# Package ‘RAppArmor’

January 31, 2020

**Type** Package

**Title** Bindings to AppArmor and Security Related Linux Tools

**Version** 3.2.1

**Description** Bindings to kernel methods for enforcing security restrictions.
AppArmor can apply mandatory access control (MAC) policies on a given task
(process) via security profiles with detailed ACL definitions. In addition
this package implements bindings for setting process resource limits (rlimit),
uid, gid, affinity and priority. The high level R function 'eval.secure'
builds on these methods to perform dynamic sandboxing: it evaluates a single
R expression within a temporary fork which acts as a sandbox by enforcing
fine grained restrictions without affecting the main R process. A portable
version of this function is now available in the 'unix' package.

**License** Apache License 2.0

**URL** `http://www.jstatsoft.org/v55/i07/` (paper),
`http://github.com/jeroen/RAppArmor#readme` (devel)

**BugReports** `http://github.com/jeroen/RAppArmor/issues`

**OS_type** unix

**SystemRequirements** linux (>= 3.0), libapparmor-dev

**VignetteBuilder** R.rsp

**Suggests** testthat, R.rsp

**Depends** unix (>= 1.4)

**RoxygenNote** 6.1.1

**Language** en-US

**NeedsCompilation** yes

**Author** Jeroen Ooms [aut, cre] (<https://orcid.org/0000-0002-4035-0289>)

**Maintainer** Jeroen Ooms <`jeroen@berkeley.edu`>

**Repository** CRAN

**Date/Publication** 2020-01-31 13:10:05 UTC

# R topics documented:

---

## affinity *Process Affinity*

---

### Description

Get/set the process's CPU affinity mask. The affinity mask binds the process to specific core(s) within the machine. Not supported on all systems, `has_affinity()` shows if this is available.

### Usage

```
setaffinity(cpus = 1:ncores())

getaffinity_count()

getaffinity()

has_affinity()

ncores()
```

### Arguments

| | |
|---|---|
| `cpus` | Which cpu cores to bind to: vector of integers between 1 and `ncores()` |

### Details

Setting a process affinity allows for restricting the process to only use certain cores in the machine. The cores are indexed by the operating system as 1 to `ncores()`. Calling `setaffinity()` with no arguments resets the process to use any of the available cores.

Note that setaffinity is different from setting r_limit values in the sense that it is not a one-way process. An unprivileged user can change the process affinity to any value. In order to 'lock' an affinity value, one would have to manipulate Linux capability value for CAP_SYS_NICE.

### References

SCHED_SETAFFINITY(2)

## Examples

```
## Not run:
# Current affinity
ncores()
getaffinity()
getaffinity_count()

# Restrict process to core number 1.
setaffinity(1)
getaffinity()

# Reset
setaffinity()
getaffinity()

## End(Not run)
```

---

`apparmor` *Change hats*

---

## Description

A hat is a subprofile which name starts with a '^'. The difference between hats and profiles is that one can escape (revert) from the hat using the token. Hence this provides more limited security than a profile.

Note that in order for this function to do its work, it needs read access to the attributes of the current process. If aa_getcon fails with a permission denied error, it might actually mean that the current process is being confined with a very restrictive profile.

## Usage

```
aa_change_hat(subprofile, magic_token)

aa_revert_hat(magic_token)

aa_change_profile(profile)

aa_find_mountpoint()

aa_getcon()

aa_is_enabled()

aa_is_compiled()
```

### Arguments

| | |
|---|---|
| `subprofile` | character string identifying the subprofile (hat) name (without the "^") |
| `magic_token` | a number that will be the key to revert out of the hat. |
| `profile` | character string with the name of the profile. |

### Examples

```
## Not run:
aa_change_profile("testprofile");
aa_getcon();
test <- read.table("/etc/group");
aa_change_hat("testhat", 13337);
aa_getcon();
test <- read.table("/etc/group");
aa_revert_hat(13337);
test <- read.table("/etc/group");

## End(Not run)
 ## Not run:
test <- read.table("/etc/passwd");
aa_change_profile("testprofile");
aa_getcon();
test <- read.table("/etc/passwd");

## End(Not run)
```

## `sandboxing` *Sandboxing*

### Description

This function has been superseded by the unix::eval_safe function.

### Usage

```
eval.secure(...)
```

### Arguments

| | |
|---|---|
| `...` | arguments passed to unix::eval_safe |

## `unittests` *RAppArmor unit tests*

### Description

This function loads the 'testthat' package and runs a number of unit tests for RAppArmor. Note that the tests assume that the main process is unconfined. Try running it both as root and as a regular user to cover both cases.

### Usage

```
unittests()
```

### Details

Occasionally, one or two tests might fail due to random fluctuations in available memory, cpu, etc. If this happens, try running the tests again, possibly with less other programs running in the background.

# Index