Applies to **openSUSE Leap 42.3**

24 Building Profiles from the Command Line

AppArmor® provides the user the ability to use a command line interface rather than a graphical interface to manage and configure the system security. Track the status of AppArmor and create, delete, or modify AppArmor profiles using the AppArmor command line tools.

Tip: Background Information

Before starting to manage your profiles using the AppArmor command line tools, check out the general introduction to AppArmor given in Chapter 20, *Immunizing Programs* and Chapter 21, *Profile Components and Syntax*.

24.1 Checking the AppArmor Status

AppArmor can be in any one of three states:

Unloaded
AppArmor is not activated in the kernel.

Running
AppArmor is activated in the kernel and is enforcing AppArmor program policies.

Stopped
AppArmor is activated in the kernel, but no policies are enforced.

Detect the state of AppArmor by inspecting `/sys/kernel/security/apparmor/profiles`. If **`cat /sys/kernel/security/apparmor/profiles`** reports a list of profiles, AppArmor is running. If it is empty and returns nothing, AppArmor is stopped. If the file does not exist, AppArmor is unloaded.

Manage AppArmor with **`systemctl`**. It lets you perform the following operations:

`sudo systemctl start apparmor`
Behavior depends on the state of AppArmor. If it is not activated, `start` activates and starts it, putting it in the running state. If it is stopped, `start` causes the re-scan of AppArmor profiles usually found in `/etc/apparmor.d` and puts AppArmor in the running state. If AppArmor is already running, `start` reports a warning and takes no action.

Note: Already Running Processes

Already running processes need to be restarted to apply the AppArmor profiles on them.

`sudo systemctl stop apparmor`
Stops AppArmor if it is running by removing all profiles from kernel memory, effectively disabling all access controls, and putting AppArmor into the stopped state. If the AppArmor is already stopped, `stop` tries to unload the profiles again, but nothing happens.

`sudo systemctl reload apparmor`
Causes the AppArmor module to re-scan the profiles in `/etc/apparmor.d` without unconfining running processes. Freshly created profiles are enforced and recently deleted ones are removed from the `/etc/apparmor.d` directory.

24.2 Building AppArmor Profiles

The AppArmor module profile definitions are stored in the `/etc/apparmor.d` directory as plain text files. For a detailed description of the syntax of these files, refer to Chapter 21, *Profile Components and Syntax*.

All files in the `/etc/apparmor.d` directory are interpreted as profiles and are loaded as such. Renaming files in that directory is not an effective way of preventing profiles from being loaded. You must remove profiles from this directory to prevent them from being read and evaluated effectively, or call **`aa-disable`** on the profile, which will create a symbolic link in `/etc/apparmor.d/disabled/`.

You can use a text editor, such as **`vi`**, to access and make changes to these profiles. The following sections contain detailed steps for building profiles:

Adding or Creating AppArmor Profiles
Refer to Section 24.3, “Adding or Creating an AppArmor Profile”

Editing AppArmor Profiles
Refer to Section 24.4, “Editing an AppArmor Profile”

Deleting AppArmor Profiles
Refer to Section 24.6, “Deleting an AppArmor Profile”

24.3 Adding or Creating an AppArmor Profile

To add or create an AppArmor profile for an application, you can use a systemic or stand-alone profiling method, depending on your needs. Learn more about these two approaches in Section 24.7, “Two Methods of Profiling”.

24.4 Editing an AppArmor Profile

The following steps describe the procedure for editing an AppArmor profile:

1. If you are not currently logged in as `root`, enter **`su`** in a terminal window.
2. Enter the `root` password when prompted.
3. Go to the profile directory with **`cd /etc/apparmor.d/`**.

4. Enter `ls` to view all profiles currently installed.
5. Open the profile to edit in a text editor, such as vim.
6. Make the necessary changes, then save the profile.
7. Restart AppArmor by entering `systemctl reload apparmor` in a terminal window.

24.5 Unloading Unknown AppArmor Profiles

Warning: Danger of Unloading Wanted Profiles

`aa-remove-unkown` will unload all profiles that are not stored in `/etc/apparmor.d`, for example automatically generated LXD profiles. This may compromise the security of the system. Use the `-n` parameter to list all profiles that will be unloaded.

To unload all profiles that are not longer in `/etc/apparmor.d/` AppArmor profiles, run:

```
tux > sudo aa-remove-unknown
```

You can print a list of profiles that will be removed:

```
tux > sudo aa-remove-unknown -n
```

24.6 Deleting an AppArmor Profile

The following steps describe the procedure for deleting an AppArmor profile.

1. Remove the AppArmor definition from the kernel:

   ```
   tux > sudo apparmor_parser -R /etc/apparmor.d/PROFILE
   ```

2. Remove the definition file:

   ```
   tux > sudo rm /etc/apparmor.d/PROFILE
       tux > sudo rm /var/lib/apparmor/cache/PROFILE
   ```

24.7 Two Methods of Profiling

Given the syntax for AppArmor profiles in Chapter 21, *Profile Components and Syntax*, you could create profiles without using the tools. However, the effort involved would be substantial. To avoid such a situation, use the AppArmor tools to automate the creation and refinement of profiles.

There are two ways to approach AppArmor profile creation. Tools are available for both methods.

Stand-Alone Profiling
: A method suitable for profiling small applications that have a finite runtime, such as user client applications like mail clients. For more information, refer to Section 24.7.1, “Stand-Alone Profiling”.

Systemic Profiling
: A method suitable for profiling many programs at once and for profiling applications that may run for days, weeks, or continuously across reboots, such as network server applications like Web servers and mail servers. For more information, refer to Section 24.7.2, “Systemic Profiling”.

Automated profile development becomes more manageable with the AppArmor tools:

1. Decide which profiling method suits your needs.
2. Perform a static analysis. Run either **`aa-genprof`** or **`aa-autodep`**, depending on the profiling method chosen.
3. Enable dynamic learning. Activate learning mode for all profiled programs.

24.7.1 Stand-Alone Profiling

Stand-alone profile generation and improvement is managed by a program called **`aa-genprof`**. This method is easy because **`aa-genprof`** takes care of everything, but is limited because it requires **`aa-genprof`** to run for the entire duration of the test run of your program (you cannot reboot the machine while you are still developing your profile).

To use **`aa-genprof`** for the stand-alone method of profiling, refer to Section 24.7.3.8, “aa-genprof—Generating Profiles”.

24.7.2 Systemic Profiling

This method is called *systemic profiling* because it updates all of the profiles on the system at once, rather than focusing on the one or few targeted by **aa-genprof** or stand-alone profiling. With systemic profiling, profile construction and improvement are somewhat less automated, but more flexible. This method is suitable for profiling long-running applications whose behavior continues after rebooting, or many programs at once.

Build an AppArmor profile for a group of applications as follows:

1. Create profiles for the individual programs that make up your application.

   Although this approach is systemic, AppArmor only monitors those programs with profiles and their children. To get AppArmor to consider a program, you must at least have **aa-autodep** create an approximate profile for it. To create this approximate profile, refer to Section 24.7.3.1, “aa-autodep—Creating Approximate Profiles”.

2. Put relevant profiles into learning or complain mode.

   Activate learning or complain mode for all profiled programs by entering

   ```
   aa-complain /etc/apparmor.d/*
   ```

   in a terminal window while logged in as `root`. This functionality is also available through the YaST Profile Mode module, described in Section 23.4.2, “Changing the Mode of Individual Profiles”.

   When in learning mode, access requests are not blocked, even if the profile dictates that they should be. This enables you to run through several tests (as shown in Step 3) and learn the access needs of the program so it runs properly. With this information, you can decide how secure to make the profile.

   Refer to Section 24.7.3.2, “aa-complain—Entering Complain or Learning Mode” for more detailed instructions for using learning or complain mode.

3. Exercise your application.

   Run your application and exercise its functionality. How much to exercise the program is up to you, but you need the program to access each file representing its access needs. Because the execution is not being supervised by **aa-genprof**, this step can go on for days or weeks and can span complete system reboots.

4. Analyze the log.

In systemic profiling, run **aa-logprof** directly instead of letting **aa-genprof** run it (as in stand-alone profiling). The general form of **aa-logprof** is:

```
aa-logprof [ -d /path/to/profiles ] [ -f /path/to/logfile ]
```

Refer to Section 24.7.3.9, "aa-logprof—Scanning the System Log" for more information about using **aa-logprof**.

5. Repeat Step 3 and Step 4.

   This generates optimal profiles. An iterative approach captures smaller data sets that can be trained and reloaded into the policy engine. Subsequent iterations generate fewer messages and run faster.

6. Edit the profiles.

   You should review the profiles that have been generated. You can open and edit the profiles in `/etc/apparmor.d/` using a text editor.

7. Return to enforce mode.

   This is when the system goes back to enforcing the rules of the profiles, not only logging information. This can be done manually by removing the `flags=(complain)` text from the profiles or automatically by using the **aa-enforce** command, which works identically to the **aa-complain** command, except it sets the profiles to enforce mode. This functionality is also available through the YaST Profile Mode module, described in Section 23.4.2, "Changing the Mode of Individual Profiles".

   To ensure that all profiles are taken out of complain mode and put into enforce mode, enter **aa-enforce /etc/apparmor.d/***.

8. Re-scan all profiles.

   To have AppArmor re-scan all of the profiles and change the enforcement mode in the kernel, enter **systemctl reload apparmor**.

24.7.3 Summary of Profiling Tools

All of the AppArmor profiling utilities are provided by the `apparmor-utils` RPM package and are stored in `/usr/sbin`. Each tool has a different purpose.

24.7.3.1 aa-autodep—Creating Approximate Profiles

This creates an approximate profile for the program or application selected. You can generate approximate profiles for binary executables and interpreted script programs. The resulting profile is called “approximate” because it does not necessarily contain all of the profile entries that the program needs to be properly confined by AppArmor. The minimum **aa-autodep** approximate profile has, at minimum, a base include directive, which contains basic profile entries needed by most programs. For certain types of programs, **aa-autodep** generates a more expanded profile. The profile is generated by recursively calling **ldd(1)** on the executables listed on the command line.

To generate an approximate profile, use the **aa-autodep** program. The program argument can be either the simple name of the program, which **aa-autodep** finds by searching your shell's path variable, or it can be a fully qualified path. The program itself can be of any type (ELF binary, shell script, Perl script, etc.). **aa-autodep** generates an approximate profile to improve through the dynamic profiling that follows.

The resulting approximate profile is written to the `/etc/apparmor.d` directory using the AppArmor profile naming convention of naming the profile after the absolute path of the program, replacing the forward slash (`/`) characters in the path with period (`.`) characters. The general syntax of **aa-autodep** is to enter the following in a terminal window when logged in as `root` :

```
aa-autodep [ -d /PATH/TO/PROFILES ] [PROGRAM1 PROGRAM2...]
```

If you do not enter the program name or names, you are prompted for them. */path/to/profiles* overrides the default location of `/etc/apparmor.d`, should you keep profiles in a location other than the default.

To begin profiling, you must create profiles for each main executable service that is part of your application (anything that might start without being a child of another program that already has a profile). Finding all such programs depends on the application in question. Here are several strategies for finding such programs:

Directories

If all the programs to profile are in one directory and there are no other programs in that directory, the simple command **aa-autodep** */path/to/your/programs/** creates basic profiles for all programs in that directory.

pstree -p

You can run your application and use the standard Linux **pstree** command to find all processes running. Then manually hunt down the location of these programs and run the **aa-autodep** for each one. If the programs are in your path, **aa-autodep** finds them for you. If they are not in your path, the standard Linux command **find** might be helpful in finding your programs. Execute **find / -name '** *MY_APPLICATION*' -print to determine an application's path (*MY_APPLICATION* being an example application). You may use wild cards if appropriate.

24.7.3.2 aa-complain—Entering Complain or Learning Mode

The complain or learning mode tool (**aa-complain**) detects violations of AppArmor profile rules, such as the profiled program accessing files not permitted by the profile. The violations are permitted, but also logged. To improve the profile, turn complain mode on, run the program through a suite of tests to generate log events that characterize the program's access needs, then postprocess the log with the AppArmor tools to transform log events into improved 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)` . To use complain mode, open a terminal window and enter one of the following lines as `root` :

- If the example program (*PROGRAM1*) is in your path, use:

```
aa-complain [PROGRAM1 PROGRAM2 ...]
```

- If the program is not in your path, specify the entire path as follows:

```
aa-complain /sbin/PROGRAM1
```

- If the profiles are not in `/etc/apparmor.d` , use the following to override the default location:

```
aa-complain /path/to/profiles/PROGRAM1
```

- Specify the profile for */sbin/program1* as follows:

```
aa-complain /etc/apparmor.d/sbin.PROGRAM1
```

Each of the above commands activates the complain mode for the profiles or programs listed. If the program name does not include its entire path, **aa-complain** searches $PATH for the program. For example, **aa-complain /usr/sbin/*** finds profiles associated with all of the programs in /usr/sbin and puts them into complain mode. **aa-complain /etc/apparmor.d/*** puts all of the profiles in /etc/apparmor.d into complain mode.

Tip: Toggling Profile Mode with YaST

YaST offers a graphical front-end for toggling complain and enforce mode. See Section 23.4.2, “Changing the Mode of Individual Profiles” for information.

24.7.3.3 aa-decode—Decoding Hex-encoded Strings in AppArmor Log Files

aa-decode will decode hex-encoded strings in the AppArmor log output. It can also process the audit log on standard input, convert any hex-encoded AppArmor log entries, and display them on standard output.

24.7.3.4 aa-disable—Disabling an AppArmor Security Profile

Use **aa-disable** to disable the enforcement mode for one or more AppArmor profiles. This command will unload the profile from the kernel, and prevent the profile from being loaded on AppArmor start-up. Use **aa-enforce** or **aa-complain** utilities to change this behavior.

24.7.3.5 aa-easyprof—Easy Profile Generation

aa-easyprof provides an easy-to-use interface for AppArmor profile generation. **aa-easyprof** supports the use of templates and profile groups to quickly profile an application. While **aa-easyprof** can help with profile generation, its utility is dependent on the quality of the templates, profile groups and abstractions used. Also, this tool may create a profile that is less restricted than when creating a profile manually or with **aa-genprof** and **aa-logprof**.

For more information, see the man page of **aa-easyprof** (8).

24.7.3.6 aa-enforce—Entering Enforce Mode

The enforce mode detects violations of AppArmor profile rules, such as the profiled program accessing files not permitted by the profile. The violations are logged and not permitted. The default is for enforce mode to be enabled. To log the violations only, but still permit them, use complain mode.

Manually activating enforce mode (using the command line) removes the complain flag from the top of the profile so that `/bin/foo flags=(complain)` becomes `/bin/foo`. To use enforce mode, open a terminal window and enter one of the following lines as `root`.

- If the example program (*`PROGRAM1`*) is in your path, use:

```
aa-enforce [PROGRAM1 PROGRAM2 ...]
```

- If the program is not in your path, specify the entire path, as follows:

```
aa-enforce /sbin/PROGRAM1
```

- If the profiles are not in */etc/apparmor.d*, use the following to override the default location:

```
aa-enforce -d /path/to/profiles/     program1
```

- Specify the profile for */sbin/program1* as follows:

```
aa-enforce /etc/apparmor.d/sbin.PROGRAM1
```

Each of the above commands activates the enforce mode for the profiles and programs listed.

If you do not enter the program or profile names, you are prompted to enter one. */path/to/profiles* overrides the default location of `/etc/apparmor.d`.

The argument can be either a list of programs or a list of profiles. If the program name does not include its entire path, **`aa-enforce`** searches `$PATH` for the program.

Tip: Toggling Profile Mode with YaST

YaST offers a graphical front-end for toggling complain and enforce mode. See Section 23.4.2, “Changing the Mode of Individual Profiles” for information.

24.7.3.7 aa-exec—Confining a Program with the Specified Profile

Use **aa-exec** to launch a program confined by a specified profile and/or profile namespace. If both a profile and namespace are specified, the program will be confined by the profile in the new namespace. If only a profile namespace is specified, the profile name of the current confinement will be used. If neither a profile nor namespace is specified, the command will be run using the standard profile attachment—as if you did not use the **aa-exec** command.

For more information on the command's options, see its manual page **man 8 aa-exec**.

24.7.3.8 aa-genprof—Generating Profiles

aa-genprof is AppArmor's profile generating utility. It runs **aa-autodep** on the specified program, creating an approximate profile (if a profile does not already exist for it), sets it to complain mode, reloads it into AppArmor, marks the log, and prompts the user to execute the program and exercise its functionality. Its syntax is as follows:

```
aa-genprof [ -d /path/to/profiles ]  PROGRAM
```

To create a profile for the Apache Web server program httpd2-prefork, do the following as `root`:

1. Enter **systemctl stop apache2**.
2. Next, enter **aa-genprof httpd2-prefork**.

   Now **aa-genprof** does the following:

   1. Resolves the full path of httpd2-prefork using your shell's path variables. You can also specify a full path. On openSUSE Leap, the default full path is `/usr/sbin/httpd2-prefork`.
   2. Checks to see if there is an existing profile for httpd2-prefork. If there is one, it updates it. If not, it creates one using the **aa-autodep** as described in Section 24.7.3, “Summary of Profiling Tools”.
   3. Puts the profile for this program into learning or complain mode so that profile violations are logged, but are permitted to proceed. A log event looks like this (see `/var/log/audit/audit.log`):

      ```
      type=APPARMOR_ALLOWED msg=audit(1189682639.184:20816): \
      apparmor="DENIED" operation="file_mmap" parent=2692 \
      profile="/usr/sbin/httpd2-prefork//HANDLING_UNTRUSTED_INPUT" \
      name="/var/log/apache2/access_log-20140116" pid=28730 comm="http
      requested_mask="::r" denied_mask="::r" fsuid=30 ouid=0
      ```

If you are not running the audit daemon, the AppArmor events are logged directly to `systemd` journal (see *Book "Reference", Chapter 11 "* **`journalctl`** *: Query the* `systemd` *Journal"*):

```
Sep 13 13:20:30 K23 kernel: audit(1189682430.672:20810): \
apparmor="DENIED" operation="file_mmap" parent=2692 \
profile="/usr/sbin/httpd2-prefork//HANDLING_UNTRUSTED_INPUT" \
name="/var/log/apache2/access_log-20140116" pid=28730 comm="http
requested_mask="::r" denied_mask="::r" fsuid=30 ouid=0
```

They also can be viewed using the **`dmesg`** command:

```
audit(1189682430.672:20810): apparmor="DENIED" \
operation="file_mmap" parent=2692 \
profile="/usr/sbin/httpd2-prefork//HANDLING_UNTRUSTED_INPUT" \
name="/var/log/apache2/access_log-20140116" pid=28730 comm="http
requested_mask="::r" denied_mask="::r" fsuid=30 ouid=0
```

4. Marks the log with a beginning marker of log events to consider. For example:

```
Sep 13 17:48:52 figwit root: GenProf: e2ff78636296f16d0b5301209a
```

3. When prompted by the tool, run the application to profile in another terminal window and perform as many of the application functions as possible. Thus, the learning mode can log the files and directories to which the program requires access to function properly. For example, in a new terminal window, enter **`systemctl start apache2`**.
4. Select from the following options that are available in the **`aa-genprof`** terminal window after you have executed the program function:
   - S runs **`aa-genprof`** on the system log from where it was marked when **`aa-genprof`** was started and reloads the profile. If system events exist in the log, AppArmor parses the learning mode log files. This generates a series of questions that you must answer to guide **`aa-genprof`** in generating the security profile.
   - F exits the tool.

> **Note**
>
> If requests to add hats appear, proceed to Chapter 25, *Profiling Your Web Applications Using ChangeHat*.

5. Answer two types of questions:

- A resource is requested by a profiled program that is not in the profile (see Example 24.1, “Learning Mode Exception: Controlling Access to Specific Resources”).
- A program is executed by the profiled program and the security domain transition has not been defined (see Example 24.2, “Learning Mode Exception: Defining Permissions for an Entry”).

Each of these categories results in a series of questions that you must answer to add the resource or program to the profile. Example 24.1, “Learning Mode Exception: Controlling Access to Specific Resources” and Example 24.2, “Learning Mode Exception: Defining Permissions for an Entry” provide examples of each one. Subsequent steps describe your options in answering these questions.

- Dealing with execute accesses is complex. You must decide how to proceed with this entry regarding which execute permission type to grant to this entry:

  EXAMPLE 24.1: LEARNING MODE EXCEPTION: CONTROLLING ACCESS TO SPECIFIC RESOURCES

  ```
  Reading log entries from /var/log/audit/audit.log.
  Updating AppArmor profiles in /etc/apparmor.d.

  Profile:  /usr/sbin/xinetd
  Program:  xinetd
  Execute:  /usr/lib/cups/daemon/cups-lpd
  Severity: unknown

  (I)nherit / (P)rofile / (C)hild / (N)ame / (U)nconfined / (X)i
  ```

  Inherit (ix)

  The child inherits the parent's profile, running with the same access controls as the parent. This mode is useful when a confined program needs to call another confined program without gaining the permissions of the target's profile or losing the permissions of the current profile. This mode is often used when the child program is a *helper application*, such as the **`/usr/bin/mail`** client using **`less`** as a pager.

  Profile (px/Px)

The child runs using its own profile, which must be loaded into the kernel. If the profile is not present, attempts to execute the child fail with permission denied. This is most useful if the parent program is invoking a global service, such as DNS lookups or sending mail with your system's MTA.
Choose the *profile with clean exec* (Px) option to scrub the environment of environment variables that could modify execution behavior when passed to the child process.

Child (cx/Cx)

Sets up a transition to a subprofile. It is like px/Px transition, except to a child profile.
Choose the *profile with clean exec* (Cx) option to scrub the environment of environment variables that could modify execution behavior when passed to the child process.

Unconfined (ux/Ux)

The child runs completely unconfined without any AppArmor profile applied to the executed resource.
Choose the *unconfined with clean exec* (Ux) option to scrub the environment of environment variables that could modify execution behavior when passed to the child process. Note that running unconfined profiles introduces a security vulnerability that could be used to evade AppArmor. Only use it as a last resort.

mmap (m)

This permission denotes that the program running under the profile can access the resource using the mmap system call with the flag `PROT_EXEC`. This means that the data mapped in it can be executed. You are prompted to include this permission if it is requested during a profiling run.

Deny

Adds a `deny` rule to the profile, and permanently prevents the program from accessing the specified directory path entries. AppArmor then continues to the next event.

Abort

Aborts **`aa-logprof`**, losing all rule changes entered so far and leaving all profiles unmodified.

Finish

Closes **`aa-logprof`**, saving all rule changes entered so far and modifying all profiles.

- Example 24.2, “Learning Mode Exception: Defining Permissions for an Entry” shows AppArmor suggest allowing a globbing pattern `/var/run/nscd/*` for reading, then using an abstraction to cover common Apache-related access rules.

EXAMPLE 24.2: LEARNING MODE EXCEPTION: DEFINING PERMISSIONS FOR AN ENTRY

```
Profile:  /usr/sbin/httpd2-prefork
Path:     /var/run/nscd/dbSz9CTr
Mode:     r
Severity: 3

  1 - /var/run/nscd/dbSz9CTr
 [2 - /var/run/nscd/*]

(A)llow / [(D)eny] / (G)lob / Glob w/(E)xt / (N)ew / Abo(r)t /
Adding /var/run/nscd/* r to profile.

Profile:  /usr/sbin/httpd2-prefork
Path:     /proc/11769/attr/current
Mode:     w
Severity: 9

 [1 - #include <abstractions/apache2-common>]
  2 - /proc/11769/attr/current
  3 - /proc/*/attr/current

(A)llow / [(D)eny] / (G)lob / Glob w/(E)xt / (N)ew / Abo(r)t /
Adding #include <abstractions/apache2-common> to profile.
```

AppArmor provides one or more paths or includes. By entering the option number, select the desired options then proceed to the next step.

Note

Not all of these options are always presented in the AppArmor menu.

`#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.

Globbed Version

This is accessed by selecting *Glob* as described in the next step. For information about globbing syntax, refer to Section 21.6, “Profile Names, Flags, Paths, and Globbing”.

Actual Path

This is the literal path to which the program needs access so that it can run properly.

After you select the path or include, process it as an entry into the AppArmor profile by selecting *Allow* or *Deny*. If you are not satisfied with the directory path entry as it is displayed, you can also *Glob* it.

The following options are available to process the learning mode entries and build the profile:

Select `Enter`

Allows access to the selected directory path.

Allow

Allows access to the specified directory path entries. AppArmor suggests file permission access. For more information, refer to Section 21.7, “File Permission Access Modes”.

Deny

Prevents the program from accessing the specified directory path entries. AppArmor then continues to the next event.

New

Prompts you to enter your own rule for this event, allowing you to specify a regular expression. If the expression does not actually satisfy the event that prompted the question in the first place, AppArmor asks for confirmation and lets you reenter the expression.

Glob

Select a specific path or create a general rule using wild cards that match a broader set of paths. To select any of the offered paths, enter the number that is printed in front of the path then decide how to proceed with the selected item.
For more information about globbing syntax, refer to Section 21.6, “Profile Names, Flags, Paths, and Globbing”.

Glob w/Ext
This modifies the original directory path while retaining the file name extension. For example, `/etc/apache2/file.ext` becomes `/etc/apache2/*.ext`, adding the wild card (asterisk) in place of the file name. This allows the program to access all files in the suggested directory that end with the `.ext` extension.

Abort
Aborts **`aa-logprof`**, losing all rule changes entered so far and leaving all profiles unmodified.

Finish
Closes **`aa-logprof`**, saving all rule changes entered so far and modifying all profiles.

6. To view and edit your profile using **`vi`**, enter **`vi /etc/apparmor.d/`***`PROFILENAME`* in a terminal window. To enable syntax highlighting when editing an AppArmor profile in vim, use the commands **`:syntax on`** then **`:set syntax=apparmor`**. For more information about vim and syntax highlighting, refer to Section 24.7.3.14, “apparmor.vim”.
7. Restart AppArmor and reload the profile set including the newly created one using the **`systemctl reload apparmor`** command.

Like the graphical front-end for building AppArmor profiles, the YaST Add Profile Wizard, **`aa-genprof`** also supports the use of the local profile repository under `/etc/apparmor/profiles/extras` and the remote AppArmor profile repository.

To use a profile from the local repository, proceed as follows:

1. Start **`aa-genprof`** as described above.

   If **`aa-genprof`** finds an inactive local profile, the following lines appear on your terminal window:

   ```
   Profile: /usr/bin/opera
   ```

```
 [1 - Inactive local profile for /usr/bin/opera]

[(V)iew Profile] / (U)se Profile / (C)reate New Profile / Abo(r)t / (F
```

2. To use this profile, press U (*Use Profile*) and follow the profile generation procedure outlined above.

   To examine the profile before activating it, press V (*View Profile*).

   To ignore the existing profile, press C (*Create New Profile*) and follow the profile generation procedure outlined above to create the profile from scratch.

3. Leave **aa-genprof** by pressing F (*Finish*) when you are done and save your changes.

24.7.3.9 aa-logprof—Scanning the System Log

aa-logprof is an interactive tool used to review the complain and enforce mode events found in the log entries in `/var/log/audit/audit.log`, or directly in the `systemd` journal (see *Book "Reference", Chapter 11 "* **journalctl** *: Query the* `systemd` *Journal"*), and generate new entries in AppArmor security profiles.

When you run **aa-logprof**, it begins to scan the log files produced in complain and enforce mode and, if there are new security events that are not covered by the existing profile set, it gives suggestions for modifying the profile. **aa-logprof** uses this information to observe program behavior.

If a confined program forks and executes another program, **aa-logprof** sees this and asks the user which execution mode should be used when launching the child process. The execution modes *ix*, *px*, *Px*, *ux*, *Ux*, *cx*, *Cx*, and named profiles, are options for starting the child process. If a separate profile exists for the child process, the default selection is *Px*. If one does not exist, the profile defaults to *ix*. Child processes with separate profiles have **aa-autodep** run on them and are loaded into AppArmor, if it is running.

When **aa-logprof** exits, profiles are updated with the changes. If AppArmor is active, the updated profiles are reloaded and, if any processes that generated security events are still running in the null-XXXX profiles (unique profiles temporarily created in complain mode), those processes are set to run under their proper profiles.

To run **aa-logprof**, enter **aa-logprof** into a terminal window while logged in as `root`. The following options can be used for **aa-logprof**:

aa-logprof -d */path/to/profile/directory/*

Specifies the full path to the location of the profiles if the profiles are not located in the standard directory, `/etc/apparmor.d/`.

`aa-logprof -f` *`/path/to/logfile/`*
Specifies the full path to the location of the log file if the log file is not located in the default directory or `/var/log/audit/audit.log`.

`aa-logprof -m "string marker in logfile"`
Marks the starting point for **`aa-logprof`** to look in the system log. **`aa-logprof`** ignores all events in the system log before the specified mark. If the mark contains spaces, it must be surrounded by quotes to work correctly. For example:

```
aa-logprof -m "17:04:21"
```

or

```
aa-logprof -m e2ff78636296f16d0b5301209a04430d
```

`aa-logprof` scans the log, asking you how to handle each logged event. Each question presents a numbered list of AppArmor rules that can be added by pressing the number of the item on the list.

By default, **`aa-logprof`** looks for profiles in `/etc/apparmor.d/`. Often running **`aa-logprof`** as `root` is enough to update the profile. However, there might be times when you need to search archived log files, such as if the program exercise period exceeds the log rotation window (when the log file is archived and a new log file is started). If this is the case, you can enter **`zcat -f `ls -1tr`** *`/path/to/logfile*`* | aa-logprof -f -.

24.7.3.10 aa-logprof Example 1

The following is an example of how **`aa-logprof`** addresses httpd2-prefork accessing the file `/etc/group`. `[]` indicates the default option.

In this example, the access to **`/etc/group`** is part of httpd2-prefork accessing name services. The appropriate response is `1`, which includes a predefined set of AppArmor rules. Selecting `1` to `#include` the name service package resolves all of the future questions pertaining to DNS lookups and makes the profile less brittle in that any changes to DNS configuration and the associated name service profile package can be made once, rather than needing to revise many profiles.

```
Profile:  /usr/sbin/httpd2-prefork
Path:     /etc/group
New Mode: r

[1 - #include <abstractions/nameservice>]
```

```
 2 - /etc/group
[(A)llow] / (D)eny / (N)ew / (G)lob / Glob w/(E)xt / Abo(r)t / (F)inish
```

Select one of the following responses:

Select Enter
: Triggers the default action, which is, in this example, allowing access to the specified directory path entry.

Allow
: Allows access to the specified directory path entries. AppArmor suggests file permission access. For more information about this, refer to Section 21.7, “File Permission Access Modes”.

Deny
: Permanently prevents the program from accessing the specified directory path entries. AppArmor then continues to the next event.

New
: Prompts you to enter your own rule for this event, allowing you to specify whatever form of regular expression you want. If the expression entered does not actually satisfy the event that prompted the question in the first place, AppArmor asks for confirmation and lets you reenter the expression.

Glob
: Select either a specific path or create a general rule using wild cards that matches on a broader set of paths. To select any of the offered paths, enter the number that is printed in front of the paths then decide how to proceed with the selected item.
For more information about globbing syntax, refer to Section 21.6, “Profile Names, Flags, Paths, and Globbing”.

Glob w/Ext
: This modifies the original directory path while retaining the file name extension. For example, `/etc/apache2/file.ext` becomes `/etc/apache2/*.ext`, adding the wild card (asterisk) in place of the file name. This allows the program to access all files in the suggested directory that end with the `.ext` extension.

Abort
: Aborts **`aa-logprof`**, losing all rule changes entered so far and leaving all profiles unmodified.

Finish
: Closes **`aa-logprof`**, saving all rule changes entered so far and modifying all profiles.

24.7.3.11 aa-logprof Example 2

For example, when profiling vsftpd, see this question:

```
Profile:  /usr/sbin/vsftpd
Path:     /y2k.jpg

New Mode: r

[1 - /y2k.jpg]

(A)llow / [(D)eny] / (N)ew / (G)lob / Glob w/(E)xt / Abo(r)t / (F)inish
```

Several items of interest appear in this question. First, note that vsftpd is asking for a path entry at the top of the tree, even though vsftpd on openSUSE Leap serves FTP files from `/srv/ftp` by default. This is because vsftpd uses chroot and, for the portion of the code inside the chroot jail, AppArmor sees file accesses in terms of the chroot environment rather than the global absolute path.

The second item of interest is that you should grant FTP read access to all JPEG files in the directory, so you could use *Glob w/Ext* and use the suggested path of `/*.jpg`. Doing so collapses all previous rules granting access to individual `.jpg` files and forestalls any future questions pertaining to access to `.jpg` files.

Finally, you should grant more general access to FTP files. If you select *Glob* in the last entry, **`aa-logprof`** replaces the suggested path of `/y2k.jpg` with `/*`. Alternatively, you should grant even more access to the entire directory tree, in which case you could use the *New* path option and enter `/**.jpg` (which would grant access to all `.jpg` files in the entire directory tree) or `/**` (which would grant access to all files in the directory tree).

These items deal with read accesses. Write accesses are similar, except that it is good policy to be more conservative in your use of regular expressions for write accesses. Dealing with execute accesses is more complex. Find an example in Example 24.1, “Learning Mode Exception: Controlling Access to Specific Resources”.

In the following example, the `/usr/bin/mail` mail client is being profiled and **`aa-logprof`** has discovered that **`/usr/bin/mail`** executes **`/usr/bin/less`** as a helper application to “page” long mail messages. Consequently, it presents this prompt:

```
/usr/bin/nail -> /usr/bin/less
(I)nherit / (P)rofile / (C)hild / (N)ame / (U)nconfined / (X)ix / (D)eny
```

Note

The actual executable file for `/usr/bin/mail` turns out to be `/usr/bin/nail`, which is not a typographical error.

The program `/usr/bin/less` appears to be a simple one for scrolling through text that is more than one screen long and that is in fact what `/usr/bin/mail` is using it for. However, **`less`** is actually a large and powerful program that uses many other helper applications, such as **`tar`** and **`rpm`**.

Tip

Run **`less`** on a tar file or an RPM file and it shows you the inventory of these containers.

You do not want to run **`rpm`** automatically when reading mail messages (that leads directly to a Microsoft* Outlook–style virus attack, because RPM has the power to install and modify system programs), so, in this case, the best choice is to use *Inherit*. This results in the less program executed from this context running under the profile for `/usr/bin/mail`. This has two consequences:

- You need to add all of the basic file accesses for `/usr/bin/less` to the profile for `/usr/bin/mail`.
- You can avoid adding the helper applications, such as **`tar`** and **`rpm`**, to the `/usr/bin/mail` profile so that when `/usr/bin/mail` runs `/usr/bin/less` in this context, the less program is far less dangerous than it would be without AppArmor protection. Another option is to use the Cx execute modes. For more information on execute modes, see Section 21.8, “Execute Modes”.

In other circumstances, you might instead want to use the *Profile* option. This has the following effects on **`aa-logprof`**:

- The rule written into the profile uses px/Px, which forces the transition to the child's own profile.
- **`aa-logprof`** constructs a profile for the child and starts building it, in the same way that it built the parent profile, by assigning events for the child process to the child's profile and asking the **`aa-logprof`** user questions. The profile will also be applied if you run the child as a stand-alone program.

If a confined program forks and executes another program, **aa-logprof** sees this and asks the user which execution mode should be used when launching the child process. The execution modes of inherit, profile, unconfined, child, named profile, or an option to deny the execution are presented.

If a separate profile exists for the child process, the default selection is profile. If a profile does not exist, the default is inherit. The inherit option, or `ix`, is described in Section 21.7, “File Permission Access Modes”.

The profile option indicates that the child program should run in its own profile. A secondary question asks whether to sanitize the environment that the child program inherits from the parent. If you choose to sanitize the environment, this places the execution modifier `Px` in your AppArmor profile. If you select not to sanitize, `px` is placed in the profile and no environment sanitizing occurs. The default for the execution mode is `Px` if you select profile execution mode.

The unconfined execution mode is not recommended and should only be used in cases where there is no other option to generate a profile for a program reliably. Selecting unconfined opens a warning dialog asking for confirmation of the choice. If you are sure and choose *Yes*, a second dialog ask whether to sanitize the environment. To use the execution mode `Ux` in your profile, select *Yes*. To use the execution mode `ux` in your profile instead, select *No*. The default value selected is `Ux` for unconfined execution mode.

> **Important: Running Unconfined**
>
> Selecting `ux or Ux` is very dangerous and provides no enforcement of policy (from a security perspective) of the resulting execution behavior of the child program.

24.7.3.12 aa-unconfined—Identifying Unprotected Processes

The **aa-unconfined** command examines open network ports on your system, compares that to the set of profiles loaded on your system, and reports network services that do not have AppArmor profiles. It requires `root` privileges and that it not be confined by an AppArmor profile.

aa-unconfined must be run as `root` to retrieve the process executable link from the `/proc` file system. This program is susceptible to the following race conditions:

- An unlinked executable is mishandled
-

A process that dies between **`netstat(8)`** and further checks is mishandled

Note

This program lists processes using TCP and UDP only. In short, this program is unsuitable for forensics use and is provided only as an aid to profiling all network-accessible processes in the lab.

24.7.3.13 aa-notify

`aa-notify` is a handy utility that displays AppArmor notifications in your desktop environment. This is very convenient if you do not want to inspect the AppArmor log file, but rather let the desktop inform you about events that violate the policy. To enable AppArmor desktop notifications, run **`aa-notify`**:

```
sudo aa-notify -p -u USERNAME --display DISPLAY_NUMBER
```

where *`USERNAME`* is your user name under which you are logged in, and *`DISPLAY_NUMBER`* is the X Window display number you are currently using, such as `:0`. The process is run in the background, and shows a notification each time a deny event happens.

Tip

The active X Window display number is saved in the `$DISPLAY` variable, so you can use `--display $DISPLAY` to avoid finding out the current display number.

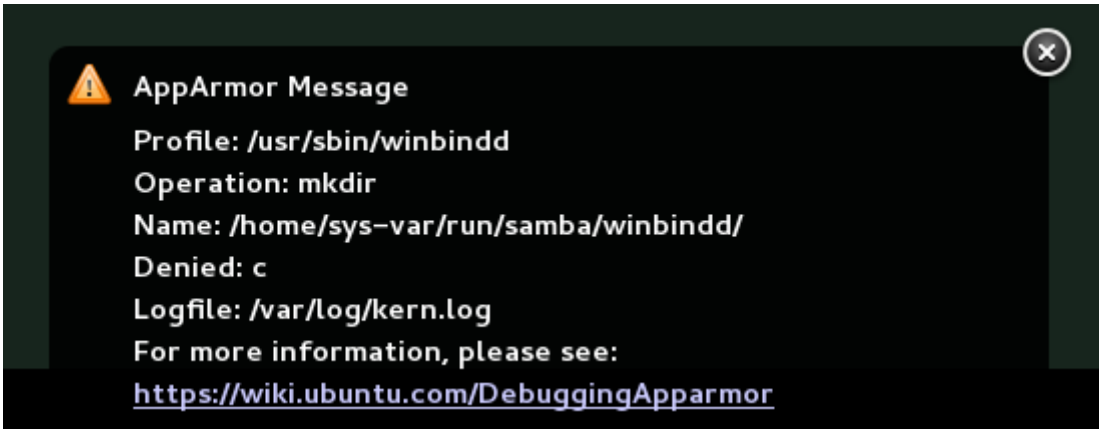

FIGURE 24.1: **aa-notify Message in GNOME**

With the `-s DAYS` option, you can also configure **aa-notify** to display a summary of notifications for the specified number of past days. For more information on **aa-notify**, see its man page **man 8 aa-notify**.

24.7.3.14 apparmor.vim

A syntax highlighting file for the vim text editor highlights various features of an AppArmor profile with colors. Using vim and the AppArmor syntax mode for vim, you can see the semantic implications of your profiles with color highlighting. Use vim to view and edit your profile by typing vim at a terminal window.

To enable the syntax coloring when you edit an AppArmor profile in vim, use the commands `:syntax on` then `:set syntax=apparmor`. To make sure vim recognizes the edited file type correctly as an AppArmor profile, add

```
# vim:ft=apparmor
```

at the end of the profile.

Tip

vim comes with AppArmor highlighting automatically enabled for files in /etc/apparmor.d/.

When you enable this feature, vim colors the lines of the profile for you:

Blue
: Comments

White
Ordinary read access lines

Brown
Capability statements and complain flags

Yellow
Lines that grant write access

Green
Lines that grant execute permission (either ix or px)

Red
Lines that grant unconfined access (ux)

Red background
Syntax errors that will not load properly into the AppArmor modules

Use the `apparmor.vim` and `vim` man pages and the `:help syntax` from within the vim editor for further vim help about syntax highlighting. The AppArmor syntax is stored in `/usr/share/vim/current/syntax/apparmor.vim`.

24.8 Important File Names and Directories

The following list contains the most important files and directories used by the AppArmor framework. If you intend to manage and troubleshoot your profiles manually, make sure that you know about these files and directories:

`/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 paths 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`
: Check this file to 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.