Red Hat Enterprise Linux Release 9.2 Manual Pages on 'request_key.2' command

$ man request_key.2

REQUEST_KEY(2) Linux Key Management Calls REQUEST_KEY(2)

NAME

request_key - request a key from the kernel's key management facility

SYNOPSIS

```
#include <sys/types.h>

#include <keyutils.h>

key_serial_t request_key(const char *type, const char *description,
                         const char *callout_info,
                         key_serial_t dest_keyring);
```

No glibc wrapper is provided for this system call; see NOTES.

DESCRIPTION

request_key() attempts to find a key of the given type with a descrip? tion (name) that matches the specified description. If such a key could not be found, then the key is optionally created. If the key is found or created, request_key() attaches it to the keyring whose ID is specified in dest_keyring and returns the key's serial number.

request_key() first recursively searches for a matching key in all of the keyrings attached to the calling process. The keyrings are searched in the order: thread-specific keyring, process-specific keyring, and then session keyring.

If request_key() is called from a program invoked by request_key() on behalf of some other process to generate a key, then the keyrings of that other process will be searched next, using that other process's

user ID, group ID, supplementary group IDs, and security context to de?
termine access.

The search of the keyring tree is breadth-first: the keys in each keyring searched are checked for a match before any child keyrings are recursed into. Only keys for which the caller has search permission be found, and only keyrings for which the caller has search permission may be searched.

If the key is not found and callout is NULL, then the call fails with the error ENOKEY.

If the key is not found and callout is not NULL, then the kernel at?
tempts to invoke a user-space program to instantiate the key. The de?
tails are given below.

The dest_keyring serial number may be that of a valid keyring for which the caller has write permission, or it may be one of the following spe?
cial keyring IDs:

KEY_SPEC_THREAD_KEYRING

This specifies the caller's thread-specific keyring (see thread-keyring(7)).

KEY_SPEC_PROCESS_KEYRING

This specifies the caller's process-specific keyring (see process-keyring(7)).

KEY_SPEC_SESSION_KEYRING

This specifies the caller's session-specific keyring (see ses?
sion-keyring(7)).

KEY_SPEC_USER_KEYRING

This specifies the caller's UID-specific keyring (see user-keyring(7)).

KEY_SPEC_USER_SESSION_KEYRING

This specifies the caller's UID-session keyring (see user-ses?
sion-keyring(7)).

When the dest_keyring is specified as 0 and no key construction has been performed, then no additional linking is done.

Otherwise, if dest_keyring is 0 and a new key is constructed, the new key will be linked to the "default" keyring. More precisely, when the kernel tries to determine to which keyring the newly constructed key should be linked, it tries the following keyrings, beginning with the keyring set via the keyctl(2) KEYCTL_SET_REQKEY_KEYRING operation and continuing in the order shown below until it finds the first keyring that exists:

? The requestor keyring (KEY_REQKEY_DEFL_REQUESTOR_KEYRING, since Linux 2.6.29).

? The thread-specific keyring (KEY_REQKEY_DEFL_THREAD_KEYRING; see thread-keyring(7)).

? The process-specific keyring (KEY_REQKEY_DEFL_PROCESS_KEYRING; see process-keyring(7)).

? The session-specific keyring (KEY_REQKEY_DEFL_SESSION_KEYRING; see session-keyring(7)).

? The session keyring for the process's user ID (KEY_RE? QKEY_DEFL_USER_SESSION_KEYRING; see user-session-keyring(7)). This keyring is expected to always exist.

? The UID-specific keyring (KEY_REQKEY_DEFL_USER_KEYRING; see user-keyring(7)). This keyring is also expected to always exist.

If the keyctl(2) KEYCTL_SET_REQKEY_KEYRING operation specifies KEY_RE? QKEY_DEFL_DEFAULT (or no KEYCTL_SET_REQKEY_KEYRING operation is per? formed), then the kernel looks for a keyring starting from the begin? ning of the list.

Requesting user-space instantiation of a key

If the kernel cannot find a key matching type and description, and callout is not NULL, then the kernel attempts to invoke a user-space program to instantiate a key with the given type and description. In this case, the following steps are performed:

a) The kernel creates an uninstantiated key, U, with the requested type and description.

b) The kernel creates an authorization key, V, that refers to the key U and records the facts that the caller of request_key() is:

(1) the context in which the key U should be instantiated and se?

cured, and

(2) the context from which associated key requests may be satis? fied.

The authorization key is constructed as follows:

* The key type is ".request_key_auth".

* The key's UID and GID are the same as the corresponding filesys? tem IDs of the requesting process.

* The key grants view, read, and search permissions to the key possessor as well as view permission for the key user.

* The description (name) of the key is the hexadecimal string rep? resenting the ID of the key that is to be instantiated in the requesting program.

* The payload of the key is taken from the data specified in call? out_info.

* Internally, the kernel also records the PID of the process that called request_key().

c) The kernel creates a process that executes a user-space service such as request-key(8) with a new session keyring that contains a link to the authorization key, V.

This program is supplied with the following command-line arguments:

[0] The string "/sbin/request-key".

[1] The string "create" (indicating that a key is to be created).

[2] The ID of the key that is to be instantiated.

[3] The filesystem UID of the caller of request_key().

[4] The filesystem GID of the caller of request_key().

[5] The ID of the thread keyring of the caller of request_key(). This may be zero if that keyring hasn't been created.

[6] The ID of the process keyring of the caller of request_key(). This may be zero if that keyring hasn't been created.

[7] The ID of the session keyring of the caller of request_key().

Note: each of the command-line arguments that is a key ID is en? coded in decimal (unlike the key IDs shown in /proc/keys, which are shown as hexadecimal values).

d) The program spawned in the previous step:

* Assumes the authority to instantiate the key U using the keyctl(2) KEYCTL_ASSUME_AUTHORITY operation (typically via the keyctl_assume_authority(3) function).

* Obtains the callout data from the payload of the authorization key V (using the keyctl(2) KEYCTL_READ operation (or, more com? monly, the keyctl_read(3) function) with a key ID value of KEY_SPEC_REQKEY_AUTH_KEY).

* Instantiates the key (or execs another program that performs that task), specifying the payload and destination keyring. (The destination keyring that the requestor specified when call? ing request_key() can be accessed using the special key ID KEY_SPEC_REQUESTOR_KEYRING.) Instantiation is performed using the keyctl(2) KEYCTL_INSTANTIATE operation (or, more commonly, the keyctl_instantiate(3) function). At this point, the re? quest_key() call completes, and the requesting program can con? tinue execution.

If these steps are unsuccessful, then an ENOKEY error will be returned to the caller of request_key() and a temporary, negatively instantiated key will be installed in the keyring specified by dest_keyring. This will expire after a few seconds, but will cause subsequent calls to re? quest_key() to fail until it does. The purpose of this negatively in? stantiated key is to prevent (possibly different) processes making re? peated requests (that require expensive request-key(8) upcalls) for a key that can't (at the moment) be positively instantiated.

Once the key has been instantiated, the authorization key (KEY_SPEC_RE? QKEY_AUTH_KEY) is revoked, and the destination keyring (KEY_SPEC_RE? QUESTOR_KEYRING) is no longer accessible from the request-key(8) pro? gram.

If a key is created, then?regardless of whether it is a valid key or a negatively instantiated key?it will displace any other key with the same type and description from the keyring specified in dest_keyring.

RETURN VALUE

On success, request_key() returns the serial number of the key it found or caused to be created. On error, -1 is returned and errno is set to indicate the cause of the error.

ERRORS

EACCES The keyring wasn't available for modification by the user.

EDQUOT The key quota for this user would be exceeded by creating this key or linking it to the keyring.

EFAULT One of type, description, or callout_info points outside the process's accessible address space.

EINTR The request was interrupted by a signal; see signal(7).

EINVAL The size of the string (including the terminating null byte) specified in type or description exceeded the limit (32 bytes and 4096 bytes respectively).

EINVAL The size of the string (including the terminating null byte) specified in callout_info exceeded the system page size.

EKEYEXPIRED

An expired key was found, but no replacement could be obtained.

EKEYREJECTED

The attempt to generate a new key was rejected.

EKEYREVOKED

A revoked key was found, but no replacement could be obtained.

ENOKEY No matching key was found.

ENOMEM Insufficient memory to create a key.

EPERM The type argument started with a period ('.').

VERSIONS

This system call first appeared in Linux 2.6.10. The ability to in?stantiate keys upon request was added in Linux 2.6.13.

CONFORMING TO

This system call is a nonstandard Linux extension.

NOTES

No wrapper for this system call is provided in glibc. A wrapper is provided in the libkeyutils package. When employing the wrapper in that library, link with -lkeyutils.

EXAMPLES

The program below demonstrates the use of request_key(). The type, de?

scription, and callout_info arguments for the system call are taken

from the values supplied in the command-line arguments. The call spec?

ifies the session keyring as the target keyring.

In order to demonstrate this program, we first create a suitable entry

in the file /etc/request-key.conf.

```
$ sudo sh
# echo 'create user mtk:* *   /bin/keyctl instantiate %k %c %S' \
          > /etc/request-key.conf
# exit
```

This entry specifies that when a new "user" key with the prefix "mtk:"

must be instantiated, that task should be performed via the keyctl(1)

command's instantiate operation. The arguments supplied to the instan?

tiate operation are: the ID of the uninstantiated key (%k); the callout

data supplied to the request_key() call (%c); and the session keyring

(%S) of the requestor (i.e., the caller of request_key()). See re?

quest-key.conf(5) for details of these % specifiers.

Then we run the program and check the contents of /proc/keys to verify

that the requested key has been instantiated:

```
$ ./t_request_key user mtk:key1 "Payload data"
$ grep '2dddaf50' /proc/keys
2dddaf50 I--Q---  1 perm 3f010000  1000  1000 user  mtk:key1: 12
```

For another example of the use of this program, see keyctl(2).

Program source

```
/* t_request_key.c */
#include <sys/types.h>
#include <keyutils.h>
#include <stdint.h>
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
int
```

```
main(int argc, char *argv[])
{
    key_serial_t key;
    if (argc != 4) {
        fprintf(stderr, "Usage: %s type description callout-data\n",
                argv[0]);
        exit(EXIT_FAILURE);
    }
    key = request_key(argv[1], argv[2], argv[3],
                      KEY_SPEC_SESSION_KEYRING);
    if (key == -1) {
        perror("request_key");
        exit(EXIT_FAILURE);
    }
    printf("Key ID is %jx\n", (uintmax_t) key);
    exit(EXIT_SUCCESS);
}
```

SEE ALSO

keyctl(1), add_key(2), keyctl(2), keyctl(3), capabilities(7), keyrings(7), keyutils(7), persistent-keyring(7), process-keyring(7), session-keyring(7), thread-keyring(7), user-keyring(7), user-session-keyring(7), request-key(8)

The kernel source files Documentation/security/keys/core.rst and Documentation/keys/request-key.rst (or, before Linux 4.13, in the files Documentation/security/keys.txt and Documentation/security/keys-request-key.txt).

COLOPHON

This page is part of release 5.10 of the Linux man-pages project. A description of the project, information about reporting bugs, and the latest version of this page, can be found at https://www.kernel.org/doc/man-pages/.