AppArmor | Ubuntu

ubuntu.com/server/docs/apparmor

Introduction

AppArmor

> **Note:**
> This documentation has moved to a new home! Please update your bookmarks to the new URL for the up-to-date version of this page.

AppArmor is an easy-to-use Linux Security Module implementation that restricts applications' capabilities and permissions with **profiles** that are set per-program. It provides mandatory access control (MAC) to supplement the more traditional UNIX model of discretionary access control (DAC).

In Ubuntu, AppArmor is installed and loaded by default – you can check this by running `aa-status`.

It uses **profiles** of an application to determine what files and permissions the application requires. Some packages will install their own profiles, and additional profiles can be found in the `apparmor-profiles` package.

Install AppArmor Profiles

To install the `apparmor-profiles` package from a terminal prompt:

```
sudo apt install apparmor-profiles
```

AppArmor profiles have two modes of operation:

- **Complaining/Learning**: profile violations are permitted and logged. This is useful for testing and developing new profiles.
- **Enforced/Confined**: enforces profile policy in addition to logging the violation.

Using AppArmor

The optional `apparmor-utils` package contains command-line utilities you can use to change the AppArmor operation mode, find the status of a profile, create new profiles, etc.

AppArmor profiles are located in the `/etc/apparmor.d` directory. It also stores **abstractions** that can simplify profile authoring, such as `abstractions/base` that allows many shared libraries, writing logs to the journal, many pseudo-devices, receiving signals from unconfined processes, and many more things.

Common commands

- **`apparmor_status`** is used to view the current status of AppArmor profiles:

  ```
  sudo apparmor_status
  ```

- **`aa-complain`** places a profile into `complain` mode:

  ```
  sudo aa-complain /path/to/bin
  ```

- **`aa-enforce`** places a profile into `enforce` mode:

  ```
  sudo aa-enforce /path/to/bin
  ```

- **`apparmor_parser`** is used to load a profile into the kernel. It can also be used to reload a currently-loaded profile using the `-r` option after modifying it to have the changes take effect.
  To reload a profile:

  ```
  sudo apparmor_parser -r /etc/apparmor.d/profile.name
  ```

- **`systemctl`** can be used to reload all profiles:

  ```
  sudo systemctl reload apparmor.service
  ```

Disabling or re-enabling a profile

The `/etc/apparmor.d/disable` directory can be used along with the `apparmor_parser -R` option to disable a profile:

```
sudo ln -s /etc/apparmor.d/profile.name /etc/apparmor.d/disable/
sudo apparmor_parser -R /etc/apparmor.d/profile.name
```

To re-enable a disabled profile, remove the symbolic link to the profile in `/etc/apparmor.d/disable/`, then load the profile using the `-a` option:

```
sudo rm /etc/apparmor.d/disable/profile.name
cat /etc/apparmor.d/profile.name | sudo apparmor_parser -a
```

AppArmor can be disabled, and the kernel module unloaded, by entering the following:

```
sudo systemctl stop apparmor.service
sudo systemctl disable apparmor.service
```

To re-enable AppArmor, enter:

```
sudo systemctl enable apparmor.service
sudo systemctl start apparmor.service
```

> **Note**:
> Replace `profile.name` with the name of the profile you want to manipulate. Also, replace `/path/to/bin/` with the actual executable file path. For example, for the `ping` command use `/bin/ping`.

Profiles

AppArmor profiles are simple text files located in `/etc/apparmor.d/`. The files are named after the full path to the executable they profile, replacing the "`/`" with "`.`".

For example `/etc/apparmor.d/bin.ping` is the AppArmor profile for the `/bin/ping` command.

There are two main type of rules used in profiles:

- **Path entries**, detailing which files an application can access in the file system.
- **Capability entries**, which determine what privileges a confined process is allowed to use.

As an example, take a look at `/etc/apparmor.d/bin.ping`:

```
#include <tunables/global>
/bin/ping flags=(complain) {
  #include <abstractions/base>
  #include <abstractions/consoles>
  #include <abstractions/nameservice>

  capability net_raw,
  capability setuid,
  network inet raw,

  /bin/ping mixr,
  /etc/modules.conf r,
}
```

Which can be broken down as follows:

- `#include <tunables/global>`: include statements from other files. This allows statements pertaining to multiple applications to be placed in a common file.
- `/bin/ping flags=(complain)`: path to the profiled program, also setting the mode to `complain`.
- `capability net_raw,`: allows the application access to the `CAP_NET_RAW Posix.1e` capability.
- `/bin/ping mixr,`: allows the application read and execute access to the file.

> **Note**:
> After editing a profile file the profile must be reloaded.

Create a Profile

- **Design a test plan**:
  Try to think about how the application should be exercised. The test plan should be divided into small test cases. Each test case should have a small description and list the steps to follow.

  Some standard test cases are:

  - Starting the program
  - Stopping the program
  - Reloading the program
  - Testing all the commands supported by the init script
- **Generate the new profile**:
  Use `aa-genprof` to generate a new profile. From a terminal:

  ```
  sudo aa-genprof executable
  ```

  For example:

  ```
  sudo aa-genprof slapd
  ```

- To get your new profile included in the `apparmor-profiles` package, file a bug in Launchpad against the AppArmor package:

  - Include your test plan and test cases
  - Attach your new profile to the bug

Updating profiles

When the program is misbehaving, audit messages are sent to the log files. The program `aa-logprof` can be used to scan log files for AppArmor audit messages, review them and update the profiles. From a terminal:

```
sudo aa-logprof
```

Further pre-existing profiles

The packages `apport-profiles` and `apparmor-profiles-extra` ship some experimental profiles for AppArmor security policies. Do not expect these profiles to work out-of-the-box, but they can give you a head start when trying to create a new profile by starting off with a base that already exists.

These profiles are not considered mature enough to be shipped in `enforce` mode by default. Therefore, they are shipped in `complain` mode so that users can test them, choose which are desired, and help improve them upstream if needed.

Some even more experimental profiles carried by the package are placed in `/usr/share/doc/apparmor-profiles/extras/`

Checking and debugging denies

You will see in `dmesg` (and any log that collects kernel messages) if you have hit a **deny**. It is worth knowing that this will cover any access that was denied `because it was not allowed`, but `explicit denies` will put no message in your logs at all.

Examples might look like:

```
[1521056.552037] audit: type=1400 audit(1571868402.378:24425): apparmor="DENIED"
operation="open" profile="/usr/sbin/cups-browsed"
name="/var/lib/libvirt/dnsmasq/" pid=1128 comm="cups-browsed" requested_mask="r"
denied_mask="r" fsuid=0 ouid=0
[1482106.651527] audit: type=1400 audit(1571829452.330:24323): apparmor="DENIED"
operation="sendmsg" profile="snap.lxd.lxc" pid=24115 comm="lxc" laddr=10.7.0.69
lport=48796 faddr=10.7.0.231 fport=445 family="inet" sock_type="stream"
protocol=6 requested_mask="send" denied_mask="send"
```

That follows a generic structure starting with a timestamp, an audit tag and the category `apparmor="DENIED"`. From the following fields you can derive what was going on and why it was failing.

In the examples above that would be:

First example:

- `operation`: `open` (program tried to open a file)
- `profile`: `/usr/sbin/cups-browsed` (you'll find `/etc/apparmor.d/usr.bin.cups-browsed`)
- `name`: `/var/lib/libvirt/dnsmasq` (what it wanted to access)
- `pid/comm`: the program that triggered the access
- `requested_mask/denied_mask/fsuid/ouid`: parameters of that open call

Second example:

- `operation`: `sendmsg` (program tried send via network)
- `profile`: `snap.lxd.lxc` (snaps are special, you'll find `/var/lib/snapd/apparmor/profiles/snap.lxd.lxc`)
- `pid/comm`: the program that triggered the access
- `laddr/lport/faddr/fport/family/sock_type/protocol`: parameters of the `sendmsg` call

That way you know in which profile and at what action you have to start if you consider either debugging or adapting the profiles.

Profile customisation

Profiles are meant to provide security and so can't be too permissive. But often, a very special setup would work with a profile if it wold *just allow this one extra access*. To handle that situation, there are three options:

- Modify the profile itself:
  Always works, but has the drawback that profiles are in `/etc` and considered `conffiles`. So after modification on a related package update you might get a `conffile` prompt. Worst case; depending on configuration, automatic updates might even override it and your custom rule is gone.
- Use tunables:
  - These provide variables that can be used in templates, for example if you want a custom `dir` considered as it would be a home directory. You could modify `/etc/apparmor.d/tunables/home`, which defines the base path rules used for home directories.
  - By design, these variables will only influence profiles that use them.
- Modify a local override:
  - To mitigate the drawbacks of above approaches, **local includes** were introduced, adding the ability to write arbitrary rules that not run into issues during upgrades that modify the packaged rule.
  - The files can be found in `/etc/apparmor.d/local/` and exist for the packages that are known to sometimes need slight tweaks for special setups.

Further reading

- See the AppArmor Administration Guide for advanced configuration options.
- For details using AppArmor with other Ubuntu releases see the AppArmor Community Wiki page.
- The OpenSUSE AppArmor page is another introduction to AppArmor.
- (https://wiki.debian.org/AppArmor) is another introduction and basic how-to for AppArmor.
- A great place to get involved with the Ubuntu Server community and to ask for AppArmor assistance is the `#ubuntu-server` IRC channel on Libera. The `#ubuntu-security` IRC channel may also be of use.

Previous Firewall Next Smart card authentication