting model are correctly processed without changes, and (2) its credentials can be changed accordingly to the involved virtual site. Thus, we chose suPHP over a generic suexec-like, CGI executing package as a good candidate for building efficient and secure hosting platforms.

4 A Prototype of MAC-enhanced suPHP-based Hosting Server

The implementation of the prototype required two main activities: modifying the suPHP code in order to achieve the both the module-induced and the wrapper-invoked domain transition, and designing the proper configuration rules for SELinux.

4.1 Code Modifications, Configuration Issues

In the solutions based on suPHP code modification, the domain associated with the virtual site must be passed as a parameter to the suPHP module. As any Apache module, suPHP declares its configuration directives, so that, when the module is loaded, Apache is able to recognize them within its configuration file, to perform a formal check of their syntax, and to make the associated values available to the module. The original suPHP module declares a suPHP_UserGroup directive, specifying the standard Unix identity associated with the virtual site. The code has been modified in order to handle a suPHP_SeLinux directive too. The added directive allows specifying only the SELinux domain which the suphp wrapper will run within. All the instances will use the same user and role, because these attributes are not considered by the Security Server when making security decisions. A proliferation of useless users and roles would only make policy definition exponentially more complex. In the solution based on automatic, labeling-induced domain transition, the filename pointing to the correctly labeled wrapper copy must be passed as a parameter to the suPHP module. To avoid unnecessary duplications, a simple convention has been adopted regarding filenames: the wrapper copy which, by virtue of its type and of SELinux configuration, is going to run within a given *domain* has to be named /path/to/wrapper/suphp.*domain*. In this way, the same parameter suPHP_SeLinux can be used, and the module computes the filename as the concatenation of the fixed prefix and the passed domain.

4.2 Policy Definition

In order to highlight the relevant capabilities needed by the web server, and to make policies independent from the specific distribution (which, for the development of our prototype, was Fedora Core 5), a specific apache_suphp_t domain has been defined for standard Apache operation. As already noted, security decisions depend on the domain only, thus no new users and roles have been defined, using system_u and system_r which are the default for executing system daemons. The simplest configuration activity consisted in preparing the common TE directives giving the apache_suphp_t domain access to all the relevant Apache+suPHP subsystem files. The specific policy for the concession of proper capabilities to each domain is much more complex. Moreover, every time a new virtual site is added, the corresponding policy must be added to SELinux configuration. Consequently, theoretical analysis and experimental validation was performed on a single site and lead to the policy definition for a specific domain, then the policy file has been rewritten as a template, by inserting a <<<Domain>>> parameter which is substituted with the actual value by an installation script. The main template does not contain the essential rules allowing the domain transition from apache_suphp_t to SuPhp_<<<Domain>>>_t, because their implementation depends on the chosen transition model and thus are integrated from specific sub-templates as requested during system configuration.

4.3 Experimental Results

The efficiency of the proposed prototype has been tested, in order to estimate the potential performance hit introduced by the complex checks enacted by the SELinux system. The results are easily summarized:

- the increment of memory occupation is negligible;
- the increase in the average response time, measured over several tests with the apache benchmark (ab) tool, is very small (a few percents), thanks to the Access Vector Cache (AVC) component of SELinux, which stores the security decisions after the first time they are taken;
- the response time variability, however, tends to be somewhat higher than what can be observed on a plain Apache server, probably for a bottleneck effect introduced by SELinux.

5 Conclusions

When relying on the traditional access control models implemented by the most common operating systems, designing a configuration for a web server shared among several users that balances security and performance can be very difficult. The administrator is faced with the challenge of choosing a layout of ownerships and permissions allowing each webmaster to work with its files, making them readable by the web server, but keeping them confidential with respect to other sites. Any small mistake can either prevent a site from working or make its sensitive data easily available, and within a DAC system there is no guarantee of stability, since each user can change the permissions of its own files, either maliciously or mistakenly.

The presented work shows a practical application to this context of the powerful access control model implemented by SELinux. By means of the provided Type Enforcement functionalities, it has been possible to achieve effective isolation between virtual hosts served by a single process tree. Simple tools assist the administrator in configuring the security policies, which are then mandatorily and automatically enforced. The described suPHP-SELinux-based system has been fully implemented and tested, both verifying the correspondence between attended and real behavior of the domain transition policies, and measuring the impact on performance, obtaining very satisfying results on both fronts.

Current work is aimed to extend the analysis to the alternative systems, verifying the feasibility of defining a common framework for the configuration of secure multi-user web servers on different platforms.

References

1. Hypertext Transfer Protocol - HTTP/1.1 - http://www.ietf.org/rfc/rfc2616.txt
2. Apache Server website. - http://httpd.apache.org/
3. Netcraft Web Server Survey. - http://news.netcraft.com/archives/web_server_survey .html
4. Apache: Conceptual Architecture by Ahmed Hassan. - http://plg.uwaterloo.ca/~aeehassa/cs746/as1/apache1.html
5. Extending Apache: Apache Modules. - http://apache.hpi.uni-postsdam.de/document/3_3Extending _Apache.html
6. VMware web site. - http://www.vmware.com/
7. Barham P., Dragovic B., Fraser K., Hand S., Harris T., Ho A., Neugebauer R., Pratt I., and Warfield A., Xen and the art of virtualization. Proc. 19th ACM symposium on Operating systems principles, October, 2003, ACM Press, 162-177
8. Common Gateway Interface v1.1. - http://hoohoo.ncsa.uiuc.edu/cgi/
9. RSBAC web site. - http://www.rsbac.org/
10. La Padula, L. J., Rule Set Modeling of a Trusted Computer System, Essay, in: Information Security: An Integrated Collection of Essays, Hrsg.: Abrams, M. D., Jajodia, S., Podell, H. J., IEEE Computer Society Press, 1995
11. LIDS web site. - http://www.lids.org/
12. grsecurity web site. - http://www.grscurity.net/
13. National Security Agency. Security-Enhanced Linux (SELinux). - http://www.nsa.gov/selinux
14. Spencer R., Smalley S. D., Loscocco P., Hibler M., Andersen D. and Lepreau J., The Flask Security Architecture: System support for diverse security policies, Proc. 8th USENIX Security Symposium, Washington, D.C., 1999, pp 123-139
15. D. E. Bell and L. J. LaPadula, Secure Computer Systems: Mathematical Foundations and Model, Technical Report M74-244, The MITRE Corporation, Bedford, MA, May 1973
16. Smalley S., Vance C. and Salamon W. Implementing SELinux as a Linux Security Module - http://www.nsa.gov/selinux/papers/module.pdf
17. Smalley S. D., Configuring the SELinux Policy. Nai Labs Report #02-007, June 2002
18. Badger L., Sterne D. F., Sherman D. L., Walker K. M. and Haghighat S. A., A Domain and Type Enforcement Unix Prototype, Proc. 5th USENIX UNIX Security Symposium, Salt Lake City, UT, 1995, pp 127-140
19. Sandhu R., Role-Based Access Control, Advances in Computer Science, 46, Academic Press, 1998
20. PHP website. - http://www.php.net/
21. PHP usage stats. - http://www.php.net/usage.php
22. suPHP Project by Sebastian Marsching - http://www.suphp.org/