APACHE SECURITY

АБДУЛМОХСИН Х., БАТЫРКАНОВ Ж.И.

КГТУ им. И.Раззакова

izvestiya@ktu.aknet.kg

Обосновано использование семантико-фреймовой модели представления знаний для оптимизации поиска в реляционных базах данных.

Abstract

Restricting the access of a web server to system resources limits the potential damage caused to those resources through exploitation of web server vulnerabilities. However, allowing the web server to access the required resources enables the web server to provide expected functionality. This combination of denying unnecessary access and allowing required access results in providing web server functionality while limiting damage.

To demonstrate this, we hosted the Apache Security, an operating system that enforces a mandatory access control policy. By tailoring the Apache Security policy, we were able to control interaction between the Apache web server and other processes and files on the system.

The policy dictates that Apache is only allowed to display web pages and perform limited functions that support the display of web pages.

This work demonstrates the following.

∑ Apache Security is capable of supporting commonly used applications.

∑ Apache Security can confine applications so that a reduced level of risk is achieved when making applications available.

∑ Although confined, these applications provide the functionality expected by users.

Introduction. Commonly used applications, such as web servers, are often vulnerable to attack. To prevent attacks from being successful, known vulnerabilities can be eliminated by reducing application functionality or by implementing fixes or patches within the application source code. Reducing functionality is often not acceptable to users, and implementing fixes requires the cooperation of the vendor and is typically in reaction to damage that has already occurred.

An alternative is to reduce the level of risk by confining the application. Confining an application means to control the application's access to, and malicious damage to, system resources (e.g., processes and files).

ABBYY PDF Transformer 2.0
Click here to buy
www.ABBYY.com

[1] Linux is a registered trademark of Linus Torvalds. The LSM-based Security-Enhance Linux prototype is currently supported for kernel 2.4.17 and 2.5.2 with RedHat 7.1 or RedHat 7. 1. Red Hat is a registered trademark of Red Hat Software, Inc.

To adequately confine an application, the operating system that hosts the application must enforce a mandatory access control policy as specified by the system security administrator. An example of an operating system that provides mandatory access control features is Apache Security.

To demonstrate the feasibility of confining an application without reducing functionality, we hosted the Apache HTTP Server[2] on Apache Security and used features provided by Security-Enhanced Linux to confine Apache. This paper describes

1. potential damage caused as a result of exploitation of a web server,
2. Apache Security features,
3. how these features were used to confine the Apache web server, and
4. how potential damage resulting from exploitation of the Apache web server is reduced while still allowing Apache functionality.

Web Server Security Concerns

Sharing information and conducting business via the World Wide Web has become a critical requirement for most organizations. However, a web server that allows an organization to share information and conduct business could potentially be exploited to cause unauthorized modification or destruction of that information and other system resources.

Through various attacks, such as buffer overflow attacks, a malicious user could gain control of a web server process. Since web servers often run with enhanced privileges, the user who gains control of the web server process possesses enhanced privileges that can be used to cause damage to the system.

Even if a malicious user cannot gain control of the web server process, scripts potentially allow users to direct the web server to perform a malicious action. A Common Gateway Interface[3] (CGI) script accepts user input and submits it to the server for processing. For example, electronic purchasing forms and web site guest books are typically implemented through CGI scripts. Unfortunately, it is possible for a malicious user to enter executable code as input into a form or guest book. If the server executes that code, the server could cause damage to the system.

Another type of script is a Server Side Include (SSI). An SSI is a file that can be parsed by the web server to supply dynamic information for a web page, such as the current time and date. Executable shell commands or an interface to CGI scripts can be included in an SSI. For example, an SSI could include a statement such as <!--#exec cgi="runme.cgi"-->. The web server would execute runme.cgi when it parsed the SSI. If runme.cgi contained malicious code, the web server could cause damage when running the code.

Approaches to Reducing Risk. There are several approaches that can be taken to reduce the risk associated with a web server.

One of the easiest methods of reducing risk is to run the server as "nobody" [5]. This can either occur when the server is launched or whenever the server forks a process to handle a connection on port 80. However, once the server starts running as "nobody", the system administrator has to ensure that the server still has access to files it needs access to by setting permissions appropriately. This may result in granting wider access to certain files than is desired. This approach also doesn't prevent access to world-readable/writeable/executable directories and files, of which there are many on a typical system. If any of these executables happens to be setuid, it may be possible to obtain root privileges indirectly.

Another method of reducing risk is to tighten the configuration of the web server and either restrict or turn off functionality[6]. For instance, the web server could be configured to deny the use of SSI's or user-developed CGI scripts. This eliminates vulnerabilities but also eliminates functionality. It also requires the system administrator to have knowledge of web server configuration details.

[2] Developed by the Apache Software Foundation (http://www.apache.org)

[3] Specification at http://hoohoo.ncsa.uiuc.edu/cgi/

A third approach is to restrict the files and processes the web server has access to. The web server cannot damage files and processes that are inaccessible. This can be done using the *chroot()* system call. *chroot()* changes the root of the file system as it appears to the process. Any search for a file will start at this new root as if it were “/”. This will cause, with appropriate choices being made as to directory structure, any other user files to disappear from the viewpoint of the process. For the same reason though, the file structure containing the various system binaries will also become inaccessible. This means that all of the manifold utilities and libraries that web servers need must be duplicated in the newly rooted file structure.

Creation of a *chroot()* ‘jail’ does not prevent root exploit attacks and barely slows down the malefactors when they succeed. All it takes is to execute *chroot()* with the appropriate path. Even if the web server is not running with root privileges, its enhanced privileges may be enough for the attacker to do some damage when it is penetrated.

Another approach addresses the dangers associated with CGI scripts. Wrappers, such as suEXEC [7], cgi-wrap [8], and sbox [9], are called by the server to execute user scripts rather than executing them directly. These wrappers then perform functions such as checking various system and file parameters, ensuring that only approved commands are called from the scripts, enforcing resource-usage limitations, changing the uid of the process to match the uid of the script, and calling *chroot()*. However, the wrapper approach only addresses the CGI script concerns, and not other web server concerns.

The solutions above represent a reasonable attempt to deal with the problems of web-hosting on a Linux platform.

However, they all suffer from administrative overhead and provide no defense against ‘root exploit’ attacks that lead to unwanted access. None of them deal with vulnerabilities in the base server or in other services running on that server. They also focus on controlling access of a process to a file, but do not address access of one process to another.

Apache Security. To more effectively address a wider range of server concerns, the operating system that hosts the web server must enforce a mandatory access control policy as specified by the system security administrator. One such operating system is Apache Security.

A general security policy configuration [10] is included with Apache Security. This general policy contains Type Enforcement[4] and Role-Based Access Control (RBAC) components.

With Type Enforcement, types are associated with processes and files, and the policy defines allowed interaction between types[5]. For example, the policy could state that a process of type *y_t* is allowed to write to a file of type *x_t*.

A security configuration uses Role Based Access Control by defining a set of roles, and associating a list of types with each role[6]. A process executing with a particular role must always be executing with one of the associated types; the security server will not permit it to transition to any other type.

Three roles are defined in the base security configuration.

∑ *system_r* is assigned to the user identity for system-owned processes and files (*system_u*),

∑ *sysadm_r* is assigned to system administrators,

∑ *user_r* is assigned to ordinary users

Each user process starts with an initial role assigned to that user, although processes may change roles.

Roles and types are associated with a process or file through a security context[7].

4 Type Enforcement is a registered trademark of Secure Computing Corporation.

5 This differs from traditional Type Enforcement where domains are associated with processes and types are associated with objects. Permissions are defined for both pairs of domains and for domain-type pairs.

6 This differs from traditional RBAC where permissions are associated with roles.

7 Since files do not transition between types as processes do, the role associated with a file has little function. Therefore, a default role of *object_r* is used for files.

The security context labels the process or file with a user identifier, a role, and a type. For example, when a user, John Smith, first logs in, the security context of his shell is jsmith: user_r:user_t. The security context of a process or file can change, or transition, as required and as allowed by the policy.

As previously stated, a general security policy configuration is included with Security-Enhanced Linux. This policy is intended as a starting point for system administrators to customize a policy to fit the security requirements of their system.

The Apache Securityarchitecture and implementation simplify policy changes by separating policy and enforcement functions. As shown in figure 1, an Object Manager receives requests for objects.

ABBYY PDF Transformer 2.0 Click here to buy www.ABBYY.com

The Object Manager queries a Security Server to see if the policy permits the requested action. The Security Server reads the current system policy and determines if the action is allowed or not. The Security Server sends its decision to the Object Manager, and the Object Manager enforces the decision. By separating the policy and enforcement, a change in policy does not require a modification to the enforcement mechanisms.

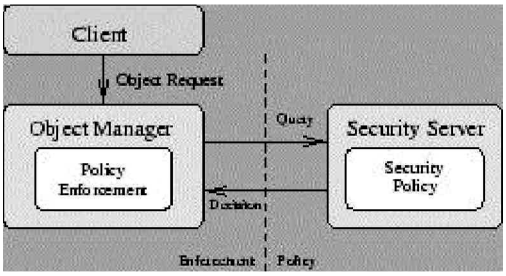


Figure 1.

Because of the flexibility of Security-Enhanced Linux, it is possible for Security-Enhanced Linux to both support and confine commonly used applications through modifications to the policy.

Confining Apache. Apache is a full-featured, open source web server that is packaged with RedHat Linux. Apache's primary role is to display web pages to users requesting the web pages. To properly display these web pages, Apache handles many of popular web technologies such as CGI scripts and SSIs.

The high-level policy we stated for Apache is:

∑ The Apache server is allowed to

- accept user requests for web pages,
- read web pages,
- execute scripts,
- check password protection on web pages and scripts, and
- display web pages back to the user.

∑ The system boot process is allowed

to start the Apache server. ∑ The web administrator is allowed to

- create and modify system web pages,
- modify and execute system scripts,
- specify password protection on system web pages and scripts, and
- specify which files can be accessed by system scripts

∑ Users are allowed to

- send requests for web pages to the Apache server,
- modify user web pages,

- modify and execute user scripts,
- specify password protection on their web pages and scripts, and
- specify which files can be accessed by user scripts.

∑ Script processes are allowed to

- execute script interpreters and libraries
- read, write, and append specially marked files.

For Apache to provide its functionality, we determined that Apache requires access to various files and modified the high-level policy to allow the Apache server to do the following:

∑ send and receive messages to and from the network

∑ bind to port 80

∑ read web configuration files located in /etc/httpd/conf

∑ read and append to web log files

located in /var/log/httpd

∑ execute system libraries and Apache-specific libraries

∑ call suEXEC prior to executing user scripts if Apache is configured to do so. To support this high-level policy, we defined a role for the web administrator called *httpd_adm_r*. We also defined new types required to control Apache processes and files. Apache processes and files and their assigned types and roles are listed in After defining these types, we used the Apache Security policy language to specify a "formal" policy that implemented the high-level policy.

This "formal" policy is described below and is depicted in figure

Summary. This work demonstrates how the Apache web server can be confined to limit the potential damage caused if vulnerabilities associated with the Apache web server are exploited.

In addition, we have also shown that Apache Security can support commonly used applications.

Therefore, users do not have to forgo their preferred applications to take advantage of the security features provided by Security-Enhanced Linux.

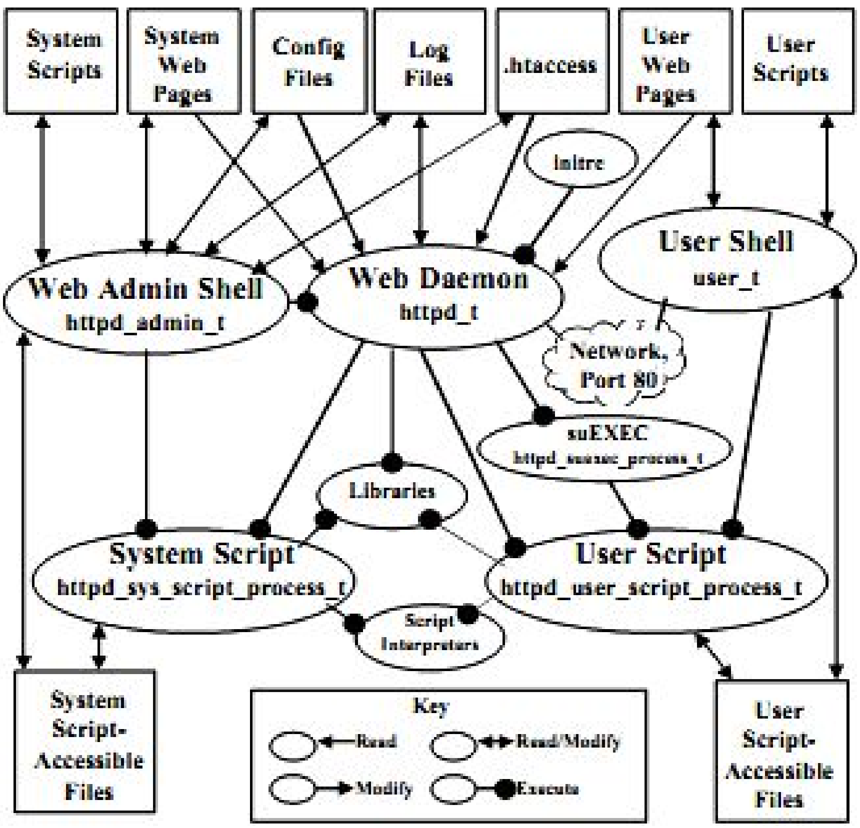
Because Apache Security separates enforcement from policy, the system administrator can tailor the policy to confine site-required applications, as was done with Apache. Tailoring the policy to confine the applications results in a reduced level of risk when making these applications available.

Because of the flexibility of the Apache Security policy, the policy is able to confine an application without reducing the application's functionality expected by the users.

Process or File	Type	Role
Apache daemon (server process)	httpdt	system
System web pages (.html or .htm files)	httpd_sys_content_t	object_
User web pages (.html or .htm files)	httpd_user_content_t	object_
System script file	httpd_sys_script_t	object_

User script file	httpd_user_script_t	object_
Files that provide web password protection on system directories	httpd_sys_htaccess_t	objectr
Files that provide web password protection on user directories	httpduserhtaccesst	objectr
Apache configuration files located in /etc/httpd/conf	httpd_config_t	object_ r
Apache log files located in /var/log/httpd	httpdlogfilest	objectr
Libraries included with Apache	httpdmodulest	objectr
Apache executable file	httpdexect	objectr
Web administrator shell process	httpd admint	httpdad
System script process	httpd_sys_script_proce	system
User script process	httpd_user	user r
Files that can be read by system scripts	httpd_sys_script_r_t	objectr
Files that can be read and written by system scripts	httpd_sys_script_rw_t	objectr
Files that can be appended by system scripts	httpd_sys_script_a_t	objectr
Files that can be read by user scripts	httpduserscriptrt	objectr
Files that can be read and written to by user	httpduserscriptrwj	objectr
Files that can be appended to by user scripts	httpduserscriptaj	objectr
suEXEC executable	httpdsuexect	objectr
suEXEC process	httpd_suexec_process_t	system

FIGURE 2



References

1- The World Wide Web Security FAQ http://www-genome.wi.mit.edu/WWW/f aqs/www- security-faq.html#contents

2- Apache suEXEC Support, http://httpd.apache.org/docs/ suexec.html

3- sbox,http://stein.cshl.org/W WW/software/sbox/

4- P. Loscocco and S. Smalley. Integrating Flexible Support for Security Policies into the Linux Operating System. Technical report, NSA and NAI Labs, February 2001.

5- Computer Incident Advisory Capability web site

http://ciac.llnl.gov//ciac/documents/ciac2 308.html#4