Novell AppArmor (2.0) Quick Start

This document helps you understand the main concepts behind Novell® AppArmor—the content of AppArmor profiles. Learn how to create or modify AppArmor profiles. There are three ways to do this:

1. With YaST (graphical or in ncurses mode)
2. Using command line tools
3. Using a text editor (especially for fine-tuning)

AppArmor Modes

Complain
: In complain or learning mode, violations of AppArmor profile rules, such as the profiled program accessing files not permitted by the profile, are detected. The violations are permitted, but also logged. This mode is convenient for developing profiles.

  Manually activating complain mode (using the command line) adds a flag to the top of the profile so that `/bin/foo` becomes `/bin/foo flags=(complain)`.

enforce
: Loading a profile in enforcement mode enforces the policy defined in the profile as well as reports policy violation attempts to syslogd.

Starting and Stopping AppArmor

Use the `rcapparmor` command with one of the following parameters:

`start`
: Load the kernel module, mount securityfs, parse and load profiles, and start event-log.

`stop`
: Stop event-log, unmount securityfs, and invalidate profiles.

`reload`
: Reload profiles.

`status`
: If AppArmor is enabled, output how many profiles are loaded in complain or enforce mode.

AppArmor Command Line Tools

autodep
: Guess basic AppArmor profile requirements. autodep creates an approximate profile for the program or application examined. The resulting profile is called “approximate” because it does not necessarily contain all of the profile entries that the program needs to be confined properly.

complain
: Set an AppArmor profile to complain mode.

enforce
: Set an AppArmor profile to enforce mode from complain mode.

genprof
: Generate a profile. When running, you must specify a program to profile. If the specified program is not an absolute path, genprof searches the $PATH variable. If a profile does not exist, genprof creates one using autodep.

logprof
: Manage AppArmor profiles. logprof is an interactive tool used to review the learning or complain mode output found in the AppArmor syslog entries and to generate new entries in AppArmor profiles.

unconfined
: Output a list of processes with tcp or udp ports that do not have AppArmor profiles loaded.

Methods of Profiling

Stand-Alone Profiling
: Using genprof. Suitable for profiling small applications.

Systemic Profiling
: Suitable for profiling large numbers of programs all at once and for profiling applications that may run "forever."

  To apply systemic profiling, proceed as follows:

  1. Create profiles for the individual programs that make up your application (autodep).
  2. Put relevant profiles into learning or complain mode.
  3. Exercise your application.
  4. Analyze the log (logprof).
  5. Repeat Steps 3-4.
  6. Edit the profiles.
  7. Return to enforce mode.
  8. Rescan all profiles (`rcapparmor restart`).

Learning Mode

When using genprof, logprof, or YaST in learning mode, you get several options for how to proceed:

Allow
: Grant access.

Deny
: Prevent access.

Glob
: Modify the directory path to include all files in the suggested directory.

Glob w/Ext
: Modify the original directory path while retaining the filename extension. This allows the program to access all files in the suggested directories that end with the specified extension.

Edit
: Enable editing of the highlighted line. The new (edited) line appears at the bottom of the list. This option is called *New* in the logprof and genprof command line tools.

Abort
: Abort logprof or YaST, losing all rule changes entered so far and leaving all profiles unmodified.

Finish
: Close logprof or YaST, saving all rule changes entered so far and modifying all profiles.

Example Profile

```
# a variable definition
@{HOME} = /home/*/ /root/

# a comment about foo.
/usr/bin/foo {
  /bin/mount           ux,
  /dev/{,u}random      r,
  /etc/ld.so.cache     r,
  /etc/foo.conf        r,
  /etc/foo/*           r,
  /lib/ld-*.so*        mr,
  /lib/lib*.so*        mr,
  /proc/[0-9]**        r,
  /usr/lib/**          mr,
  /tmp/foo.pid         wr,
  /tmp/foo.*           lrw,
  /@{HOME}/.foo_file   rw,

  # a comment about foo's subprofile, bar.
  ^bar {
    /lib/ld-*.so*        mr,
    /usr/bin/bar         px,
    /var/spool/*         rwl,
  }
}
```

Structure of a Profile

Profiles are simple text files in the `/etc/apparmor.d` directory. They consist of several parts: #include, capability entries, rules, and "hats."

#include

This is the section of an AppArmor profile that refers to an include file, which procures access permissions for programs. By using an include, you can give the program access to directory paths or files that are also required by other programs. Using includes can reduce the size of a profile. It is good practice to select includes when suggested.

To assist you in profiling your applications, AppArmor provides three classes of `#includes`: abstractions, program chunks, and variables.

Abstractions are #includes that are grouped by common application tasks. These tasks include access to authentication mechanisms, access to name service routines, common graphics requirements, and system accounting, for example, base, consoles, kerberosclient, perl, user-mail, user-tmp, authentication, bash, nameservice.

Program chunks are access controls for specific programs that a system administrator might want to control based on local site policy. Each chunk is used by a single program.

Using variables, you can design your profiles to be portable to different environments. Changes in the variable's content are just made in the variable definition while the profile containing the variable can remain untouched.

Capability Entries (POSIX.1e)

Capabilities statements are simply the word “capability” followed by the name of the POSIX.1e capability as defined in the `capabilities(7)` man page.

Rules: General Options for Files and Directories

Option	*File*
read	r
write	w
link	l

Rules: Defining Execute Permissions

For executables that may be called from the confined programs, the profile creating tools ask you for an appropriate mode, which is also reflected directly in the profile itself:

Option	*File*	*Description*
Inherit	ix	Stay in the same (parent's) profile.
Profile	px	Requires that a separate profile exists for the executed program. No environment scrubbing.
Profile	Px	Requires that a separate profile exists for the executed program. Uses environment scrubbing.
Unconstrained	ux	Executes the program without a profile. Avoid running programs in unconstrained or unconfined mode for security reasons. No environment scrubbing.
Unconstrained	Ux	Executes the program without a profile. Avoid running programs in unconstrained or unconfined mode for security reasons. This mode makes use of environment scrubbing.
Allow Executable Mapping	m	allow PROT_EXEC with `mmap(2)` calls

Running in ux Mode

Avoid running programs in ux mode as much as possible. A program running in ux mode is not only totally unprotected by AppArmor, but child processes inherit certain environment variables from the parent that might influence the child's execution behavior and create possible security risks.

For more information about the different file execute modes, refer to the `apparmor.d(5)` man page. For more information about setgid and setuid environment scrubbing, refer to the `ld.so(8)` man page.

Rules: Paths and Globbing

Glob	*Description*
`*`	Substitutes for any number of characters, except `/`.
`**`	Substitutes for any number of characters, including `/`.
`?`	Substitutes for any single character, except `/`.
`[ abc ]`	Substitutes for the single character a, b, or c.
`[ a-c ]`	Substitutes for the single character a, b, or c.
`{ ab,cd }`	Expand to one rule to match ab and another to match `cd`.

Hats

An AppArmor profile represents a security policy for an individual program instance or process. It applies to an executable program, but if a portion of the program needs different access permissions than other portions, the program can “change hats” to use a different security context, distinctive from the access of the main program. This is known as a hat or subprofile.

A profile can have an arbitrary number of subprofiles, but there are only two levels: a subprofile cannot have further sub-subprofiles.

The AppArmor ChangeHat feature requires an application modification to call the `change_hat` function. The supplied Apache module `apache2-mod-apparmor` provides this functionality.

Helpful Additions

Autodocumentation

The tool “sitar” gathers all system configuration information available from your system and creates comprehensive system documentation. It can be used to document all new and changed profiles.

Logging and Auditing

All AppArmor events are logged using the system's audit interface (log file in `/var/log/audit/audit.log`). On top of this infrastructure, event notification can be configured. Configure this feature using YaST. It is based on severity levels according to `/etc/apparmor/severity.db`. Notification frequency and type of notification (such as e-mail) can be configured.

Use YaST for generating reports in CSV or HTML format.

Directories and Files

`/sys/kernel/security/apparmor/profiles`
: Virtualized file representing the currently loaded set of profiles.

`/etc/apparmor/`
: Location of AppArmor and AppArmor tools.

`/etc/apparmor.d/`
: Location of profiles, named with the convention of replacing the / in pathnames with `.` (not for the root /) so profiles are easier to manage. For example, the profile for the program `/usr/sbin/ntpd` is named `usr.sbin.ntpd`.

`/etc/apparmor.d/abstractions/`
: Location of abstractions.

`/etc/apparmor.d/program-chunks/`
: Location of program chunks.