Package 'RAppArmor'

July 21, 2012

Type Package

Title RAppArmor

Version 0.5.0

Date 2012-07-02

Author Jeroen Ooms

Maintainer Jeroen Ooms <jeroen.ooms@stat.ucla.edu>

Description R interfaces to Linux and AppArmor security stuff.

License AGPL

R topics documented:

aa_change_hat 1
aa_change_profile 2
aa_find_mountpoint 2
aa_getcon 3
aa_is_enabled 3
eval.secure 4
getpriority 6
rlimit_as 6
rlimit_core 7
rlimit_cpu 8
rlimit_data 10
rlimit_fsize 11
rlimit_memlock 12
rlimit_msgqueue 14
rlimit_nice 15
rlimit_nofile 16
rlimit_nproc 17
rlimit_rtprio 19
rlimit_rttime 20
rlimit_sigpending 21

rlimit_stack 22
setgid 24
setpriority 24
setuid 24
userinfo 25

`aa_change_hat` *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.

Usage

```
aa_change_hat(subprofile, magic_token)
```

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.

Examples

```
## Not run: aa_change_profile("myprofile");
read.table("/etc/group");
aa_change_hat("testhat", 13337);
read.table("/etc/group");
aa_revert_hat(13337);
read.table("/etc/group");

## End(Not run)
```

`aa_change_profile` *Change profiles*

Description

This function changes the current R process to an AppArmor profile. Note that this generally is a one way process: most profiles explicitly prevent switching into another profile, otherwise it would defeat the purpose.

Usage

```
aa_change_profile(profile)
```

Arguments

profile	character string with the name of the profile.

Examples

```
## Not run: read.table("/etc/passwd");
aa_change_profile("myprofile");
read.table("/etc/passwd");

## End(Not run)
```

---

aa_find_mountpoint *Find the apparmor mountpoint*

Description

Find the apparmor mountpoint

Usage

```
aa_find_mountpoint()
```

Value

location of mountpoint

---

aa_getcon *Get AppArmor confinement context for the current task*

Description

We can use this function to see if there is an AppArmor profile associated with the current process, and in which mode it current is set (enforce, complain, disable).

Usage

```
aa_getcon()
```

Details

Note that in order for this function to do its work, it needs read access to the attributes of the current process. Hence if a profile is being enforced that is overly strict, this confinement lookup will fail as well :-)

Value

list with con and mode.

aa_is_enabled *Check if AppArmor is Enabled*

Description

Check if AppArmor is Enabled

Usage

```
aa_is_enabled()
```

Value

TRUE or FALSE

eval.secure *Secure evaluation*

Description

Evaluate in a sandboxed environment.

Usage

```
eval.secure(..., uid, gid, priority, profile,
  timeout = 60, silent = FALSE, RLIMIT_AS, RLIMIT_CORE,
  RLIMIT_CPU, RLIMIT_DATA, RLIMIT_FSIZE, RLIMIT_MEMLOCK,
  RLIMIT_MSGQUEUE, RLIMIT_NICE, RLIMIT_NOFILE,
  RLIMIT_NPROC, RLIMIT_RTPRIO, RLIMIT_RTTIME,
  RLIMIT_SIGPENDING, RLIMIT_STACK)
```

Arguments

`...`	arguments passed on to eval(...)
`uid`	integer or name of linux user.
`gid`	integer or name of linux group.
`priority`	priority. Value between -20 and 20.
`profile`	AppArmor security profile. Has to be preloaded by Linux.
`timeout`	timeout in seconds.
`silent`	suppress output on stdout. See mcparallel().
`RLIMIT_AS`	hard limit passed on to rlimit_as()
`RLIMIT_CORE`	hard limit passed on to rlimit_core()
`RLIMIT_CPU`	hard limit passed on to rlimit_cpu()

RLIMIT_DATA hard limit passed on to rlimit_data()

RLIMIT_FSIZE hard limit passed on to rlimit_fsize()

RLIMIT_MEMLOCK
: hard limit passed on to rlimit_memlock()

RLIMIT_MSGQUEUE
: hard limit passed on to rlimit_msgqueue()

RLIMIT_NICE hard limit passed on to rlimit_nice()

RLIMIT_NOFILE
: hard limit passed on to rlimit_nofile()

RLIMIT_NPROC hard limit passed on to rlimit_nproc()

RLIMIT_RTPRIO
: hard limit passed on to rlimit_rtprio()

RLIMIT_RTTIME
: hard limit passed on to rlimit_rttime()

RLIMIT_SIGPENDING
: hard limit passed on to rlimit_sigpending()

RLIMIT_STACK hard limit passed on to rlimit_stack()

Details

This function creates a fork, and then sets any rlimits, uid, gid, priority, apparmor profile where specified, and then evaluates the expression inside the fork. After evaluation returns, the fork is killed. If the timeout is reached the fork is also killed and an error is thrown.

Evaluation of an expression through secure.eval should never have any side effects on the current R session. This also means that if the code does e.g. assignments to the global environment, sets options(), these will get lost, as we explicitly want to prevent this. However, if the expression saves any files (where allowed by apparmor), these will still be available after the evaluation finishes.

Note that if the initial process does not have superuser rights, rlimits can only be decreased and setuid/setgid might not work. In this case, specifying an RLIMIT higher than the current value will result in an error. Some of the rlimits can also be specified inside of the apparmor profile. When a rlimit is set both in the profile and through R, the more restrictive one will be effective.

Examples

```
## Not run:
## Restricting file access ##
eval.secure(list.files("/"))
eval.secure(list.files("/"), profile="r-base")

eval.secure(system("ls /", intern=TRUE))
eval.secure(system("ls /", intern=TRUE), profile="r-base")

## Limiting CPU time ##
cputest <- function(){
A <- matrix(rnorm(1e7), 1e3);
B <- svd(A);
}
```

```
## setTimeLimit doesn't always work:
setTimeLimit(5);
cputest();
setTimeLimit();

#timeout does work:
eval.secure(cputest(), timeout=5)

## Limiting memory ##
A <- matrix(rnorm(1e8), 1e4);
B <- eval.secure(matrix(rnorm(1e8), 1e4), RLIMIT_AS = 100*1024*1024)

## Limiting procs ##
forkbomb <- function(){
repeat{
parallel::mcparallel(forkbomb());
}
}

## Forkbomb is mitigated ##
eval.secure(forkbomb(), RLIMIT_NPROC=10)

## End(Not run)
```

`getpriority` *Get process priority*

Description

Read the priority value of the current process. Should be between -20 and 20.

Usage

```
getpriority()
```

`rlimit_as` *Limit virtual memory*

Description

Limits the maximum size of the process's virtual memory (address space) in bytes.

Usage

```
rlimit_as(hardlim, softlim = hardlim, pid = 0)
```

Arguments

softlim	soft limit in bytes.
hardlim	hard limit in bytes
pid	id of the target process.

Details

The maximum size of the process's virtual memory (address space) in bytes. This limit affects calls to brk(2), mmap(2) and mremap(2), which fail with the error ENOMEM upon exceeding this limit. Also automatic stack expansion will fail (and generate a SIGSEGV that kills the process if no alternate stack has been made available via sigaltstack(2)). Since the value is a long, on machines with a 32-bit long either this limit is at most 2 GiB, or this resource is unlimited.

See Also

Other rlimit: `rlimit_core`, `rlimit_cpu`, `rlimit_data`, `rlimit_fsize`, `rlimit_memlock`, `rlimit_msgqueue`, `rlimit_nice`, `rlimit_nofile`, `rlimit_nproc`, `rlimit_rtprio`, `rlimit_rttime`, `rlimit_sigpending`, `rlimit_stack`

Examples

```
#load lib
library(RAppArmor)

#current limit
rlimit_as();

#set hard limit
rlimit_as(1e9);

#set separate hard and soft limit.
rlimit_as(1e9, 1e8);

#soft limits can be elevated
rlimit_as(soft = 1e7);
rlimit_as(soft = 1e9);

#set other limits
rlimit_core(1e9);
rlimit_data(1e9);
rlimit_fsize(1e9);
rlimit_memlock(10000);
rlimit_msgqueue(1e5);
rlimit_nofile(10);
rlimit_nproc(100);
rlimit_rttime(1e9);
rlimit_sigpending(1e4);
rlimit_stack(1000);
rlimit_cpu(10);
```

`rlimit_core` *Limit core size.*

Description

Maximum size of core file. When 0 no core dump files are created. When nonzero, larger dumps are truncated to this size.

Usage

```
rlimit_core(hardlim, softlim = hardlim, pid = 0)
```

Arguments

`hardlim`	size
`softlim`	size
`pid`	id of the target process

See Also

Other rlimit: `rlimit_as`, `rlimit_cpu`, `rlimit_data`, `rlimit_fsize`, `rlimit_memlock`, `rlimit_msgqueue`, `rlimit_nice`, `rlimit_nofile`, `rlimit_nproc`, `rlimit_rtprio`, `rlimit_rttime`, `rlimit_sigpending`, `rlimit_stack`

Examples

```
#load lib
library(RAppArmor)

#current limit
rlimit_as();

#set hard limit
rlimit_as(1e9);

#set separate hard and soft limit.
rlimit_as(1e9, 1e8);

#soft limits can be elevated
rlimit_as(soft = 1e7);
rlimit_as(soft = 1e9);

#set other limits
rlimit_core(1e9);
rlimit_data(1e9);
rlimit_fsize(1e9);
rlimit_memlock(10000);
rlimit_msgqueue(1e5);
rlimit_nofile(10);
```

```
rlimit_nproc(100);
rlimit_rttime(1e9);
rlimit_sigpending(1e4);
rlimit_stack(1000);
rlimit_cpu(10);
```

`rlimit_cpu` *Limit CPU time*

Description

CPU time limit in seconds. When the process reaches the soft limit, it is sent a SIGXCPU signal.

Usage

```
rlimit_cpu(hardlim, softlim = hardlim, pid = 0)
```

Arguments

`hardlim`	cpu time in seconds
`softlim`	cpu time in seconds
`pid`	id of the target process

Details

CPU time limit in seconds. When the process reaches the soft limit, it is sent a SIGXCPU signal. The default action for this signal is to terminate the process. However, the signal can be caught, and the handler can return control to the main program. If the process continues to consume CPU time, it will be sent SIGXCPU once per second until the hard limit is reached, at which time it is sent SIGKILL. (This latter point describes Linux behavior. Implementations vary in how they treat rocesses which continue to consume CPU time after reaching the soft limit. Portable applications that need to catch this signal should perform an orderly termination upon first receipt of SIGXCPU.)

See Also

Other rlimit: `rlimit_as`, `rlimit_core`, `rlimit_data`, `rlimit_fsize`, `rlimit_memlock`, `rlimit_msgqueue`, `rlimit_nice`, `rlimit_nofile`, `rlimit_nproc`, `rlimit_rtprio`, `rlimit_rttime`, `rlimit_sigpending`, `rlimit_stack`

Examples

```
#load lib
library(RAppArmor)

#current limit
rlimit_as();

#set hard limit
```

```
rlimit_as(1e9);

#set separate hard and soft limit.
rlimit_as(1e9, 1e8);

#soft limits can be elevated
rlimit_as(soft = 1e7);
rlimit_as(soft = 1e9);

#set other limits
rlimit_core(1e9);
rlimit_data(1e9);
rlimit_fsize(1e9);
rlimit_memlock(10000);
rlimit_msgqueue(1e5);
rlimit_nofile(10);
rlimit_nproc(100);
rlimit_rttime(1e9);
rlimit_sigpending(1e4);
rlimit_stack(1000);
rlimit_cpu(10);
```

`rlimit_data` *Limit data segment*

Description

The maximum size of the process's data segment (initialized data, uninitialized data, and heap).

Usage

```
rlimit_data(hardlim, softlim = hardlim, pid = 0)
```

Arguments

`hardlim`	size
`softlim`	size
`pid`	id of the target process

Details

The maximum size of the process's data segment (initialized data, uninitialized data, and heap). This limit affects calls to brk(2) and sbrk(2), which fail with the error ENOMEM upon encountering the soft limit of this resource.

See Also

Other rlimit: `rlimit_as`, `rlimit_core`, `rlimit_cpu`, `rlimit_fsize`, `rlimit_memlock`, `rlimit_msgqueue`, `rlimit_nice`, `rlimit_nofile`, `rlimit_nproc`, `rlimit_rtprio`, `rlimit_rttime`, `rlimit_sigpending`, `rlimit_stack`

Examples

```
#load lib
library(RAppArmor)

#current limit
rlimit_as();

#set hard limit
rlimit_as(1e9);

#set separate hard and soft limit.
rlimit_as(1e9, 1e8);

#soft limits can be elevated
rlimit_as(soft = 1e7);
rlimit_as(soft = 1e9);

#set other limits
rlimit_core(1e9);
rlimit_data(1e9);
rlimit_fsize(1e9);
rlimit_memlock(10000);
rlimit_msgqueue(1e5);
rlimit_nofile(10);
rlimit_nproc(100);
rlimit_rttime(1e9);
rlimit_sigpending(1e4);
rlimit_stack(1000);
rlimit_cpu(10);
```

---

rlimit_fsize *Limit size of files*

Description

The maximum size of files that the process may create.

Usage

```
rlimit_fsize(hardlim, softlim = hardlim, pid = 0)
```

Arguments

hardlim	size
softlim	size
pid	id of the target process

Details

The maximum size of files that the process may create. Attempts to extend a file beyond this limit result in delivery of a SIGXFSZ signal. By default, this signal terminates a process, but a process can catch this signal instead, in which case the relevant system call (e.g., write(2), truncate(2)) fails with the error EFBIG.

See Also

Other rlimit: `rlimit_as`, `rlimit_core`, `rlimit_cpu`, `rlimit_data`, `rlimit_memlock`, `rlimit_msgqueue`, `rlimit_nice`, `rlimit_nofile`, `rlimit_nproc`, `rlimit_rtprio`, `rlimit_rttime`, `rlimit_sigpending`, `rlimit_stack`

Examples

```
#load lib
library(RAppArmor)

#current limit
rlimit_as();

#set hard limit
rlimit_as(1e9);

#set separate hard and soft limit.
rlimit_as(1e9, 1e8);

#soft limits can be elevated
rlimit_as(soft = 1e7);
rlimit_as(soft = 1e9);

#set other limits
rlimit_core(1e9);
rlimit_data(1e9);
rlimit_fsize(1e9);
rlimit_memlock(10000);
rlimit_msgqueue(1e5);
rlimit_nofile(10);
rlimit_nproc(100);
rlimit_rttime(1e9);
rlimit_sigpending(1e4);
rlimit_stack(1000);
rlimit_cpu(10);
```

`rlimit_memlock` *Limit locked memory*

Description

The maximum number of bytes of memory that may be locked into RAM.

Usage

```
rlimit_memlock(hardlim, softlim = hardlim, pid = 0)
```

Arguments

hardlim	number of bytes
softlim	number of bytes
pid	id of the target process

Details

The maximum number of bytes of memory that may be locked into RAM. In effect this limit is rounded down to the nearest multiple of the system page size. This limit affects mlock(2) and mlockall(2) and the mmap(2) MAP_LOCKED operation. Since Linux 2.6.9 it also affects the shmctl(2) SHM_LOCK operation, where it sets a maximum on the total bytes in shared memory segments (see shmget(2)) that may be locked by the real user ID of the calling process. The shmctl(2) SHM_LOCK locks are accounted for separately from the per-process memory locks established by mlock(2), mlockall(2), and mmap(2) MAP_LOCKED; a process can lock bytes up to this limit in each of these two categories. In Linux kernels before 2.6.9, this limit controlled the amount of memory that could be locked by a privileged process. Since Linux 2.6.9, no limits are placed on the amount of memory that a privileged process may lock, and this limit instead governs the amount of memory that an unprivileged process may lock.

See Also

Other rlimit: `rlimit_as`, `rlimit_core`, `rlimit_cpu`, `rlimit_data`, `rlimit_fsize`, `rlimit_msgqueue`, `rlimit_nice`, `rlimit_nofile`, `rlimit_nproc`, `rlimit_rtprio`, `rlimit_rttime`, `rlimit_sigpending`, `rlimit_stack`

Examples

```
#load lib
library(RAppArmor)

#current limit
rlimit_as();

#set hard limit
rlimit_as(1e9);

#set separate hard and soft limit.
rlimit_as(1e9, 1e8);

#soft limits can be elevated
rlimit_as(soft = 1e7);
rlimit_as(soft = 1e9);

#set other limits
rlimit_core(1e9);
rlimit_data(1e9);
```

```
rlimit_fsize(1e9);
rlimit_memlock(10000);
rlimit_msgqueue(1e5);
rlimit_nofile(10);
rlimit_nproc(100);
rlimit_rttime(1e9);
rlimit_sigpending(1e4);
rlimit_stack(1000);
rlimit_cpu(10);
```

`rlimit_msgqueue` *Limit user message queue*

Description

Specifies the limit on the number of bytes that can be allocated for POSIX message queues for the real user ID of the calling process.

Usage

```
rlimit_msgqueue(hardlim, softlim = hardlim, pid = 0)
```

Arguments

`hardlim`	number of bytes
`softlim`	number of bytes
`pid`	id of the target process

Details

Specifies the limit on the number of bytes that can be allocated for POSIX message queues for the real user ID of the calling process. This limit is enforced for mq_open(3). Each message queue that the user creates counts (until it is removed) against this limit according to the formula:

bytes = attr.mq_maxmsg * sizeof(struct msg_msg *) + attr.mq_maxmsg * attr.mq_msgsize

where attr is the mq_attr structure specified as the fourth argument to mq_open(3).

The first addend in the formula, which includes sizeof(struct msg_msg *) (4 bytes on Linux/i386), ensures that the user cannot create an unlimited number of zero-length messages (such messages nevertheless each consume some system memory for bookkeeping overhead).

See Also

Other rlimit: `rlimit_as`, `rlimit_core`, `rlimit_cpu`, `rlimit_data`, `rlimit_fsize`, `rlimit_memlock`, `rlimit_nice`, `rlimit_nofile`, `rlimit_nproc`, `rlimit_rtprio`, `rlimit_rttime`, `rlimit_sigpending`, `rlimit_stack`

Examples

```
#load lib
library(RAppArmor)

#current limit
rlimit_as();

#set hard limit
rlimit_as(1e9);

#set separate hard and soft limit.
rlimit_as(1e9, 1e8);

#soft limits can be elevated
rlimit_as(soft = 1e7);
rlimit_as(soft = 1e9);

#set other limits
rlimit_core(1e9);
rlimit_data(1e9);
rlimit_fsize(1e9);
rlimit_memlock(10000);
rlimit_msgqueue(1e5);
rlimit_nofile(10);
rlimit_nproc(100);
rlimit_rttime(1e9);
rlimit_sigpending(1e4);
rlimit_stack(1000);
rlimit_cpu(10);
```

---

rlimit_nice *Limit nice value.*

Description

Specifies a ceiling to which the process's nice value can be raised using setpriority(2) or nice(2).

Usage

```
rlimit_nice(hardlim, softlim = hardlim, pid = 0)
```

Arguments

hardlim	priority value between -20 and 20
softlim	priority value between -20 and 20
pid	id of the target process

Details

Specifies a ceiling to which the process's nice value can be raised using setpriority(2) or nice(2). The actual ceiling for the nice value is calculated as 20 - rlim_cur. (This strangeness occurs because negative numbers cannot be specified as resource limit values, since they typically have special meanings. For example, RLIM_INFINITY typically is the same as -1.)

See Also

Other rlimit: `rlimit_as`, `rlimit_core`, `rlimit_cpu`, `rlimit_data`, `rlimit_fsize`, `rlimit_memlock`, `rlimit_msgqueue`, `rlimit_nofile`, `rlimit_nproc`, `rlimit_rtprio`, `rlimit_rttime`, `rlimit_sigpending`, `rlimit_stack`

Examples

```
#load lib
library(RAppArmor)

#current limit
rlimit_as();

#set hard limit
rlimit_as(1e9);

#set separate hard and soft limit.
rlimit_as(1e9, 1e8);

#soft limits can be elevated
rlimit_as(soft = 1e7);
rlimit_as(soft = 1e9);

#set other limits
rlimit_core(1e9);
rlimit_data(1e9);
rlimit_fsize(1e9);
rlimit_memlock(10000);
rlimit_msgqueue(1e5);
rlimit_nofile(10);
rlimit_nproc(100);
rlimit_rttime(1e9);
rlimit_sigpending(1e4);
rlimit_stack(1000);
rlimit_cpu(10);
```

`rlimit_nofile` *Limit file descriptors*

Description

Specifies a value one greater than the maximum file descriptor number that can be opened by this process.

Usage

```
rlimit_nofile(hardlim, softlim = hardlim, pid = 0)
```

Arguments

hardlim	number greater than 1
softlim	number greater than 1
pid	id of the target process

Details

Specifies a value one greater than the maximum file descriptor number that can be opened by this process. Attempts (open(2), pipe(2), dup(2), etc.) to exceed this limit yield the error EMFILE. (Historically, this limit was named RLIMIT_OFILE on BSD.)

See Also

Other rlimit: rlimit_as, rlimit_core, rlimit_cpu, rlimit_data, rlimit_fsize, rlimit_memlock, rlimit_msgqueue, rlimit_nice, rlimit_nproc, rlimit_rtprio, rlimit_rttime, rlimit_sigpending, rlimit_stack

Examples

```
#load lib
library(RAppArmor)

#current limit
rlimit_as();

#set hard limit
rlimit_as(1e9);

#set separate hard and soft limit.
rlimit_as(1e9, 1e8);

#soft limits can be elevated
rlimit_as(soft = 1e7);
rlimit_as(soft = 1e9);

#set other limits
rlimit_core(1e9);
rlimit_data(1e9);
rlimit_fsize(1e9);
rlimit_memlock(10000);
rlimit_msgqueue(1e5);
rlimit_nofile(10);
rlimit_nproc(100);
rlimit_rttime(1e9);
rlimit_sigpending(1e4);
rlimit_stack(1000);
rlimit_cpu(10);
```

`rlimit_nproc` *Limit number of processes*

Description

The maximum number of processes (or, more precisely on Linux, threads) that can be created for the real user ID.

Usage

```
rlimit_nproc(hardlim, softlim = hardlim, pid = 0)
```

Arguments

`hardlim`	number greater than 1
`softlim`	number greater than 1
`pid`	id of the target process

Details

The maximum number of processes (or, more precisely on Linux, threads) that can be created for the real user ID of the calling process. Upon encountering this limit, fork(2) fails with the error EAGAIN.

See Also

Other rlimit: `rlimit_as`, `rlimit_core`, `rlimit_cpu`, `rlimit_data`, `rlimit_fsize`, `rlimit_memlock`, `rlimit_msgqueue`, `rlimit_nice`, `rlimit_nofile`, `rlimit_rtprio`, `rlimit_rttime`, `rlimit_sigpending`, `rlimit_stack`

Examples

```
#load lib
library(RAppArmor)

#current limit
rlimit_as();

#set hard limit
rlimit_as(1e9);

#set separate hard and soft limit.
rlimit_as(1e9, 1e8);

#soft limits can be elevated
rlimit_as(soft = 1e7);
rlimit_as(soft = 1e9);
```

```
#set other limits
rlimit_core(1e9);
rlimit_data(1e9);
rlimit_fsize(1e9);
rlimit_memlock(10000);
rlimit_msgqueue(1e5);
rlimit_nofile(10);
rlimit_nproc(100);
rlimit_rttime(1e9);
rlimit_sigpending(1e4);
rlimit_stack(1000);
rlimit_cpu(10);
```

`rlimit_rtprio` *Limit real-time priority*

Description

Specifies a ceiling on the real-time priority

Usage

```
rlimit_rtprio(hardlim, softlim = hardlim, pid = 0)
```

Arguments

`hardlim`	real time priority value
`softlim`	real time priority value
`pid`	id of the target process

Details

Specifies a ceiling on the real-time priority that may be set for this process using sched_setscheduler(2) and sched_setparam(2).

See Also

Other rlimit: `rlimit_as`, `rlimit_core`, `rlimit_cpu`, `rlimit_data`, `rlimit_fsize`, `rlimit_memlock`, `rlimit_msgqueue`, `rlimit_nice`, `rlimit_nofile`, `rlimit_nproc`, `rlimit_rttime`, `rlimit_sigpending`, `rlimit_stack`

Examples

```
#load lib
library(RAppArmor)

#current limit
rlimit_as();
```

```
#set hard limit
rlimit_as(1e9);

#set separate hard and soft limit.
rlimit_as(1e9, 1e8);

#soft limits can be elevated
rlimit_as(soft = 1e7);
rlimit_as(soft = 1e9);

#set other limits
rlimit_core(1e9);
rlimit_data(1e9);
rlimit_fsize(1e9);
rlimit_memlock(10000);
rlimit_msgqueue(1e5);
rlimit_nofile(10);
rlimit_nproc(100);
rlimit_rttime(1e9);
rlimit_sigpending(1e4);
rlimit_stack(1000);
rlimit_cpu(10);
```

`rlimit_rttime` *Limit real-time cpu*

Description

Specifies a limit (in microseconds) on the amount of CPU time that a process scheduled under a real-time scheduling policy may consume without making a blocking system call.

Usage

```
rlimit_rttime(hardlim, softlim = hardlim, pid = 0)
```

Arguments

`hardlim`	time in microsec
`softlim`	time in microsec
`pid`	id of the target process

Details

Specifies a limit (in microseconds) on the amount of CPU time that a process scheduled under a real-time scheduling policy may consume without making a blocking system call. For the purpose of this limit, each time a process makes a blocking system call, the count of its consumed CPU time is reset to zero. The CPU time count is not reset if the process continues trying to use the CPU but is preempted, its time slice expires, or it calls sched_yield(2). Upon reaching the soft limit,

the process is sent a SIGXCPU signal. If the process catches or ignores this signal and continues consuming CPU time, then SIGXCPU will be generated once each second until the hard limit is reached, at which point the process is sent a SIGKILL signal. The intended use of this limit is to stop a runaway real-time process from locking up the system.

See Also

Other rlimit: `rlimit_as`, `rlimit_core`, `rlimit_cpu`, `rlimit_data`, `rlimit_fsize`, `rlimit_memlock`, `rlimit_msgqueue`, `rlimit_nice`, `rlimit_nofile`, `rlimit_nproc`, `rlimit_rtprio`, `rlimit_sigpending`, `rlimit_stack`

Examples

```
#load lib
library(RAppArmor)

#current limit
rlimit_as();

#set hard limit
rlimit_as(1e9);

#set separate hard and soft limit.
rlimit_as(1e9, 1e8);

#soft limits can be elevated
rlimit_as(soft = 1e7);
rlimit_as(soft = 1e9);

#set other limits
rlimit_core(1e9);
rlimit_data(1e9);
rlimit_fsize(1e9);
rlimit_memlock(10000);
rlimit_msgqueue(1e5);
rlimit_nofile(10);
rlimit_nproc(100);
rlimit_rttime(1e9);
rlimit_sigpending(1e4);
rlimit_stack(1000);
rlimit_cpu(10);
```

`rlimit_sigpending` *Limit signal queue*

Description

Specifies the limit on the number of signals that may be queued for the real user ID of the calling process.

Usage

```
rlimit_sigpending(hardlim, softlim = hardlim, pid = 0)
```

Arguments

hardlim	number of signals
softlim	number of signals
pid	id of the target process

Details

Specifies the limit on the number of signals that may be queued for the real user ID of the calling process. Both standard and real-time signals are counted for the purpose of checking this limit. However, the limit is only enforced for sigqueue(3); it is always possible to use kill(2) to queue one instance of any of the signals that are not already queued to the process.

See Also

Other rlimit: rlimit_as, rlimit_core, rlimit_cpu, rlimit_data, rlimit_fsize, rlimit_memlock, rlimit_msgqueue, rlimit_nice, rlimit_nofile, rlimit_nproc, rlimit_rtprio, rlimit_rttime, rlimit_stack

Examples

```
#load lib
library(RAppArmor)

#current limit
rlimit_as();

#set hard limit
rlimit_as(1e9);

#set separate hard and soft limit.
rlimit_as(1e9, 1e8);

#soft limits can be elevated
rlimit_as(soft = 1e7);
rlimit_as(soft = 1e9);

#set other limits
rlimit_core(1e9);
rlimit_data(1e9);
rlimit_fsize(1e9);
rlimit_memlock(10000);
rlimit_msgqueue(1e5);
rlimit_nofile(10);
rlimit_nproc(100);
rlimit_rttime(1e9);
rlimit_sigpending(1e4);
```

```
rlimit_stack(1000);
rlimit_cpu(10);
```

`rlimit_stack` *Limit stack size.*

Description

Limits the maximum size of the process stack, in bytes.

Usage

```
rlimit_stack(hardlim, softlim = hardlim, pid = 0)
```

Arguments

`hardlim`	size of stack
`softlim`	size of stack
`pid`	id of the target process

Details

The maximum size of the process stack, in bytes. Upon reaching this limit, a SIGSEGV signal is generated. To handle this signal, a process must employ an alternate signal stack (sigaltstack(2)). Since Linux 2.6.23, this limit also determines the amount of space used for the process's command-line arguments and environment variables; for details, see execve(2).

See Also

Other rlimit: `rlimit_as`, `rlimit_core`, `rlimit_cpu`, `rlimit_data`, `rlimit_fsize`, `rlimit_memlock`, `rlimit_msgqueue`, `rlimit_nice`, `rlimit_nofile`, `rlimit_nproc`, `rlimit_rtprio`, `rlimit_rttime`, `rlimit_sigpending`

Examples

```
#load lib
library(RAppArmor)

#current limit
rlimit_as();

#set hard limit
rlimit_as(1e9);

#set separate hard and soft limit.
rlimit_as(1e9, 1e8);

#soft limits can be elevated
```

```
rlimit_as(soft = 1e7);
rlimit_as(soft = 1e9);

#set other limits
rlimit_core(1e9);
rlimit_data(1e9);
rlimit_fsize(1e9);
rlimit_memlock(10000);
rlimit_msgqueue(1e5);
rlimit_nofile(10);
rlimit_nproc(100);
rlimit_rttime(1e9);
rlimit_sigpending(1e4);
rlimit_stack(1000);
rlimit_cpu(10);
```

`setgid` *Get/Set GID*

Description

Wrappers for getgid and setgid in Linux.

Usage

```
setgid(gid)
```

Arguments

`gid`	group ID

`setpriority` *Set process priority*

Description

Set the priority of the current process. High value is low priority. Only superusers can lower the value.

Usage

```
setpriority(prio)
```

Arguments

`prio`	priority value between -20 and 20

setuid *Get/Set UID*

Description

Wrappers for getuid and setuid in Linux.

Usage

```
setuid(uid)
```

Arguments

uid	user ID

userinfo *Lookup user info*

Description

Function looks up uid, gid, and userinfo for a given username.

Usage

```
userinfo(username, uid, gid)
```

Arguments

username	character name identifying the loginname of the user.
uid	integer specifying the uid of the user to lookup.
gid	integer specifying the gid to lookup.

Value

a parsed row from /etc/passwd