Architecture of Rule Set Based Access Control (RSBAC)

Security Architecture

Integrating multiple security models simultaneously in the kernel and detailed logs of any access: The free Rule Set Based Access Control (RSBAC) security System offers customized protection for a wide range of requirements. BY AMON OTT

Ronald Raeffle, visipix.com

Linux security holes typically occur in Server programs and S bit tools. The best approach would be to avoid mistakes and update programs immediately if a bug occurs. This is not always possible, the next best thing is to restrict potential damage, and this is where access control systems such as RSBAC come into play [1].

If an attacker exploits the security holes in servers or s bit tools, access to the system should be restricted to a minimum. In this case, even a successful compromise will cause only limited damage, and protective mechanisms can be implemented directly in the operating system kernel.

The standard Linux kernel prevents access to various resources such as files, directories or system configurations, but unfortunately the standard mechanisms are fraught with weaknesses:

- Poor granularity
- Discretionary access control
- An all-powerful root user

Linux access controls only offer the standard privileges read, write and execute; additionally they only allow distinct privileges to be defined for the owner of a file, the members of a group and all others. Restrictions typically do not apply to the root user. The granularity of these privileges is thus insufficient for many tasks.

Linux Privileges – not enough

The owner of a file can do what she pleases with that file, this is commonly referred to as DAC, or discretionary access control. If an attacker has compromised a process, the attacker's activities assume the privileges of the account used to run that process. Thus, an attacker, if fortunate, can manipulate any files belonging to the compromised account with the gained access privileges.

The all-powerful system administrator, root, is the most dangerous of the issues mentioned so far. Many activities are restricted to the root user, from administrative tasks to simple actions. Thus most service are originally launched with root privileges and have superuser access to the whole system, without ever actually needing these extensive privileges.

What is worse is the fact that many services need to be run with root privileges (or to be able to assume root privileges at any time) to allow them to change to any user account. The POSIX capabilities introduced to the Linux kernel a while back, allow a program launched by root to drop some privileges, but this is left up to the program itself.

Architectural Requirements

The main aim for the developers of RSBAC was to produce a flexible and effective access control system as an add-on for existing Linux mechanisms. To achieve this goal, the system must fulfill a number of requirements:

It must provide the underlying platform to allow the developer to program access control models quickly and simply. This permits a clear distinction between components that make decisions, and components that enforce them.

The enforcement components act independently of the components that

THE AUTHOR

Amon Ott is a self-employed computer scientist and the author of the RSBAC system. His mainstay is bespoke development and Linux firewalls, preferably with RSBAC. He is also working on his doctorate, which he hopes to complete shortly.

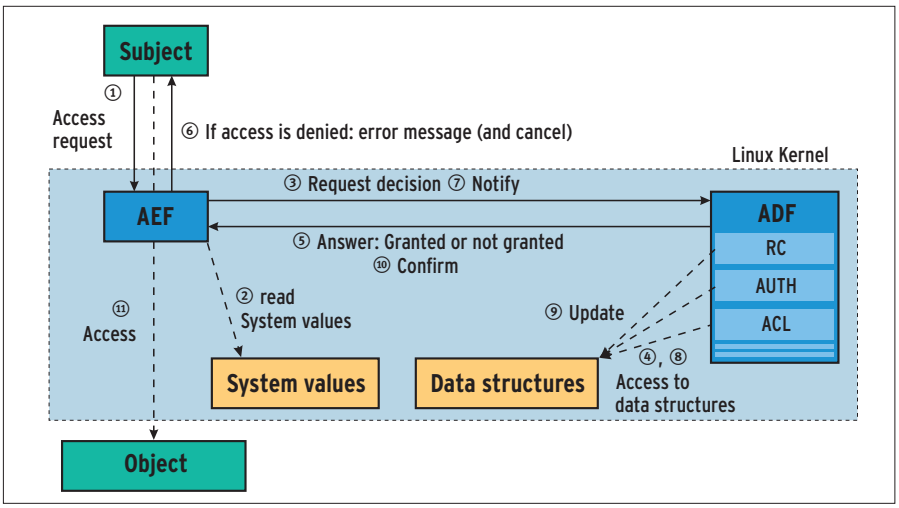

Figure 1: A subject's access to an object is monitored by the Access Enforcement Facility (AEF). The Access Decision Facility (ADF) decides whether to permit or deny access

make decisions. New decision models can use the underlying infrastructure.

A large number of tried and trusted security models exist for various tasks, and combinations of these models sometimes make sense, depending on the situation. The underlying framework should thus support multiple security models simultaneously and independently, allowing the administrator to choose the most suitable model for the current assignment. No matter what model is in use, activities and any decisions taken need to be logged, and the logs must be protected from attempted manipulation.

The original RSBAC system designed fulfills nearly all these criteria. Over the course of the last five years the range of functions and monitored objects has increased dramatically, allowing RSBAC to monitor networks. The main elements of the original have been tested during this time and the developers see no reason to revise them.

Inner Values

We need to explain a few terms, in order to describe the internal architecture, so bear with us. From the access control perspective a subject attempts to invoke a specific type of access to an object. On a Linux system, the following occurs: a process (the subject) attempts to read (access type) a file (object).

The various object types are categorized by target type on an RSBAC system (see Table 1 for an overview). RSBAC also distinguishes a large number of access types (request types) that are applied to the object types.

Table 2 lists a selection of access types, some of which are used in our practical example later. The entire list is available in the documentation from [1]. The basic building blocks of the RSBAC systems are shown in Figure 1. The enforcement component, or Access Control Enforcement Facility, AEF, mainly comprises enhancements of existing system functions.

These enhancements require the decision making element, or Access Control Decision Facility, ADF, to reach a decision before any access, and so possible compromise is permitted.

If the ADF refuses access, the AEF will return an "access denied" error to the subject. The decision facility and the data structures used are mostly independent of the kernel version. Only AEF requires one or two changes to existing kernel functions. This component was produced by enhancing existing syscalls.

Components co-operating

Access control involves a number of steps. The subject (the process) calls a system function to request access to an object (1). An extension of this function (the AEF) reads some system values, such as the process ID, the type, and ID of the target object (2), before calling the decision facility, ADF; and handing on the information it has collected and the type of access (3). The request is originally addressed to the central decision facility of the ADF. This function requests individual decisions from all active decision modules. The modules read attributes from data structures (4) and reach a decision: permitted, not defined, or denied.

The central function collates the individual decisions, and returns a collective decision (5). The ADF is restrictive in this respect; if a single module returns a negative reply, the ADF will deny access. Actions are only permitted if all the modules agree that they should be permitted.

In the case of a negative decision, the system call is halted and returns an access error to the process (6). In the case of positive decisions, the AEF forks to the system call itself. If the call is successful, the AEF sends a message to this effect to the ADF (7).

The central messaging function of the ADF is responsible for passing the message to the appropriate module functions. The module functions retrieve the current attributes from the data structures (8), update them (9) and confirm that the call has been completed correctly (10).

If a new object was created by the system call, the message from the AEF to the ADF will contain the type and ID for the new object. The decision modules then create the attributes of the object. After confirming, the system function passes the requested

Table 1: Target Types

Name	Description
FILE	Also includes special device and UNIX network files if they are handled as files
DIR	Directory
FIFO	Pipe with name entry in file system
SYMLINK	Symbolic link
IPC	Inter Process Communication object on System V basis
SCD	System Control Data – global system settings and objects such as host names or time
USER	User object mainly serves the purpose of managing attribute assignments
PROCESS	Process object for receiving signals or reading process statuses
NETDEV	Network device
NETTEMP	Network template
NETOBJ	Network object – normally sockets

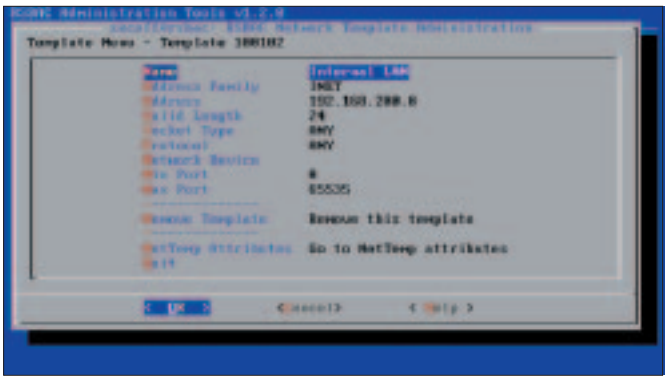

Figure 2: Admins can use network templates to assign access privileges to network address and port ranges. In our example, the ports on all the hosts in the IP network 192.168.200.0/24 have been selected

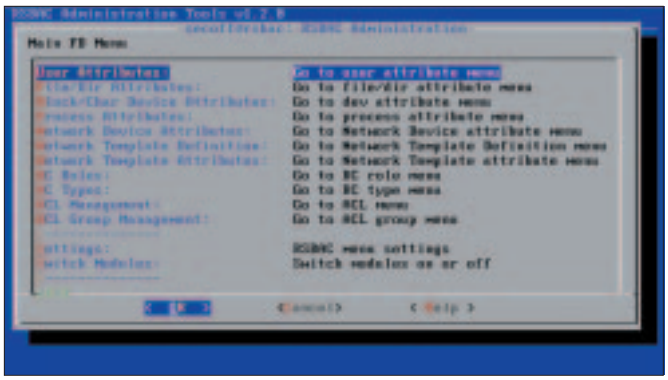

Figure 3: The main administration menu provides the RSBAC user with a straightforward configuration interface, with simple control over all of the Decision Module functions

data (11) and control back to the invoking process.

A Practical Example

A practical example is useful to our understanding of the theoretical path. When a process wants to open a file for read and write access, it uses "sys_open()" with appropriate parameters. The parameters might specify that "sys_open()" should create the file if it does not already exist, or possibly truncates the file to zero length if the file exists.

If the ADF rejects one of the following decisive questions, the system call terminates and issues an "access denied" message. The first thing "sys_open()" needs to do, is to resolve the filename, to discover the inode.

An auxiliary function, whose RSBAC extension sends a "SEARCH" request to the ADF for every folder touched, takes care of this. If the file does not exist, the extension of the open function generates a "CREATE" request for the target directory, creates the file and informs the decision facility of the new object. Otherwise a "TRUNCATE" request is issued for the file, the open function truncates the file to zero, and reports the success of the operation.

After this preparatory work, the syscall generates a "READ_WRITE_OPEN" request and opens the file. The ADF learns that the file has been opened and updates the file's attributes. This provides the process with a descriptor so that the process can go on running.

Data Storage Structures

As already mentioned, so-called attributes, which are assigned to every user, process, and object are the basis for each access decision. Attribute management is the task of the general data storage facilities. Additional model specific data, such as groups or access matrixes that cannot be organized within generic structures, also exist. Model specific structures provide storage facilities in this case.

The data storage component takes care of the thankless task of list management, thus reducing the load on the data storage components; this involves disk storage, SMP locking (for multi-processor systems), and similar tasks. It stores the majority of this data in a generic list system that allows any number of generic one or two-tiered list systems (lists of sublists), with indices and data fields of any size to be easily registered.

The decision facilities register their lists on RSBAC initialization or when binding a file system. Only a few of the lists are implemented differently due to specific conditions.

Persistent Data

If necessary generic lists can provide persistent data storage, that is the data stored in the lists will survive a reboot or deregistration. To achieve this, the

Table 2: Request Types

Name	Object Types	Description
BIND	NETDEV, NETOBJ	Bind network addresses
CLOSE	FILE, DIR, FIFO, DEV, IPC, NETOBJ	Close a file descriptor
CONNECT	NETOBJ	Open connection to remote node
CREATE	DIR (where), IPC, NETTEMP, NETOBJ	Create object
DELETE	FILE, DIR, FIFO, IPC, NETTEMP	Delete object
EXECUTE	FILE	Execute file
NET_SHUTDOWN	NETOBJ	Close connection channel
READ	DIR, SYMLINK, IPC, NETTEMP (optionally FILE, FIFO, DEV, NETOBJ)	Read from object
READ_WRITE_OPEN	FILE, FIFO, DEV, IPC	Open for reading and writing
RECEIVE	NETOBJ	Receive data from remote node
SEARCH	DIR, SYMLINK	Name resolution
SEND	NETOBJ	Send data to remote node
TRUNCATE	FILE	Change length of file

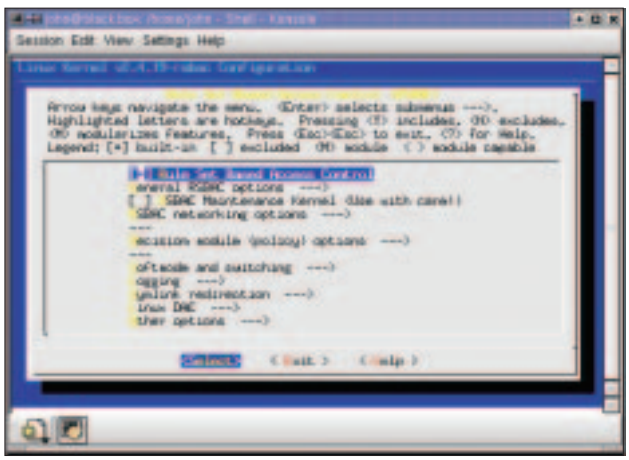

Figure 4: RSBAC needs to be enabled in the Linux kernel. A configuration menu is available for basic settings

"rsbacd" kernel daemon periodically saves any lists tagged as changed in special protected directories on the hard disk, where they are read by the data storage facility on re-registering the list. A registration parameter specifies what partition these files can be stored on to allow targeted binding of any of the file system objects.

Modules can optionally supply a default value when registering a list. If a list element goes missing at a later date, the data storage facility will also supply this value. For optimization purposes, any persistent elements containing default values are deleted.

If a value changes, the data storage facility reinstates the element. In the case of two-tiered lists, sublists are generated or deleted as required. This procedure keeps the length of the lists to a minimum and thus reduces the access times needed.

Every list element is assigned a time limit when it is created or updated, and is removed once this period expires. This characteristic is used by some decision facilities to generate temporary entries or privileges. Persistent values are marked with a value of "0". Generic lists are implemented as double linked, sorted lists that allow you to register both descriptive and comparative functions for optimized access.

If a function of this type has not been registered, simple "memcmp()" based memory comparison is used. [2] provides a more detailed description of the list management interfaces and parameters.

Rule Templates

Network connections are fairly ephemeral in most cases; data packets are often transmitted individually and independently. That makes it particularly difficult to assign attributes to them, as administrative overheads would be punitive. RSBAC provides network templates for this task. They describe multiple network end nodes based on various criteria, such as the protocol family, connection type, network protocol or port number. Figure 2 shows an example of how they are defined.

RSBAC will not store the attributes separately for each network end node, or for each connection, but collectively in a template. The end nodes of the network (that is the source or target of data transmission) inherit their values from the most suitable template (that is the template with the lowest descriptor). This allows the ADF to reach a decision for "CONNECT" type access by reference to the template attributes of the source or target address by simply looking up the template.

Administration

Templates allow you to specify that a specific user should only be allowed access to the local network via the Internet Protocol TCP, or that a browser can only access the HTTP proxy port on your firewall. There is no need to configure each individual connection.

As RSBAC stores all of these settings in the kernel or in protected files, administrative tasks mean initiating system calls or accessing the "/proc" file system. This allows the kernel to

designate users who are permitted to change specific settings. And this is RSBAC's solution to the major issue of the all-powerful root user: If the configuration files were stored in normal files, users with write access to these files would then automatically have administrative privileges. RSBAC allows multiple administrators to have different privileges.

Self-Control

With only a few exceptions, each decision module is responsible for its own attributes. Models with scientific backgrounds, such as RC and ACL (see insert **"Decision Modules in RSBAC"**) in particular, support the delegation of administrative tasks to multiple users. Root still has special rights in the default configuration of most modules, but is a normal user like everyone else, apart from that.

A support module called "AUTH" was introduced to help out with user ID management, which is a critical issue. "AUTH" allows you to define the user IDs that specific programs and processes can assume. A process can only assume an ID that "AUTH" allows it to assume, any others are prohibited.

A number of RSBAC administration tools are available. They facilitate many administrative tasks and provide user interfaces for the RSBAC system calls. Menus provide for easier use – see Figure 3 for an example of the main "rsbac_menu" menu. RSBAC is probably the oldest and – judged by its codebase most extensive – free access control system for the Linux kernel. Its clear and modular structure ensure that the authors could keep track on development activities. RSBAC has become quite popular in Europe where the system is in widespread use. Conservative estimates suggest that RSBAC is in use on several hundred production systems. ■

Decision Modules in RSBAC

The current, stable RSBAC version 1.2.1 comprises the following decision modules and rules, some of which are used to implement more complex security models.

MAC – Mandatory Access Control, Bell-La Padula.

FC – Functional Control: This simple role model allows access to security information for security officers only and allows only administrators to access system information.

SIM – Security Information Modification: Only security officers are allowed to modify data tagged as security information.

PM – Privacy Model: A data protection model devised by Simone Fischer-Hübner to implement European data protection guidelines.

MS – Malware Scan: Checks files for malevolent software during read and execute access. Version 1.2.1 contains only a scanner prototype, the pre-release version 1.2.2-pre1 uses a professional virus protection software by F-Prot. Support for additional scanners is planned.

FF – File Flags: Global attributes apply to files and directories, "execute_only", "no_execute", "read_only", "append_only", for example.

RC – Role Compatibility: This powerful role model was designed specifically with Linux servers in mind. It defines roles for users and programs, and types for all kinds of objects. Access privileges for each type can be specified for every role. The model also allows a schema for a strict delegation of administrative tasks to multiple roles, and defines time limits for access and administrative privileges.

AUTH – Authentication Enforcement: This module governs "CHANGE_OWNER" requests for processes and thus any "setuid()" calls. Processes and programs can only access user IDs specifically allowed to them.

ACL – Access Control Lists: An access control list is assigned for each object, to define the permissible access types for various subjects. Subjects are defined as user IDs, RC roles, and ACL groups. If an object does not have an entry for a specific subject, it will inherit the rights assigned to a superordinate object, for example a directory. An inherited rights mask is available to filter inheritance, allowing any rights assigned to be filtered out for all subjects. The ACL model also defines superordinate default ACLs, individual group management for every user, and time limits for any rights and group memberships assigned.

CAP – Linux Capabilities: Allows you to assign minimum and maximum Linux capabilities (delegated root privileges) to any user and program. Thus server programs can run as normal user accounts, or root programs can be executed with restricted privileges.

JAIL – Process Jails: This module introduces a new system call "rsbac_jail()", which is fundamentally an extension of the FreeBSD jail. Programs launched within the jail are captured in a chroot environment with restricted administrative and network privileges.

Installation

Before you can install RSBAC, you first need to download the sources from the home page. They comprise three parts: a tar archive contains modules that are independent of the kernel version. A version dependent kernel patch is additionally required. There is also a tar archive with administration tools. The RSBAC patch mainly comprises the initialization calls and adds system calls for AEF tasks. As an alternative, you can also download pre-compiled kernel sources as a bzip2 tar archive.

The kernels supplied by most distributions have mostly been through wide ranging modifications, and this often leads to issues. In this case, you may have to resort to the original kernel, available from *ftp://ftp.kernel.org/pub/linux/kernels* or a mirror site.

After expanding the tar archive in the main directory for your kernel sources, and applying the patches, follow normal procedure to configure, compile and install the kernel. The additional "Rule Set Based Access Control" menu shown in Figure 4 comprises a number of submenus with a wide range of options, with help texts for each option. Default values are OK for most applications.

When you reboot, the "rsbac_auth_enable_login" kernel parameter allows the login program to switch to any user ID in order to permit users to log on. The "rsbac_softmode" parameter is useful for initial tests, as it merely logs decisions without enforcing them.

After successfully launching the system, you can go on to expand the support tools and follow the usual steps, "./configure && make && make install" to compile and install them. If the RSBAC kernel sources are not available in "/usr/src/linux", you might like to try the configure parameter "--with-kerneldir".

INFO

[1] RSBAC home page: *http://www.rsbac.org*

[2] Interface to the generic list system: *http://www.rsbac.org/lists.htm*