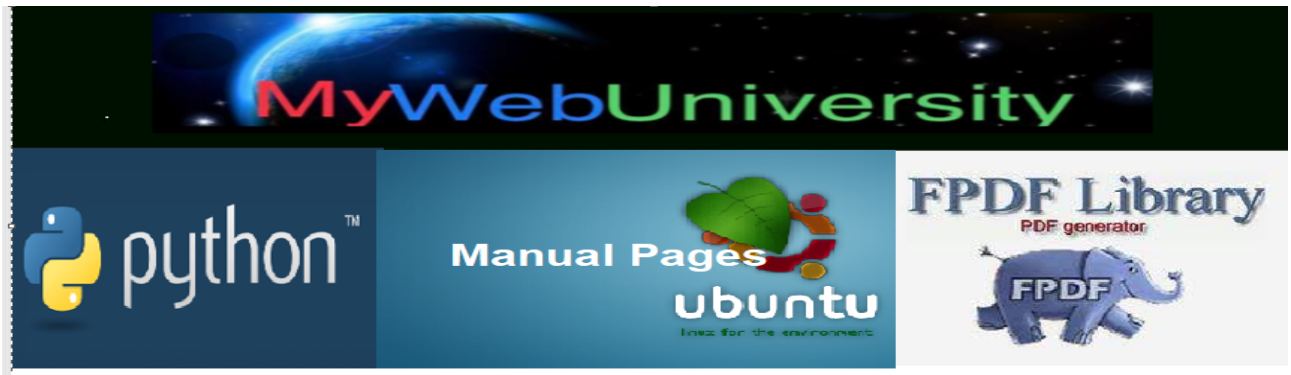

Full credit is given to the above companies including the OS that this PDF file was generated!

Linux Ubuntu 26.04 Manual Pages on command 'memfd_secret.2'

$ man memfd_secret.2

memfd_secret(2) System Calls Manual memfd_secret(2)

NAME

memfd_secret - create an anonymous RAM-based file to access secret memory regions

LIBRARY

Standard C library (libc, -lc)

SYNOPSIS

```
#include <sys/syscall.h>      /* Definition of SYS_* constants */
#include <unistd.h>
int syscall(SYS_memfd_secret, unsigned int flags);
```

Note: glibc provides no wrapper for memfd_secret(), necessitating the use of syscall(2).

DESCRIPTION

memfd_secret() creates an anonymous RAM-based file and returns a file descriptor that refers to it. The file provides a way to create and access memory regions with stronger protection than usual RAM-based files and anonymous memory mappings. Once all open references to the file are closed, it is automatically released. The initial size of the file is set to 0. Following the call, the file size should be set using ftruncate(2).

The memory areas backing the file created with memfd_secret(2) are visible only to the processes that have access to the file descriptor. The memory region is removed from the kernel page tables and only the page tables of the processes holding the file descriptor map the corresponding physical memory. (Thus, the pages in the region can't be accessed by the kernel itself, so that, for example, pointers to the region can't be passed to system calls.)

The following values may be bitwise ORed in flags to control the behavior of memfd_secret():

FD_CLOEXEC

Set the close-on-exec flag on the new file descriptor, which causes the region to be removed from the process on execve(2). See the description of the O_CLOEXEC flag in open(2)

As its return value, memfd_secret() returns a new file descriptor that refers to an anony? mous file. This file descriptor is opened for both reading and writing (O_RDWR) and O_LARGEFILE is set for the file descriptor.

With respect to fork(2) and execve(2), the usual semantics apply for the file descriptor created by memfd_secret(). A copy of the file descriptor is inherited by the child produced by fork(2) and refers to the same file. The file descriptor is preserved across execve(2), unless the close-on-exec flag has been set.

The memory region is locked into memory in the same way as with mlock(2), so that it will never be written into swap, and hibernation is inhibited for as long as any memfd_secret() descriptions exist. However the implementation of memfd_secret() will not try to populate the whole range during the mmap(2) call that attaches the region into the process's address space; instead, the pages are only actually allocated as they are faulted in. The amount of memory allowed for memory mappings of the file descriptor obeys the same rules as mlock(2) and cannot exceed RLIMIT_MEMLOCK.

RETURN VALUE

On success, memfd_secret() returns a new file descriptor. On error, -1 is returned and er? rno is set to indicate the error.

ERRORS

EINVAL flags included unknown bits.

EMFILE The per-process limit on the number of open file descriptors has been reached.

EMFILE The system-wide limit on the total number of open files has been reached.

ENOMEM There was insufficient memory to create a new anonymous file.

ENOSYS memfd_secret() is not implemented on this architecture, or has not been enabled on the kernel command-line with secretmem_enable=1.

STANDARDS

Linux.

HISTORY

Linux 5.14.

Before Linux 6.5, memfd_secret() was disabled by default and only available if the system

administrator turned it on using "secretmem.enable=y" kernel parameter.

NOTES

The memfd_secret() system call is designed to allow a user-space process to create a range of memory that is inaccessible to anybody else - kernel included. There is no 100% guaran? tee that kernel won't be able to access memory ranges backed by memfd_secret() in any cir? cumstances, but nevertheless, it is much harder to exfiltrate data from these regions.

memfd_secret() provides the following protections:

? Enhanced protection (in conjunction with all the other in-kernel attack prevention sys? tems) against ROP attacks. Absence of any in-kernel primitive for accessing memory backed by memfd_secret() means that one-gadget ROP attack can't work to perform data ex? filtration. The attacker would need to find enough ROP gadgets to reconstruct the miss? ing page table entries, which significantly increases difficulty of the attack, espe? cially when other protections like the kernel stack size limit and address space layout randomization are in place.

? Prevent cross-process user-space memory exposures. Once a region for a memfd_secret() memory mapping is allocated, the user can't accidentally pass it into the kernel to be transmitted somewhere. The memory pages in this region cannot be accessed via the direct map and they are disallowed in get_user_pages.

? Harden against exploited kernel flaws. In order to access memory areas backed by memfd_secret(), a kernel-side attack would need to either walk the page tables and create new ones, or spawn a new privileged user-space process to perform secrets exfiltration using ptrace(2).

To prevent potential data leaks of memory regions backed by memfd_secret() from a hyberna? tion image, hybernation is prevented when there are active memfd_secret() users.

SEE ALSO

fcntl(2), ftruncate(2), mlock(2), memfd_create(2), mmap(2), setrlimit(2)