Production Systems and Information Engineering
Volume 6 (2013), pp. 69-80

A FAST METHOD FOR SECURING USER SUPPLIED CODE EXECUTION IN WEB SERVERS

DÁVID VINCZE
University of Miskolc, Hungary
Department of Information Technology
david.vincze@iit.uni-miskolc.hu

[Received January 2012 and accepted September 2012]

Abstract. This paper presents a novel method for securing web servers while keeping performance overhead as low as possible. There are already existing methods for separating the execution user and group identities for different websites on the same web server, hence improving security, but all of them have performance and resource usage weaknesses. The mechanism presented here requires modifications also in the kernel of the operating system, not only in the web server. The method works by extracting the calling address of the process invoking a specific newly implemented system call in the Linux kernel. Based on this address, the new system call can decide whether to grant additional privileges to the invoking process or not. Integrating this method into the Apache web server (the most popular web server application for many years as of writing) makes it possible to create a secure environment for the different websites belonging to different users on the same server. Compared to similar solutions, the overhead of the method is very low. The last chapter presents measurement results comparing the performance of the original version and that of the modified version of the web server.

Keywords: operating systems security, Linux kernel, web server security, Apache web server

1. Introduction

Web based applications are getting more and more widespread nowadays. In most cases on multi-user systems for performance and resource benefits, separate web applications are served by the same web server running with the same user identities and permissions. Hence the served applications are equal when it comes to accessing resources, independently from which user they belong to, because they are served under the identity of the web server. In some cases this can be uncomfortable for the users, moreover malicious users could exploit this easily, e.g. they can access each other's source code, database access passwords, in some

cases they could disturb the web server causing it to malfunction, hide backdoors, etc. [5] [14]

In production environments these problems emerge quite soon if the given web server has many users. Most commonly: uploaded files must be separately adjusted by the system administrator or the user to the correct privileges; working with the uploaded files can be difficult because the web server created these files with its own user identity, deleting these files via File Transfer Protocol (FTP) [11] often causes problems (because FTP separates identities); these files are accounted to the resources of the web server and not to the user who had the file uploaded; limitation and logging of resource usage cannot be distinguished among users; setting arbitrary headers in e-mails when sending mail using the *mail()* function in PHP [1] scripts; session manipulation [10] and other seemingly small, but serious problems [13] [5] [14].

As nowadays most of the web servers in production apply this structure and configuration, these problems affect a considerable number of users: internet service providers, educational institutions, various enterprises, etc.

This paper presents a possible solution to eliminate these problems, providing a fast and secure environment for web applications using runtime user and group identity switching.

Currently the Apache HTTPD is one of the most commonly used web servers [20], and most of its installations are deployed on Linux based systems. The Apache web server and the Linux kernel are both open source software, therefore it is possible to modify their source code freely. The presented solution has been developed for the Linux operating system in C and Assembly (x86 32-bit and 64-bit) programming languages.

First, the paper gives an overview of the operation of web servers at a glance, especially the operation of the Apache HTTPD, then of the possibilities of run-time user and group identity switching in the Linux kernel. A novel method is introduced to determine whether a privilege elevation request is legitimate or not. The modifications required for this new mechanism to work are also presented both in the Linux kernel and the Apache web server. Finally, results of performance benchmarks are presented.

2. Overview of the operation of web servers

In the beginning of the web-era, the first web servers were only able to serve regular files. This practically means that only static content was available, e.g. text files, documents, images, etc. These simple web servers only read the contents of the requested files and sent it to the client unmodified. This is called a static web server, which is still in use nowadays, because a great deal of static content can still be found (e.g. images, videos, stylesheets, JavaScript scripts, etc.) on regular web pages.

Nowadays it is common that web pages show dynamic content, which means the pages are generated on-the-fly, taking various input data into consideration (usually exchanging data with databases). In this case the web server, instead of sending the exact content, runs the specified program/script corresponding to the request, and sends back its output (the generated web page) to the requesting client. This is called dynamic serving mode.

The first widespread method for generating dynamic content for web pages was the Common Gateway Interface (CGI) mechanism [12]. The concept of CGI is simple: the requested file (CGI application) is executed on the web server and the output of the CGI application is sent as an answer to the request. Scripts can be also used as CGI application, in this case the script interpreter must be defined. The CGI application is executed in a newly created independent separate process, and after the application finishes, this process is also terminated. CGI is still in use, but not very common nowadays.

After the success of CGI, it was realized that CGI was mainly used for running script interpreters. Interpreters embedded in the web server were developed for various script languages (e.g. Perl, PHP, Python, Ruby, etc.). These were much faster because there was no need for forking new processes and initialization every time a request arrived. Also these embedded interpreters provide an environment which is easier for programmers to use. Nowadays these solutions are in use, superseding CGI.

According to the latest NetCraft survey [20], the most commonly used web server on public web sites is the Apache HTTPD. Most of these are installed on Linux-based servers.

Traditionally the Apache HTTPD uses the *prefork* model [21]. The prefork model has a control process which does not serve incoming Hypertext Transfer Protocol (HTTP) [4] requests, but only starts and monitors child processes, which do the actual work. A child process can serve many independent HTTP requests (configurable, some hundred on an average basis), hence it is not required to start a new process and perform initialization for every request, which yields a considerable performance gain.

The *worker* model [21] is similar to the prefork model, the difference being that it uses threads instead of processes. That is the main reason why it is not used in multi-user environments where users are not trusted.

Using either the prefork or the worker model, the user and group identities stay the same through every served request. This means that all programs (e.g. CGI) are running with the same permissions, which is considered a security weakness for various reasons, see e.g. [13].

For securing CGI executions there is a method called *suexec CGI* [21], which

extends the CGI mechanism with the capability of running each CGI program as a different user/group identity. The main idea is that CGI programs are executed through a wrapper program. This wrapper program has a special permission setting (suid – setuid, see [2] and [3] for details) allowing it to run with system administrator privileges. The wrapper checks which user/group identities must be set for the CGI program, and after changing the identities it loses all of the administrator privileges (hence the CGI program cannot revert to be administrator again), and the original CGI application will be executed. Injecting this wrapper program in the chain of execution on one hand increases security, but the serving requests becomes slower compared to normal CGI execution.

Also there was a model called *perchild* [21], which ran separate dedicated threads for every website configured. These separated threads or thread groups had their own configured user and group identities set, hence the requests were served with the correct permissions. A drawback of this concept was that a certain number (the number of the sites configured) of thread groups were always running even if they never served a request. In case of many configured websites this consumes resources unnecessarily. The main drawback of the perchild model is that the model was never finished, furthermore it was dropped from the newest versions of the Apache web server [21].

Another implementation of this concept was *metuxmpm* [16], but as with perchild, the development was cancelled. Then came the peruser model [9], which tends to be a working implementation, but it uses separate processes instead of threads. This means that in case of many websites the amount of unnecessarily allocated resources can be huge.

A promising model based on prefork is the *itk* model [6]. It can run the programs required to generate pages with their own identities, without using dedicated thread/process groups. All the children processes run with administrator privileges till the necessary information (which identities to set) can be extracted from an incoming HTTP request. After changing the user and group identities and serving the requests belonging to the same website (user id), the identities cannot be set back to administrator level, hence the process has to be terminated. This means that processes cannot be reused, new processes should be created if a new incoming request belongs to another website (user id). Thus compared to the prefork model, this module also has performance and resource usage drawbacks.

A similar, experimental solution called Harache has been proposed in [7]. Harache works very much like the *itk* model: sets the identifiers before processing the incoming HTTP request, and after processing the request, terminates the process.

Another solution from the same authors called Hi-Sap [8] uses a complex model consisting of a dispatcher/content scheduler and Apache worker pools. While the performance is better compared to Harache, the overall resource usage is

considerably higher.

Being aware of these facts, it is clear that system administrators have to make a choice between security and performance. As impacts of performance can usually be experienced directly (faster web pages, more customers can be handled by fewer servers, etc.), performance is chosen over security.

The next chapter introduces a novel method which retains both performance and security in the previously described situations.

3. Enhancing security while retaining performance

The main difficulty is to unambiguously determine whether the currently executed code in the process is a normal web server code or a user-written code (e.g. embedded script interpreter running a user's script), as a normal web server process does not have any properties which could be used to distinguish between these two states. For example the operating system kernel does not know whether the code of the Apache web server itself is executed, or a user code is being interpreted by e.g. the embedded PHP interpreter: *mod_php*.

Therefore, first a method should be developed which can distinguish between these two conditions inside a process. This paper presents a novel approach which is based on the calling memory address of a privilege elevating system call. In the possession of this calling address it becomes possible to determine whether the privilege elevation request is legitimate or not. If the request is allowed, then various privileges can be granted for the requesting process.

In this section a new system call implemented in the Linux kernel will be presented, which gives certain privileges to the calling process based on the return address to the user-space (also referred to as calling address earlier). Then the modifications required in the Apache web server to make use of this newly added system call will be discussed.

First, some concepts required for understanding the new method will be overviewed shortly.

3.1. The setuid() / setgid() system call family

On multi-user POSIX (Portable Operating System Interface) [17] compatible systems, users and groups posses their own identifiers (one user id - uid, and at least one group id - gid) for the purpose of separating the users. Authentication and access control are based on these identities. The following system calls are standardized in POSIX.1 [17]: setuid() / seteuid() / setreuid() / setresuid(). These syscalls allow an arbitrary process in the system to change their user identities. The same syscalls exist regarding changing group identities: setgid() / setegid() / setregid() / setresgid().

The case of normal usage is when a process is running with administrator privileges and after the application completes the procedures requiring administrator privileges, the process switches its user and group identities to unprivileged identities. These are carried out with calling the setuid()/setgid() system calls, hence it drops the administrator privileges and continues to run as a normal user process. See [2] [3] for more details.

The Apache web server works the same way as described. The system administrator privileges are needed only for binding to TCP ports below 1024 – by default port 80 or port 443 (HTTP / HTTPS ports) - and also for opening the log files. Once these tasks are completed, the web server switches to a normal user for the rest of its lifetime.

3.2. The Linux capabilities system

Access control of traditional UNIX systems (including Linux) distinguishes two privilege levels: one level for the system administrator (user identity equals 0) and another one for normal user level (user identity other than 0). Controlling access is very simple by default: at the system administrator level no checking is performed for accessing resources, but at the user level the current user and group identities are compared with the identities required by the desired resource.

The capabilities mechanism is described in chapter 25 of the POSIX.1e [18] standard. This allows finer granularity for permission distribution and access control. The POSIX.1e standard has never been finalized, only a draft has been published, and even that has been withdrawn later. Despite this fact, the capabilities mechanism has been implemented in some operating systems, e.g. Linux. The capabilities mechanism describes elemental privileges which can be assigned to processes, independent from their current user identities. Some of the implemented capabilities in the Linux operating system [15] are:

- CAP_NET_BIND_SERVICE: Allows binding to TCP/UDP ports under 1024.
- CAP_SETUID: Allows setting the user identity to an arbitrary value on the current process (allows the successful calling of the setuid() syscall family).
- CAP_SETGID: Allows setting the groups identity and supplemental groups to arbitrary groups on the current process (allows the successful calling of the setgid() syscall family).

Naturally, there are many more capabilities defined. By default, the system administrator possesses all the capabilities and normal users do not have any of the system capabilities. The proposal in this paper makes use of the CAP_SETUID and the CAP_SETGID capabilities.

3.3. The new system call

As described earlier, a new method has been implemented, which elevates process privileges based on the memory address in the memory area of the process where the system call was invoked from. A new system call was implemented in the latest Linux kernel (version 3.1.10 as of writing). Unfortunately the code is architecture specific, currently the implementation is for the x86 and the x86_64 architectures only. Details of the implementation are presented in this subsection.

Before a system call executes, the memory address of the next instruction to be executed (the return address) is stored on the stack before entering into the kernel. After the system call completes its task in the kernel, the control will be returned to the user-space application, which will continue its execution exactly at the memory address previously stored on the stack. The control returns to the saved memory address in the process of the web server with a *ret* instruction. The return address is popped from the stack, which was pushed to the stack earlier just before the actual system call. This return address has to be traced back on the stack to be used for authentication.

Tracing back the return address is done in the new syscall named *getmyretaddr*(). It does not have any input parameters as its only task is to check whether to allow the caller process elevated privileges or not. The newly implemented system call first traces back the return address on the stack to determine where exactly the system call came from the memory area of the calling process. If this is the first invocation of the new system call within a given process, the traced return address will be stored in the process context. Otherwise the traced memory address will be compared with the previously stored memory address by the first invocation. In case of a match, the privilege elevation will be successful, otherwise the privilege elevation request will be ignored.

In possession of the calling address, the access verification is realized as described in the following. In the case of the first invocation of the *getmyretaddr()* system call, a new attribute in the process context (called *retaddr* – default value is zero) will be initialized with the gathered return address. The initialization will only be successful if the process possesses the CAP_SETUID and the CAP_SETGID capabilities. Any next invocation of the *getmyretaddr()* system call checks the return address against the stored *retaddr*; if these two match, then the calling process will be granted both with CAP_SETUID and CAP_SETGID. Then the system call finishes and the control is returned into user space. Now it is the task of the privilege elevation requesting process to change the user and group identifiers, then drop the two capabilities.

The next chapter describes how this new system call has been integrated into the Apache web server.

4.2. Modifications in the Apache web server

The prefork module [21] was modified to make use of the presented mechanism. The required modifications will be described briefly in the following.

Right after staring the web server, initially forked child processes wait for incoming requests with the default user and group identities. The web server was modified to leave both CAP_SETUID and CAP_SETGID capabilities set on the inital processes. When a request arrives, the child process does its usual initialization and just before serving the request, the *getmyretaddr()* syscall will be invoked. If this is the first request served by the child process, the *getmyretaddr()* does only initialization (stores the valid return address in the process context: *retaddr*). The pre-forked child process should be in possession of the CAP_SETUID and the CAP_SETGID capabilities, otherwise the system call will be unsuccessful. Next, after successfully changing the user and group identities with *setresuid()* and *setresgid()* calls, the two capabilities are not required anymore, so they will be dropped. Finally the actual request will be served but now with the correctly set unique identities. When serving the forthcoming requests, the child process is no longer in possession of the two capabilities when calling *getmyretaddr()*, but if it is called from the permitted memory address (the first call's return address), then the syscall will be successful, and the kernel will grant the two capabilities to the calling process. Hence the *setresuid()* and *setresgid()* syscalls will be also successful when called. In case when the request is for another user's resource, then the capabilities can be dropped again. This way the process executes user supplied code (e.g. embedded interpreter runs a script) with the user and group identities configured for the actual website.

Serving the incoming requests is the task of the child processes of the web server. Every request triggers the *ap_process_request_internal()* function in the Apache web server, hence it is an adequate function where the *getmyretaddr()* syscall and its supplemental code can be injected. Just before invoking *getmyretaddr(),* it should be determined which identities should be used for serving the request. A new configuration directive called *ServeAsUIDGID* was defined, which can be used in the configuration of the web server to specify which identities should be set on a website. If the newly added directive was supplied in the configuration, then it is straightforward which identities should be used (e.g '*ServeAsUIDGID 1303 1011'* – the user id will be set to 1303, and the group id will be set to 1011). Otherwise (very useful when applying solutions to automatize the publishing of users' web pages, e.g. *mod_userdir* [21]) the same identities as the owner of the requested file will be set. If the ownership of the file cannot be determined, a HTTP error 404 (not found) is returned. And in case the owner of the file to be served is the system administrator (root – uid 0), the request is denied with

returning a HTTP error 403 (access forbidden).

It is possible that an incoming request requests a resource belonging to the same identity as the one which was already set for the previous request. For achieving better performance, it is first decided whether it is required to change identities or not, by comparing the identities needed to be set and the currently set identities. If these match, then the *getmyretaddr()* system call will not be invoked.

In the unfortunate case when an error occurs while dropping the now unwanted capabilities, the web server fails with an internal error returning a HTTP error 500 (internal server error) to the client, and the serving of the request will be terminated.

With these modifications the Apache web server became capable of using the presented mechanism. After compiling the modified source code, the method was tested and verified with the usage of the GNU Debugger (not covered in this paper).

The modifications in the source code of both the Linux kernel and the Apache web server in the form of a patch can be found at [22].

4. Performance analysis

According to the modifications, the Apache web server and the Linux kernel contain additional codes to be executed in every iteration when a request is served, which yields additional performance overhead. This overhead was measured empirically with the help of the *siege* HTTP testing utility [19]. The measurement tests were conducted both using the original and modified web server in conjunction with the modified kernel, and also using the previously mentioned ITK module in the Apache web server.

The siege utility simply sends requests to a given web server, and measures the elapsed time till the requests finish. The web server was configured with 16 separate websites with different user and group identities to be set for each site. Siege was configured to send requests to these 16 sites in a round-robin fashion. This results in an identity switch with every request, which means the worst possible case was simulated and benchmarked.

The same measurement was conducted several times with various concurrency level settings (1, 4 and 16 respectively) to minimize measurement incorrectness. An average of these results were evaluated. The results corresponding to the unmodified web server were taken as reference (100%). Accordingly, the results are shown in Figure 1.

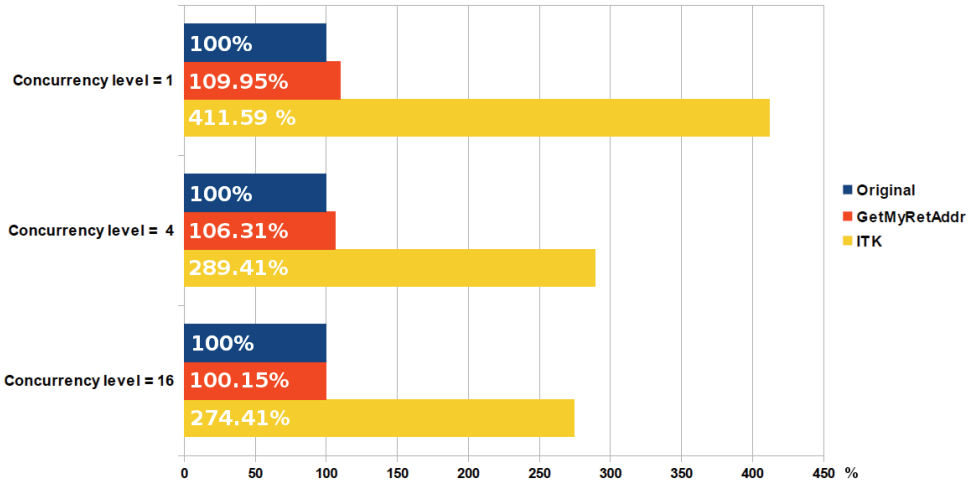

Figure 1. Performance overhead measurement results

It can be noticed that the overhead is very slight (+0.15%) when using concurrency level 16. On production servers many clients access many sites, hence concurrency level could be high. Also, keeping in mind that the tests were conducted simulating the worst possible scenario, and on real world web servers many request batches are for the same website, the average overhead should be lower. Therefore the performance overhead can be so low that the overhead can go unnoticed in real environments.

Conclusion

The mechanism introduced in this paper defines a novel approach for access control. The main idea of the mechanism is to check the calling memory address in the invoking process, and if this value is the same as it was during initialization, then various privilege elevations can be performed. The mechanism was implemented in the latest Linux kernel (version 3.1.10 as of writing) as a new system call. Integrating this new system call into the most commonly used web server nowadays - the Apache HTTPD web server - makes it possible to serve every HTTP request with different user and group identities, with very slight performance overhead. Hence the mechanism introduced in this paper is significantly faster compared to other existing solutions with the same security approach.

The separated identities allow some additional advantages for users and administrators of web servers. E.g. limiting and accounting resources on a per user

basis: OS level authentication for database access, controlled network access, file system access, quota management for uploaded files, etc.

Further development possibilities include porting to other operating systems (e.g. BSD), and also the new security risks possibly opened by the new system call (e.g. rewriting code on the trusted memory address) should be investigated closely.

Acknowledgements

This research was carried out as part of the TAMOP-4.2.1.B-10/2/KONV-2010-0001 project with support by the European Union, co-financed by the European Social Fund.

REFERENCES

[1] ACHOUR, M., BETZ, F., DOVGAL, A., ET. AL.: *PHP Manual*, PHP Documentation Group, 2012

[2] BUNCH, S.: *The Setuid Feature in Unix and Security,* Proceedings of the 10th National Computer Security Conference, pp. 245 – 253, 21-24 Sept. 1987

[3] CHEN, H., WAGNER, D., DEAN, D.: *Setuid Demystified,* Proceedings of the 11th USENIX Security Symposium, pp. 171 – 190, 2002

[4] FIELDING, R., GETTYS, J., MOGUL, J., FRYSTYK, H., MASINTER, L., LEACH, P., BERNERS-LEE, T.: *Hypertext Transfer Protocol – HTTP/1.1, Request for Comments #2616*, Internet Engineering Task Force, June 1999

[5] GARFINKEL, S.: *Web Security, Privacy & Commerce*, Second Edition, O'Reilly Media Inc., ISBN: 978-0596000455, Jan. 2002

[6] GUNDERSON, S. H.: *The Apache 2 ITK MPM*

HTTP://MPM-ITK.SESSE.NET (ACCESSED JAN. 2012)

[7] HARA, D., OZAKI, R., HYOUDOU, K., NAKAYAMA, Y.: *Design and Implementation of a Web Server For a Hosting Service*, Proceedings of the Ninth IASTED International Conference, Internet and Multimedia Systems and Applications, August 15-17, 2005, Honolulu, Hawaii, USA

[8] HARA, D., NAKAYAMA, Y.: *Secure and High-performance Web Server System for Shared Hosting Service*, Proceedings of the 12th International Conference on Parallel and Distributed Systems (ICPADS'06), Minneapolis, USA, 2006

[9] HEACOCK, S. G.: *Peruser MPM for Apache 2.x Technical Details*

HTTP://WWW.PERUSER.ORG/ (ACCESSED JAN. 2012)

[10] NIKIFORAKIS, N., JOOSEN, W., JOHNS, M.: *Abusing Locality in Shared Web Hosting,* Proceedings EUROSEC '11 Proceedings of the Fourth European Workshop on System Security, Salzburg, Austria, 2011

[11] POSTEL, J., REYNOLDS, J.: *File Transfer Protocol, Request for Comments #959*, Internet Engineering Task Force, October 1985

[12] ROBINSON, D., COAR, K.: *The Common Gateway Interface (CGI) Version 1.1, Request for Comments #3875.* Internet Engineering Task Force, Oct. 2004

[13] SHIFLETT, C.: *Essential PHP Security*, O'Reilly Media Inc., 2005, ISBN: 0-596-00656-X

[14] SNYDER, C., SOUTHWEL, M.: *Pro PHP Security*, Apress Media LLC, ISBN: 978-1590595084, Sept. 2005

[15] TOBOTRAS, B.: *Linux Capabilities FAQ*, 1999
HTTP://FTP.KERNEL.ORG/PUB/LINUX/LIBS/SECURITY/LINUX-PRIVS/KERNEL-2.4/CAPFAQ-0.2.TXT

[16] WEIGEL, E.: *Muxmpm/metuxmpm for Apache 2*
HTTP://WWW.SANNES.ORG/METUXMPM/ (ACCESSED JAN. 2012)

[17] IEEE STANDARD FOR INFORMATION TECHNOLOGY 1003.1-2004: Portable Operating System Interface (POSIX)

[18] IEEE STANDARD FOR INFORMATION TECHNOLOGY P1003.1E Draft: Portable Operating System Interface (POSIX)

[19] JOE DOG SOFTWARE: Siege HTTP Testing Utility
HTTP://WWW.JOEDOG.ORG/INDEX/SIEGE-HOME (ACCESSED JAN. 2012)

[20] NETCRAFT LTD. Web Server Survey January 2012:
HTTP://NEWS.NETCRAFT.COM/ARCHIVES/2012/01/03/JANUARY-2012-WEB-SERVER-SURVEY.HTML

[21] APACHE HTTP SERVER DOCUMENTATION
HTTP://HTTPD.APACHE.ORG/DOCS/ (ACCESSED JAN. 2012)

[22] THE SOURCE CODE MODIFICATIONS CAN BE ACCESSED AT:
HTTP://USERS.IIT.UNI-MISKOLC.HU/~VINCZED/GETMYRETADDR/