

# Novell AppArmor (2.3.1) 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. You can create and manage AppArmor profiles in three different ways. The most convenient interface to AppArmor is provided by means of the AppArmor YaST modules which can be used either in graphical or ncurses mode. The same functionality is provided by the AppArmor command line tools or if you just edit the profiles in a text editor.

## AppArmor Modes

### complain/learning

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 and is used by the AppArmor tools for generating profiles.

### 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. Profiles and confinement are applied to any application started after this command was executed. Processes already running at the time AppArmor is started continue to run unconfined.

### stop

Unmount securityfs, and invalidate profiles.

### reload

Reload profiles.

### status

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

Use the `rcaeventd` command to control event logging with `aa-eventd`. Use the `start` and `stop` options to toggle the status of the `aa-eventd` and check its status using the `status`.

## AppArmor Command Line Tools

### autodep

Guess basic AppArmor profile requirements. `autodep` creates a stub 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.

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

Set an AppArmor profile to enforce mode from complain mode.

Manually activating enforce mode (using the command line) removes mode flags from the top of the profile  
`/bin/foo flags=(complain)` becomes `/bin/foo`.

### genprof

Generate or update 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 open 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. Reload 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

```
#include<tunables/global>

@{HOME} = /home/*/ /root/ # variable

/usr/bin/foo {
    #include <abstractions/base>
    network inet tcp,
    capability setgid,

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

    link /etc/sysconfig/foo -> /etc/foo.conf,
    deny /etc/shadow          w,
    owner /home/*/**          rw,

    /usr/bin/foobar           cx,
    /bin/**                   px -> bin_generic

# comment on foo's local profile, foobar.
    foobar {
        /bin/bash              rmix,
        /bin/cat                rmix,
        /bin/more               rmix,
        /var/log/foobar*        rwl,
        /etc/foobar             r,
    }
}
```

## 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 mediates 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 tunables.

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.

Tunables are global variable definitions. When used in a profile, these variables expand to a value that can be changed without changing the entire profile. Therefore your profiles become portable to different environments.

### Local Variables

Local variables are defined at the head of a profile. Use local variables to create shortcuts for paths, for example to provide the base for a chrooted path:

```
@{CHROOT_BASE}=/tmp/foo
/sbin/syslog-ng {
...
# chrooted applications
@{CHROOT_BASE}/var/lib/*/dev/log w,
@{CHROOT_BASE}/var/log/** w,
...
}
```

### Aliases

Alias rules provide an alternative form of path rewriting to using variables, and are done post variable resolution:

```
alias /home/ -> /mnt/users/
```

## Network Access Control

AppArmor provides network access mediation based on network domain and type:

```
/bin/ping {
network inet dgram,
network inet raw,
...
}
```

The example would allow IPv4 network access of the datagram and raw type for the ping command. For details on the network rule syntax, refer to the Part “Confining Privileges with Novell AppArmor” (↑*Security Guide*).

### 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

<i>Option</i>	<i>File</i>
read	r
write	w
link	l
file locking	k
file append (mutually exclusive to w)	a

### Rules: Link Pair

The link mode grants permission to create links to arbitrary files, provided the link has a subset of the permissions granted by the target (subset permission test). By specifying origin and destination, the link pair rule provides greater control over how hard links are created. Link pair rules by default do not enforce the link subset permission test that the standard rules link permission requires. To force the rule to require the test the subset keyword is used. The following rules are equivalent:

```
/link l,
link subset /link -> /**,
```

### Rules: Denying rules

AppArmor provides `deny` rules which are standard rules but with the keyword `deny` prepended. They are used to remember known rejects, and quiet them so the reject messages don't fill up the log files. For more information see Part “Confining Privileges with Novell AppArmor” (↑*Security Guide*).

## Rules: Owner Conditional Rules

The file rules can be extended so that they can be conditional upon the the user being the owner of the file. by prepending the keyword `owner` to the rule. Owner conditional rules accumulate just as regular file rules and are considered a subset of regular file rules. If a regular file rule overlaps with an owner conditional file rule, the resultant permissions will be that of the regular file rule.

## 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:

<b>Option</b>	<b>File</b>	<b>Description</b>
Inherit	<code>ix</code>	Stay in the same (parent's) profile.
Profile	<code>px</code>	Requires that a separate profile exists for the executed program. Use <code>Px</code> to make use of environment scrubbing.
Local profile	<code>cx</code>	Requires that a local profile exists for the executed program. Use <code>Cx</code> to make use of environment scrubbing.
Unconstrained	<code>ux</code>	Executes the program without a profile. Avoid running programs in unconstrained or unconfined mode for security reasons. Use <code>Ux</code> to make use of environment scrubbing.
Allow Executable Mapping	<code>m</code>	<code>allow PROT_EXEC with mmap (2) calls</code>

### WARNING: 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

AppArmor supports explicit handling of directories. Use a trailing `/` for any directory path that needs to be explicitly distinguished:

```
/some/random/example/* r
    Allow read access to files in the /some/random/
    example directory.
/some/random/example/ r
    Allow read access to the directory only.
/some/**/ r
    Give read access to any directories below /some.
/some/random/example/** r
    Give read access to files and directories under /some/
    random/example.
/some/random/example/**[^/] r
    Give read access to files under /some/random/
    example. Explicitly exclude directories ([^/]).
```

To spare users from specifying similar paths all over again, AppArmor supports basic globbing:

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

## Rules: Auditing rules

AppArmor provides the ability to audit given rules so that when they are matched an audit message will appear in the audit log. To enable audit messages for a given rule the audit keyword is prepended to the rule:

```
audit /etc/foo/* rw,
```

## Rules: Setting Capabilities

Normally AppArmor only restricts existing native Linux controls and does not grant additional privileges. The only exception from this strict rule is the set capability rule. For security reasons, set capability rules will not be inherited, so once a program leaves the profile, it loses the elevated privilege. Setting a capability also implicitly adds a capability rule allowing that capability. Since this rule allows to give processes root privileges it should be used with extreme caution and only in exceptional cases.

```
set capability cap_chown,
```

## 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 hats, but there are only two levels: a hat cannot have further hats.

The AppArmor ChangeHat feature can be used by applications to access hats during execution. Currently the packages `apache2-mod_apparmor` and `tomcat_apparmor` utilize ChangeHat to provide sub-process confinement for the Apache Web server and the Tomcat servlet container.

## Confining Users with pam\_apparmor

The `pam_apparmor` PAM module allows applications to confine authenticated users into subprofiles based on group names, user names, or a default profile. To accomplish this, `pam_apparmor` needs to be registered as a PAM session module.

Details about how to set up and configure `pam_apparmor` can be found in `/usr/share/doc/packages/pam_apparmor/README`. A HOWTO on setting up role-based access control (RBAC) with `pam_apparmor` is available at [http://developer.novell.com/wiki/index.php/Apparmor\\_RBAC\\_in\\_version\\_2.3](http://developer.novell.com/wiki/index.php/Apparmor_RBAC_in_version_2.3).

## Logging and Auditing

All AppArmor events are logged using the system's audit interface (the `auditd` logging to `/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.

If `auditd` is not running, AppArmor logs to the system log located under `/var/log/messages` using the `LOG_KERN` facility.

Use YaST for generating reports in CSV or HTML format.

The Linux audit framework contains a dispatcher that can send AppArmor events to any consumer application via `dbus`. The GNOME AppArmor Desktop Monitor applet is one example of an application that gathers AppArmor events via `dbus`. To configure audit to use the `dbus` dispatcher, just set the dispatcher in your audit configuration in `/etc/audit/auditd.conf` to `apparmor-dbus` and restart `auditd`:

```
dispatcher=/usr/bin/apparmor-dbus
```

Once the `dbus` dispatcher is configured correctly, add the AppArmor Desktop Monitor to the GNOME panel. As soon as a `REJECT` event is logged, the applet's panel icon changes appearance and you can click the applet to see the number of reject events per confined application. To view the exact log messages, refer to the audit log under `/var/log/audit/audit.log`. Use the YaST Update Profile Wizard to adjust the respective profile.

## Directories and Files

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

`/etc/apparmor/`  
Location of AppArmor configuration files.

`/etc/apparmor/profiles/extras/`  
A local repository of profiles shipped with AppArmor, but not enabled by default.

`/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.

`/proc/*/attr/current`  
Review the confinement status of a process and the profile that is used to confine the process. The `ps auxZ` command retrieves this information automatically.

## For More Information

To learn more about the AppArmor project, check out the project's home page under <http://en.opensuse.org/AppArmor>. Find more information on the concept and the configuration of AppArmor in Part “Confining Privileges with Novell AppArmor” (↑*Security Guide*).

## Legal Notice

All content is copyright © 2006– 2009 Novell, Inc.

This manual is protected under Novell intellectual property rights. By reproducing, duplicating or distributing this manual you explicitly agree to conform to the terms and conditions of this license agreement.

This manual may be freely reproduced, duplicated and distributed either as such or as part of a bundled package in electronic and/or printed format, provided however that the following conditions are fulfilled:

That this copyright notice and the names of authors and contributors appear clearly and distinctively on all reproduced, duplicated and distributed copies. That this manual, specifically for the printed format, is reproduced and/or distributed for noncommercial use only. The express authorization of Novell, Inc must be obtained prior to any other use of any manual or part thereof.

For Novell trademarks, see the Novell Trademark and Service Mark list <http://www.novell.com/company/>

[legal/trademarks/tmlist.html](http://www.novell.com/legal/trademarks/tmlist.html). Linux\* is a registered trademark of Linus Torvalds. All other third party trademarks are the property of their respective owners. A trademark symbol (®, ™ etc.) denotes a Novell trademark; an asterisk (\*) denotes a third party trademark.

All information found in this book has been compiled with utmost attention to detail. However, this does not guarantee complete accuracy. Neither Novell, Inc., SUSE LINUX Products GmbH, the authors, nor the translators shall be held liable for possible errors or the consequences thereof.

Novell.

