NAME

security_compute_av, security_compute_av_flags, security_compute_create, security_compute_create_name, security_compute_relabel, security_compute_member, security_compute_user, security_validatetrans, security_get_initial_context − query the SELinux policy database in the kernel

SYNOPSIS

#include <selinux/selinux.h>

int security_compute_av(char ****scon*, char ****tcon***, security_class_t** *tclass***, access_vector_t** *requested***, struct av_decision ****avd***);**

int security_compute_av_raw(char ****scon*, char ****tcon***, security_class_t** *tclass***, access_vector_t** *requested***, struct av_decision ****avd***);**

int security_compute_av_flags(char ****scon*, char ****tcon***, security_class_t** *tclass***, access_vector_t** *requested***, struct av_decision ****avd***);**

int security_compute_av_flags_raw(char ****scon*, char ****tcon***, security_class_t** *tclass***, access_vector_t** *requested***, struct av_decision ****avd***);**

int security_compute_create(char ****scon*, char ****tcon***, security_class_t** *tclass***, char *******newcon***);**

int security_compute_create_raw(char ****scon*, char ****tcon***, security_class_t** *tclass***, char *******newcon***);**

int security_compute_create_name(char ****scon*, char ****tcon***, security_class_t** *tclass***, const char *****objname***, char *******newcon***);**

int security_compute_create_name_raw(char *****scon*, char *****tcon***, security_class_t** *tclass***, const char *****objname***, char *******newcon***);**

int security_compute_relabel(char *****scon*, char *****tcon***, security_class_t** *tclass***, char *******newcon***);**

int security_compute_relabel_raw(char *****scon*, char *****tcon***, security_class_t** *tclass***, char *******newcon***);**

int security_compute_member(char *****scon*, char *****tcon***, security_class_t** *tclass***, char *******newcon***);**

int security_compute_member_raw(char *****scon*, char *****tcon***, security_class_t** *tclass***, char *******newcon***);**

int security_compute_user(char *****scon*, const char *****username***, char *********con***);**

int security_compute_user_raw(char *****scon*, const char *****username***, char *********con***);**

int security_validatetrans(char *****scon*, const char *****tcon***, security_class_t** *tclass***, char *****newcon***);**

int security_validatetrans_raw(char *****scon*, const char *****tcon***, security_class_t** *tclass***, char *****newcon***);**

int security_get_initial_context(const char *****name*, char *******con***);**

int security_get_initial_context_raw(const char *****name*, char *******con***);**

int selinux_check_access(const char *****scon*, const char *****tcon***, const char *****class***, const char *****perm***, void *****auditdata);**

int selinux_check_passwd_access(access_vector_t *requested***);**

int checkPasswdAccess(access_vector_t *requested***);**

DESCRIPTION

This family of functions is used to obtain policy decisions from the SELinux kernel security server (policy engine). In general, direct use of**security_compute_a v**() and its variant interfaces is discouraged in favor of using **selinux_check_access**() since the latter automatically handles the dynamic mapping of class and permission names to their policy values, initialization and use of the Access Vector Cache (AVC), and proper handling of per-domain and global permissive mode and allow_unknown.

When using any of the functions that take policy integer values for classes or permissions as inputs, use **string_to_security_class(3)** and **string_to_av_perm(3)** to map the class and permission names to their policy values. These values may change across a policy reload, so they should be re-acquired on every use or using a **SELINUX_CB_POLICYLOAD** callback set via **selinux_set_callback(3).**

An alternative approach is to use **selinux_set_mapping(3)** to create a mapping from class and permission index values used by the application to the policy values, thereby allowing the application to pass its own fixed constants for the classes and permissions to these functions and internally mapping them on demand. However, this also requires setting up a callback as above to address policy reloads.

security_compute_av() queries whether the policy permits the source context *scon* to access the target context *tcon* via class *tclass* with the *requested* access vector. The decision is returned in*avd*.

security_compute_av_flags() is identical to **security_compute_av** but additionally sets the *fla gs* field of *avd*. Currently one flag is supported:**SELINUX_AVD_FLAGS_PERMISSIVE**, which indicates the decision is computed on a permissive domain.

security_compute_create() is used to compute a context to use for labeling a new object in a particular class based on a SID pair.

security_compute_create_name() is identical to **security_compute_create**() but also takes name of the new object in creation as an argument. When**TYPE_TRANSITION** rule on the gi ven class and a SID pair has object name extension, we shall be able to obtain a correct *newcon* according to the security policy. Note that this interface is only supported on the linux 2.6.40 or later. In the older kernel, the object name will be simply ignored.

security_compute_relabel() is used to compute the new context to use when relabeling an object, it is used in the pam_selinux.so source and the newrole source to determine the correct label for the tty at login time, but can be used for other things.

security_compute_member() is used to compute the context to use when labeling a polyinstantiated object instance.

security_compute_user() is used to determine the set of user contexts that can be reached from a source context. This function is deprecated; use **get_ordered_context_list**(3) instead.

security_validatetrans() is used to determine if a transition from scon to newcon using tcon as the object is valid for object class tclass. This checks against the mlsvalidatetrans and validatetrans constraints in the loaded policy. Returns 0 if allowed, and -1 if an error occurred with errno set.

security_get_initial_context() is used to get the context of a kernel initial security identifier specified by *name*

security_compute_av_raw(), **security_compute_av_flags_raw**(), **security_compute_create_raw**(), **security_compute_create_name_raw**(), **security_compute_relabel_raw**(), **security_compute_member_raw**(), **security_compute_user_raw**() **security_validatetrans_raw**() and

security_get_initial_context_raw() behave identically to their non-raw counterparts but do not perform context translation.

selinux_check_access() is used to check if the source context has the access permission for the specified class on the target context.

selinux_check_passwd_access() is used to check for a permission in the *passwd* class. **selinux_check_passwd_access**() uses **getprevcon(3)** for the source and target security contexts.

checkPasswdAccess() is a deprecated alias of the **selinux_check_passwd_access**() function.

RETURN VALUE

Returns zero on success or −1 on error.

SEE ALSO

string_to_security_class(3), **string_to_av_perm**(3), **selinux_set_callback**(3), **selinux_set_mapping**(3), **getprevcon**(3), **get_ordered_context_list**(3), **selinux**(8)