NAME

getcon, getprevcon, getpidcon − get SELinux security context of a process

freecon, freeconary − free memory associated with SELinux security contexts

getpeercon − get security context of a peer socket

setcon − set current security context of a process

SYNOPSIS

#include <selinux/selinux.h>

int getcon(char *****context*);**

int getcon_raw(char *****context*);**

int getprevcon(char *****context*);**

int getprevcon_raw(char *****context*);**

int getpidcon(pid_t *pid***, char *****context***);**

int getpidcon_raw(pid_t *pid***, char *****context***);**

int getpeercon(int *fd***, char *****context***);**

int getpeercon_raw(int *fd***, char *****context***);**

void freecon(char *****con*);**

void freeconary(char *****con*);**

int setcon(const char *****context*);**

int setcon_raw(const char *****context*);**

DESCRIPTION

getcon()
retrieves the context of the current process, which must be free'd with **freecon**().

getprevcon()
same as getcon but gets the context before the last exec.

getpidcon()
returns the process context for the specified PID, which must be free'd with **freecon**().

getpeercon()
retrieves the context of the peer socket, which must be free'd with **freecon**().

freecon()
frees the memory allocated for a security context.

If *con* is NULL, no operation is performed.

freeconary()
frees the memory allocated for a context array.

If *con* is NULL, no operation is performed.

setcon()
sets the current security context of the process to a new value. Note that use of this function requires that the entire application be trusted to maintain any desired separation between the old and new security contexts, unlike exec-based transitions performed via **setexeccon**(3). When possible, decompose your application and use **setexeccon**(3) and **execve**(3) instead.

Since access to file descriptors is revalidated upon use by SELinux, the new context must be explicitly authorized in the policy to use the descriptors opened by the old context if that is desired. Otherwise, attempts by the process to use any existing descriptors (including *stdin*, *stdout*, and *stderr*) after performing the **setcon**() will fail.

A multi-threaded application can perform a **setcon**() prior to creating any child threads, in which case all of the child threads will inherit the new context. However, prior to Linux 2.6.28, **setcon**() would fail if there are any other threads running in the same process since this would yield an inconsistency among the security contexts of threads sharing the same memory space. Since Linux 2.6.28, **setcon()** is permitted for threads within a multi-threaded process if the new security context is bounded by the old security context, where the bounded relation is defined through typebounds statements in the policy and guarantees that the new security context has a subset of the permissions of the old security context.

If the process was being ptraced at the time of the **setcon**() operation, ptrace permission will be revalidated against the new context and the **setcon**() will fail if it is not allowed by policy.

***_raw()**
getcon_raw(), **getprevcon_raw**(), **getpidcon_raw**(), **getpeercon_raw**() and **setcon_raw**() behave identically to their non-raw counterparts but do not perform context translation.

RETURN VALUE
On error −1 is returned with errno set. On success 0 is returned.

NOTES
The retrieval functions might return success and set**conte xt* to NULL if and only if SELinux is not enabled.

SEE ALSO
selinux(8), **setexeccon**(3)