AppArmor

AppArmor (https://apparmor.net/) is a **Mandatory Access Control** (MAC) system, implemented upon the **Linux Security Modules** (LSM).

Related articles

- **Security**
- **SELinux**
- **TOMOYO Linux**

AppArmor, like most other LSMs, supplements rather than replaces the default Discretionary Access Control (DAC). As such it is impossible to grant a process more privileges than it had in the first place.

Ubuntu, SUSE and a number of other distributions use it by default. RHEL (and its variants) use SELinux which requires good userspace integration to work properly. SELinux attaches labels to all files, processes and objects and is therefore very flexible. However configuring SELinux is considered to be very complicated and requires a supported filesystem. AppArmor on the other hand works using file paths and its configuration can be easily adapted.

AppArmor proactively protects the operating system and applications from external or internal threats and even zero-day attacks by enforcing a specific rule set on a per application basis. Security policies completely define what system resources individual applications can access, and with what privileges. Access is denied by default if no profile says otherwise. A few default policies are included with AppArmor and using a combination of advanced static analysis and learning-based tools, AppArmor policies for even very complex applications can be deployed successfully in a matter of hours.

Every breach of policy triggers a message in the system log, and AppArmor can be configured to notify users with real-time violation warnings popping up on the desktop.

1 Installation

AppArmor is available in all **officially supported kernels**.

Install `apparmor (https://archlinux.org/packages/?name=apparmor)` for userspace tools and libraries to control AppArmor. To load all AppArmor profiles on startup, **enable** `apparmor.service`.

To enable AppArmor as default security model on every boot, set the following **kernel parameter**:

```
lsm=landlock,lockdown,yama,integrity,apparmor,bpf
```

Note: The `lsm=` kernel parameter sets the initialization order of Linux security modules. The kernel's configured `lsm=` value can be found with `zgrep CONFIG_LSM= /proc/config.gz` and the current value with `cat /sys/kernel/security/lsm`.

- Make sure that `apparmor` is the first "major" module in the list.**[1] (https://docs.kernel.org/admin-guide/LSM/index.html)** Examples of valid values and their order can be found in **security/Kconfig (https://git.kernel.org/pub/scm/linux/kernel/git/torvalds/linux.git/tree/security/Kconfig)**.

- `capability` should be omitted from `lsm=` as it will always get included automatically.

1.1 Custom kernel

When **compiling the kernel**, it is required to set at least the following options:

```
CONFIG_SECURITY_APPARMOR=y
CONFIG_AUDIT=y
```

To enable the AppArmor Linux security model by default and omit the need to set kernel parameters, additionally set the `CONFIG_LSM` option and specify `apparmor` as the first "major" module in the list:

```
CONFIG_LSM="landlock,lockdown,yama,integrity,apparmor,bpf"
```

2 Usage

2.1 Display current status

To test if AppArmor has been correctly enabled:

```
$ aa-enabled
```

```
Yes
```

To display the current loaded status use **aa-status(8) (https://man.archlinux.org/man/aa-status.8)**:

```
# aa-status
```

```
apparmor module is loaded.
44 profiles are loaded.
44 profiles are in enforce mode.
 ...
0 profiles are in complain mode.
0 processes have profiles defined.
0 processes are in enforce mode.
0 processes are in complain mode.
0 processes are unconfined but have a profile defined.
```

In *complain mode*, policy violations of profiles are allowed but are logged. Complain mode is used for testing new profiles. Note that deny rules of profiles are enforced/blocked even in complain mode.

In *enforce mode*, policy violations of profiles are blocked and logged.

2.2 Parsing profiles

To load (enforce or complain), unload, reload, cache and stat profiles use `apparmor_parser`. The default action (`-a`) is to load a new profile in enforce mode, loading it in complain mode is possible using the `-C` switch, in order to overwrite an existing profile use the `-r` option and to

remove a profile use `-R`. Each action may also apply to multiple profiles. Refer to apparmor_parser(8) (https://man.archlinux.org/man/apparmor_parser.8) man page for more information.

2.3 Disabling loading

Disable AppArmor by unloading all profiles for the current session:

```
# aa-teardown
```

To prevent AppArmor profiles from loading at the next boot **disable** `apparmor.service`. To prevent the kernel from loading AppArmor, remove the `lsm=` **kernel parameter** that was added when **setting up AppArmor**.

3 Configuration

3.1 Auditing and generating profiles

To create new profiles the **Audit framework** should be running. This is because Arch Linux adopted **systemd** and does not do kernel logging to file by default. AppArmor can grab kernel audit logs from the userspace auditd daemon, allowing you to build a profile.

> **Note:** Apparmor audit messages uses **AVC (https://docs.redhat.com/en/documentation/red_hat_enterprise_linux/6/html/security_guide/sec-Audit_Record_Types#sec-Audit_Record_Types)** record type, and can break audit log parsing when searching logs with **ausearch (https://man.archlinux.org/man/ausearch.8)**. Check the bug reports **[2] (https://bugs.launchpad.net/ubuntu/+source/audit/+bug/1117804)**, **[3] (https://bugs.archlinux.org/task/60870)**, **[4] (https://github.com/linux-audit/audit-userspace/issues/351)**

New AppArmor profiles can be created by utilizing aa-genprof(8) (https://man.archlinux.org/man/aa-genprof.8) or aa-autodep(8) (https://man.archlinux.org/man/aa-autodep.8). The profile is first created in *complain mode*: in this mode policy violations are only reported but not enforced. The rules are interactively created by the aa-logprof(8) (https://man.archlinux.org/man/aa-logprof.8) tool available in apparmor (https://archlinux.org/packages/?name=apparmor) package. Finally the profile should be set into *enforce mode* with aa-enforce(8) (https://man.archlinux.org/man/aa-enforce.8). In this mode the policy defined by the rules in the respecting profile are enforced. If necessary, additional rules can be added by repeatedly executing aa-logprof(8) (https://man.archlinux.org/man/aa-logprof.8), or the profile can be set back to complain mode with aa-complain(8) (https://man.archlinux.org/man/aa-complain.8). Detailed guide about using those tools is available at **AppArmor wiki - Profiling with tools (https://gitlab.com/apparmor/apparmor/wikis/Profiling_with_tools)**.

Note that aa-logprof(8) (https://man.archlinux.org/man/aa-logprof.8) also offers *deny* rules which are actually not strictly necessary as according to the basic AppArmor logic everything is forbidden which is not explicitly allowed by a rule. However, *deny* rules serve two purposes:

1. *deny* rules take precedence over *allow* rules. They are often used in many **abstractions (https://man.archlinux.org/man/extra/apparmor/apparmor.d.5.en##include_mechanism)**

located in `/etc/apparmor.d/abstractions` in order to block any access to important folders/files. This makes sure that inadvertently created allow rules do not make a profile too permissive.
2. *deny* rules silence logging and make subsequent runs of *aa-logprof* less noisy. It is important to keep in mind that *deny* rules are enforced also in *complain mode* - hence, if an application does not work properly even in complain mode it should be checked if a deny rule in the profile or in one of the included abstractions is the culprit.

Alternatively profiles may be also created manually, see guide available at **AppArmor wiki - Profiling by hand (https://gitlab.com/apparmor/apparmor/wikis/Profiling_by_hand)**.

In addition to the default profiles in `/etc/apparmor.d/`, there are more predefined profiles in `/usr/share/apparmor/extra-profiles/`. Note that those are not necessarily deemed production-ready, so manual intervention or usage of `aa-logprof(8)` (https://man.archlinux.org/man/aa-logprof.8) may be required.

Another set of AppArmor profiles can be found in the **apparmor.d (https://github.com/roddhjav/apparmor.d)** project, however, as of writing, the project is not currently stable.

3.2 Understanding profiles

Profiles are human readable text files residing under `/etc/apparmor.d/` describing how binaries should be treated when executed. A basic profile looks similar to this:

/etc/apparmor.d/usr.bin.test

```
#include <tunables/global>

profile test /usr/lib/test/test_binary {
    #include <abstractions/base>

    # Main libraries and plugins
    /usr/share/TEST/** r,
    /usr/lib/TEST/** rm,

    # Configuration files and logs
    @{HOME}/.config/ r,
    @{HOME}/.config/TEST/** rw,
}
```

Strings preceded by a `@` symbol are variables defined by abstractions (`/etc/apparmor.d/abstractions/`), tunables (`/etc/apparmor.d/tunables/`) or by the profile itself. `#include` includes other profile-files directly. Paths followed by a set of characters are **access permissions (https://man.archlinux.org/man/extra/apparmor/apparmor.d.5.en#Access_Modes)**. Pattern matching is done using **AppArmor's globbing syntax (https://man.archlinux.org/man/extra/apparmor/apparmor.d.5.en#Globbing)**.

Most common use cases are covered by the following statements:

- `r` — read: read data
- `w` — write: create, delete, write to a file and extend it
- `m` — memory map executable: memory map a file executable
- `x` — execute: execute file; needs to be preceded by a **qualifier (https://gitlab.com/apparmor/apparmor/wikis/AppArmor_Core_Policy_Reference#execute-rules)**

Remember that those permission do not allow binaries to exceed the permission dictated by the Discretionary Access Control (DAC).

This is merely a short overview, for a more detailed guide be sure to have a look at the `apparmor.d(5)` (https://man.archlinux.org/man/apparmor.d.5) man page and documentation (https://gitlab.com/apparmor/apparmor/wikis/AppArmor_Core_Policy_Reference).

4 Tips and tricks

4.1 Get desktop notification on DENIED actions

The notification daemon displays desktop notifications whenever AppArmor denies a program access. To automatically start *aa-notify* daemon on login follow below steps:

Install the Audit framework and enable and start the userspace Linux Audit daemon. Allow your desktop user to read audit logs in `/var/log/audit` by adding it to `audit` user group:

```
# groupadd -r audit
# gpasswd -a user audit
```

Add `audit` group to `auditd.conf` :

```
/etc/audit/auditd.conf
```

```
log_group = audit
```

Tip: You may use other already existing system groups like `wheel` or `adm` .

Install `python-notify2` (https://archlinux.org/packages/?name=python-notify2) and `python-psutil` (https://archlinux.org/packages/?name=python-psutil).

Create a desktop launcher with the below content:

```
~/.config/autostart/apparmor-notify.desktop
```

```
[Desktop Entry]
Type=Application
Name=AppArmor Notify
Comment=Receive on screen notifications of AppArmor denials
TryExec=aa-notify
Exec=aa-notify -p -s 1 -w 60 -f /var/log/audit/audit.log
StartupNotify=false
NoDisplay=true
```

Reboot and check if the `aa-notify` process is running:

```
$ pgrep -ax aa-notify
```

Note: Depending on your specific system configuration there could be A LOT of notifications displayed.

For more information, see aa-notify(8) (https://man.archlinux.org/man/aa-notify.8).

4.2 Speed-up AppArmor start by caching profiles

Since AppArmor has to translate the configured profiles into a binary format it may significantly increase the boot time. You can check current AppArmor startup time with:

```
$ systemd-analyze blame | grep apparmor
```

To enable caching AppArmor profiles, uncomment:

/etc/apparmor/parser.conf

```
## Turn creating/updating of the cache on by default
write-cache
```

To change default cache location add:

/etc/apparmor/parser.conf

```
cache-loc=/path/to/location
```

Note: Since 2.13.1 default cache location is `/var/cache/apparmor/` , previously it was `/etc/apparmor.d/cache.d/` .

Reboot and check AppArmor startup time again to see improvement:

```
$ systemd-analyze blame | grep apparmor
```

5 Troubleshooting

5.1 Failing to start Samba SMB/CIFS server

See **Samba#Permission issues on AppArmor**.

5.2 No events caught by aa-logprof

Audit framework logs may have the special character `0x1d` **[5] (https://github.com/linux-audit/audit-userspace/issues/3)**. There is a **AppArmor bug report (https://gitlab.com/apparmor/apparmor/-/issues/271)**, but a workaround is to run:

```
# aa-logprof -f <(sed 's/\x1d.*//' < /var/log/audit/audit.log)
```

5.3 Login impossible after upgrading to AppArmor v4

In rare cases, after upgrading to AppArmor version 4, **it becomes impossible to log into any system account (https://gitlab.archlinux.org/archlinux/packaging/packages/apparmor/-/issues/2)**.

The **system journal** might contain errors like these:

```
unix_chkpwd[1612]: check pass; user unknown
unix_chkpwd[1612]: password check failed for user (john)
gdm-password][1574]: pam_unix(gdm-password:auth): authentication failure; logname= uid=0 euid=0 tty=/dev/tty1 ruse
r= rhost=  user=john
kernel: audit: type=1400 audit(1730844640.468:171): apparmor="DENIED" operation="capable" class="cap" profile="uni
x-chkpwd" pid=1612 comm="unix_chkpwd" capability=2  capname="dac_read_search"
kernel: audit: type=1400 audit(1730844640.468:172): apparmor="DENIED" operation="capable" class="cap" profile="uni
x-chkpwd" pid=1612 comm="unix_chkpwd" capability=1  capname="dac_override"
```

This might be caused by `/etc/shadow` and/or `/etc/gshadow` not being readable by the `root` user (i.e. the permission bits of those files might be entirely unset). Thus, a possible solution would be to:

1. Reboot, and **disable AppArmor** either by editing the appropriate **boot loader** entry during boot or by using fallback boot entry, which doesn't have AppArmor enabled.
2. Log in as `root` , and set the correct file permissions:
   `chmod 600 /etc/shadow /etc/gshadow`
3. Reboot once again.

5.4 Differences between Linux distributions

Information you find often is about AppArmor on Ubuntu, which can be confusing, since Ubuntu carries a lot of kernel patches regarding AppArmor. Other distributions may also carry their own kernel patches, while Arch Linux uses a close-to-mainline kernel.

For example, while **apparmor.d(5) (https://man.archlinux.org/man/apparmor.d.5)** already documents `dbus` rules, they require support from AppArmor userspace tools, kernel and D-Bus daemon**[6] (https://security.stackexchange.com/questions/200496/apparmor-how-to-enable-dbus-feature-of-apparmor-dbus-mediation-in-the-linu)**. The corresponding kernel patch**[7] (https://git.launchpad.net/~ubuntu-kernel/ubuntu/+source/linux/+git/oracular/commit/?id=938de00f4fc19da963a67bfca3526f493c6460e4)** is applied by Ubuntu, but not included in mainline Linux and **official Arch kernels**. (Support also varies by **D-Bus implementation**.)

AppArmor-specific kernel patches applied by Ubuntu can be found at (replace *oracular* with the codename of the Ubuntu version you are interested in):

```
https://git.launchpad.net/~ubuntu-kernel/ubuntu/+source/linux/+git/oracular/log/?qt=grep&q=UBUNTU%3A+SAUCE%3A+appa
rmor
```

The ABI versions supported by the userland tools can be found in `/etc/apparmor.d/abi/` . The ABI supported by the currently running kernel can be shown with:

```
$ aa-features-abi --extract
```

6 See also

- **Wikipedia:AppArmor**
- **AppArmor wiki (https://gitlab.com/apparmor/apparmor/wikis/home)**
- **AppArmor Core Policy Reference (https://gitlab.com/apparmor/apparmor/wikis/AppArmor_Core_Policy_Reference)** — Detailed description of available options in a profile
- **Ubuntu Tutorial (https://ubuntuforums.org/showthread.php?t=1008906)** — General overview of available utilities and profile creation

- **Ubuntu Wiki (https://help.ubuntu.com/community/AppArmor)** — Basic command overview
- **AppArmor Versions (https://gitlab.com/apparmor/apparmor/wikis/AppArmor_versions)** — Version overview and links to the respective release notes
- **Kernel Interfaces (https://gitlab.com/apparmor/apparmor/wikis/Kernel_interfaces)** — Low level interfaces to the AppArmor kernel module
- **wikipedia:Linux Security Modules** — Linux kernel module on which basis AppArmor is build upon
- **AppArmor in openSUSE Security Guide (https://doc.opensuse.org/documentation/leap/security/single-html/book-security/index.html#part-apparmor)**
