Rule Set Based Access Control (RSBAC)

Linux Kernel Security Extension

Tutorial

Amon Ott <ao@rsbac.org>

Contents:

1 Motivation: Why We Need Better Security in the Linux Kernel
2 Overview of RSBAC
3 How to Identify Security Requirements on a Server
   3.1 System Base
   3.2 Services
   3.3 Users, User IDs and Paths
   3.4 Logging

Contents II:

4 Selecting a Security Model Combination
   4.1 General Criteria
   4.2 Model Specifics
   4.3 Experiences

5 Breaking the Requirements into Model Specific Designs
   5.1 Base Protection and Service Encapsulation
   5.2 AUTH
   5.3 FF
   5.4 RC
   5.5 ACL
   5.6 Logging

Contents III:

6 Hands-On Part
   6.1 Select Simple Server Type
   6.2 Specify Requirements
   6.3 Select Models
   6.4 Design a Configuration
   6.5 Implement It

7 Ending It Up
   7.1 Conclusion: What We Learned
   7.2 How to Go On
   7.3 Open End with Questions
1 Motivation

- Classic Linux/Unix Access Control is insecure
  - Small Granularity
    - Read, write and execute for owner(?), group and others is not enough
  - Discrete Control
    - Trust in users
    - Who is ‘owner’ of data?
    - Malware: Invitation to Trojans and Viruses
  - Superuser root
    - Full Access
    - Too often needed (Bind low ports etc.)
    - Too many exploits (root kits, kernel module attacks etc.)

- Better models for other administration goals
- Flexible Model selection and combination

2 Overview of RSBAC
(External Presentation)

3 How to Identify Security Requirements on a Server

3.1 System Base
3.2 Services
3.3 Users, User IDs and Paths
3.4 Logging

3.1 Requirements: System Base

- Filesystem Structure
  - Modification often leads to denial of service
  - Find crucial elements, e.g. /bin, /etc, /boot, /var

- Executables
  - Liable to virus or trojan infection, possible denial of service
  - Identify all (dirs with) executables in the system to be protected
    - /bin, /usr/bin, /sbin, /usr/sbin, several dirs under /usr/lib, ...
  - Specify, what files should *not* be executed
    - What is not protected should never be executed, so best chose ‘everything else’

- Libraries
  - Same as executables, but different access patterns
  - Files ‘.so’, some subdirs, e.g. /usr/lib/apache
### 3.1 Requirements: System Base II

- **Configuration Files**
  - Modification can lead to illegal accesses or denial of service
  - Identify all crucial (dirs with) configuration files

- **Kernel Objects**
  - Kernel Images
  - Kernel Module Files
  - System.map
  - Raw Memory
    - Should never be accessed

- **Devices**
  - Raw access can bypass access control and lead to almost any problem
  - Identify all devices, which can be used to compromise the system
    (/dev/hda, /dev/mem, ...)

### 3.1 Requirements: System Base III

- **Authentication data**
  - Crucial for security
  - Identify programs which may read or even modify for all users
  - e.g. /bin/login, /usr/bin/passwd, /usr/sbin/user{add|mod|del}
  - Optional: 'Account Manager' user who may read or even modify

- **Other Objects**
  - boot files
  - ioports / direct hardware access (X server etc.)
  - log files
  - ...

### 3.2 Requirements: Services

- Protection of and against all services
  - Local services maintain functionality
    - Identify all local services you need (and turn all others off)

- Network services make servers, but are their main vulnerability
  - Identify all network services you need (and turn all others off)

- Identify objects and access patterns for each service
  - Don't worry: a rough approximation gives a good start

### 3.3 Requirements: Users, User IDs and Paths

- Identify all user types of the system
  - Local and remote users
  - What services do they use?

- Find all user IDs needed by each service
  - Service users and running IDs (wwwrun etc.)
  - Ranges of IDs usable

- Identify the user ID paths
  - User login paths (who logs in through which service)
  - Chains of IDs used by services
3.4 Requirements: Logging

- Detect attacks
- Provide user accountability (who did what)
- Provide a modification history etc.
- Identify the users, programs, objects and accesses you would like to know about

4 Selecting a Security Model Combination

4.1 General Criteria
4.2 Model Specifics
4.3 Experiences

4.1 Model Selection: General Criteria

- Only consider models you really understand
- Think how each model could meet your requirements *before* choosing
  ○ Identify feedback from requirement break down to models
- Keep it simple:
  ○ Choose only those models that really give you a benefit
  ○ Do not choose subset models with superset models - you will get confused
- Develop a personal order in which to apply each model from easiest to most difficult

4.2 Model Selection: Model Specifics

- AUTHorization
  ○ Use for all user ID related things, e.g. to restrict login paths
  ○ Quite simple
  ○ Essential

- File Flags (FF)
  ○ Use for filesystem object protection which is common for all users
  ○ Pretty simple
  ○ Recommended for directory structure protection

- Role Compatibility (RC)
  ○ Use for all users and objects, which can be generalized into roles and types
  ○ Use for program based administration
  ○ Medium level
  ○ Strongly recommended because of role/type generalization
4.2 Model Selection: Model Specs II

- Access Control Lists (ACL)
  - Use whenever you need rights for individual users or objects
  - Use, if you also need discretionary control or individual user groups
  - Medium level, but difficult to keep setup overview
  - Recommended for uses named above

- Other Models: MAC, FC, SIM, PM, MS
  - Only use for specific needs
  - In most cases not recommended
  - Not treated here

4.3 Model Selection: Experiences

- Typical Combination: AUTH and RC, with a bit of FF

- ACL mostly unused

5 Breaking the Requirements into Model Specific Designs

5.1 Base Protection and Service Encapsulation

5.2 AUTH

5.3 FF

5.4 RC

5.5 ACL

5.6 Logging

5.1 Base Protection and Service Encapsulation

- Base Protection: Service independent protection of the system base
  - Protect identified system base (see 3.1: Base requirements)
  - Infrastructure and ‘fallback’ for service encapsulation
  - Strongly recommended

- Service Encapsulation: ’Sandbox’ around each individual service
  - Minimum access rights
  - For remote access and root account services strongly recommended
  - Other services optional

- No strict separation
  - Service encapsulation uses Base Protection infrastructure
### 5.2 Requirements to AUTH: User ID paths

- Define setuid capabilities for all programs
- Follows directly from 3.3: User ID requirements

### 5.3 Requirements to FF: Base protection only

- Filesystem infrastructure
  - Set `no_rename_or_delete` on all important dirs and files (not inherited), e.g. `/etc`, `/bin`, `/usr/bin`, `/boot`, ...
- Protect executables, libraries, configuration files, kernel objects and boot files
  - Set flags `search_only` (only applied on dirs) and `read_only`
  - Optional: set `execute_only` on binary executables (scripts need `READ_OPEN` etc.)
- Protect against execution of uncontrolled files
  - Unset flag `add_inherited` on all objects named above
  - Set flag `no_execute` on `/` (or e.g. `/home` only)

### 5.4 Requirements to RC

- Protect executables, libraries, configuration files, kernel objects, boot files and `/tmp` dirs
  - Define one RC file/dir type for each group
  - Remove unnecessary rights to these types from all defined roles
  - Optional: Define new role 'Configuration'
    - Only role with write access to configuration files
    - Assign to config user or make System Admin role compatible with it
  - Optional: Define new role 'Module Loader'
    - Only role allowed to load modules
    - Can only read libraries and type 'Modules'
    - Set as initial role for `insmod` etc.
    - Set types for the protected objects

- Protect against execution of uncontrolled files
  - Remove EXECUTE right to all types except executables and libraries

### 5.4 Requirements to RC II

- Protect devices
  - Define RC device types, e.g. 'Raw Disk'
  - Define RC roles for specific tasks, e.g. 'Raw Disk Access' for `fsck`
  - Remove unnecessary rights to these types from all defined roles
  - Assign specific task roles to programs
  - Set types for the protected objects

- Authentication data
  - Define RC file/dir types 'Account Data' and 'Auth Data'
  - Define RC roles 'Authenticate' and 'Change Auth Data'
  - Set rights:
    - All roles may read account data (e.g. `/etc/passwd`)
    - Role 'Authenticate' may also read 'Auth Data'
    - 'Change Auth Data' may read and write 'Account Data' and 'Auth Data'
  - Assign roles to identified programs as initial roles or forced roles
  - Optional: Assign role 'Change Auth Data' to user 'Account Manager'
### 5.4 Requirements to RC III

- **Service encapsulation**
  - Define RC role(s) for service
    - Copy existing role, e.g. 'General User'
  - Define RC file/dir types for service specific data
    - Log dirs, data, file server areas etc.
  - Set role rights:
    - Access own types as necessary
    - SEARCH, READ_OPEN, READ, CLOSE and EXECUTE libraries
    - Only SEARCH 'General Type' for path resolution
    - Optional: read and write on /tmp dirs (try to avoid)
    - No access to other FD types
    - Device type access as required
  - Assign roles to service users or program file (root services)
    - User’s default role or program file initial / forced role
  - Optional: Define default process create type for role
    - Protect against signals and tracing by others

### 5.5 Requirements to ACL II

- **Authorization data**
  - Only user, group or RC role based protection possible
  - Set inheritance mask to filter out unnecessary rights to these objects
  - Explicitly grant necessary accesses for special task users (or RC roles)

- **Service encapsulation**
  - Only user, group or RC role based protection possible
  - Group everyone might have to be replaced by a controlled group
  - Set service user rights:
    - Access own dirs/files as necessary
    - SEARCH, READ_OPEN, READ, CLOSE and EXECUTE libraries
    - Only SEARCH :DEFAULT: for path resolution
    - Optional: read and write on /tmp dirs (try to avoid)
    - No access to other FD objects
    - Device access as required

### 5.5 Requirements to ACL

- **Protect executables, libraries, configuration files, kernel objects, boot files and /tmp dirs**
  - Set inheritance mask to filter out unnecessary rights to these objects

- **Protect against execution of uncontrolled files**
  - Explicitly grant SEARCH, READ_OPEN, READ, CLOSE and EXECUTE right for group 'Everyone' to all executables and libraries
  - Remove EXECUTE right from FD :DEFAULT:

- **Protect devices**
  - Set inheritance mask to filter out unnecessary rights to these objects
  - Explicitly grant necessary accesses for special task users (or groups / RC roles), e.g. for fsck

### 5.6 Requirements to Logging Setup

- **Set individual logging for identified objects and requests**

- **Set individual user and program logging for identified requests**

- **Use RSBAC own logging source at**
  - /proc/rsbac-info/rmsg for untamperable logging
### 6 Hands-On Part

6.1 Select Simple Server Type:
- Webserver, Proxy Server, Mail or File Server?

6.2 Specify Requirements
- Filesystem Structure
- Executables
- Libraries
- Configuration Files
- Kernel Objects
- Devices
- Authentication data
- Other Objects

### 6 Hands-On Part II

6.3 Select Models

6.4 Design a Configuration

6.5 Implement It

### 7 Ending It Up

7.1 Conclusion: What We Learned

7.2 How to Go On

7.3 Open End with Questions

### Rule Set Based Access Control (RSBAC)

Linux Kernel Security Extension

Amon Ott <ao@rsbac.org>

Thank you!