[PATCH] virtiofsd: prevent opening of special files (CVE-2020-35517)

From: Stefan Hajnoczi
Subject: [PATCH] virtiofsd: prevent opening of special files (CVE-2020-35517)
Date: Thu, 21 Jan 2021 14:44:29 +0000

```
A well-behaved FUSE client does not attempt to open special files with
FUSE_OPEN because they are handled on the client side (e.g. device nodes
are handled by client-side device drivers).

The check to prevent virtiofsd from opening special files is missing in
a few cases, most notably FUSE_OPEN. A malicious client can cause
virtiofsd to open a device node, potentially allowing the guest to
escape. This can be exploited by a modified guest device driver. It is
not exploitable from guest userspace since the guest kernel will handle
special files inside the guest instead of sending FUSE requests.

This patch adds the missing checks to virtiofsd. This is a short-term
solution because it does not prevent a compromised virtiofsd process
from opening device nodes on the host.

Reported-by: Alex Xu <alex@alxu.ca>
Fixes: CVE-2020-35517
Signed-off-by: Stefan Hajnoczi <stefanha@redhat.com>
---
This issue was diagnosed on public IRC and is therefore already known
and not embargoed.

A stronger fix, and the long-term solution, is for users to mount the
shared directory and any sub-mounts with nodev, as well as nosuid and
noexec. Unfortunately virtiofsd cannot do this automatically because
bind mounts added by the user after virtiofsd has launched would not be
detected. I suggest the following:

1. Modify libvirt and Kata Containers to explicitly set these mount
   options.
2. Then modify virtiofsd to check that the shared directory has the
   necessary options at startup. Refuse to start if the options are
   missing so that the user is aware of the security requirements.

As a bonus this also increases the likelihood that other host processes
besides virtiofsd will be protected by nosuid/noexec/nodev so that a
malicious guest cannot drop these files in place and then arrange for a
host process to come across them.

Additionally, user namespaces have been discussed. They seem like a
worthwhile addition as an unprivileged or privilege-separated mode
although there are limitations with respect to security xattrs and the
actual uid/gid stored on the host file system not corresponding to the
guest uid/gid.
---
 tools/virtiofsd/passthrough_ll.c | 84 ++++++++++++++++++++++-----------
 1 file changed, 56 insertions(+), 28 deletions(-)

diff --git a/tools/virtiofsd/passthrough_ll.c b/tools/virtiofsd/passthrough_ll.c
index 5fb36d9407..e08352d649 100644
--- a/tools/virtiofsd/passthrough_ll.c
+++ b/tools/virtiofsd/passthrough_ll.c
@@ -555,6 +555,29 @@ static int lo_fd(fuse_req_t req, fuse_ino_t ino)
     return fd;
 }

+/*
+ * Open a file descriptor for an inode. Returns -EBADF if the inode is not a
+ * regular file or a directory. Use this helper function instead of raw
+ * openat(2) to prevent security issues when a malicious client opens special
+ * files such as block device nodes.
+ */
+static int lo_inode_open(struct lo_data *lo, struct lo_inode *inode,
+                         int open_flags)
+{
+    g_autofree char *fd_str = g_strdup_printf("%d", inode->fd);
+    int fd;
+
+    if (!S_ISREG(inode->filetype) && !S_ISDIR(inode->filetype)) {
+        return -EBADF;
+    }
+
+    fd = openat(lo->proc_self_fd, fd_str, open_flags);
+    if (fd < 0) {
+        return -errno;
+    }
+    return fd;
+}
+
 static void lo_init(void *userdata, struct fuse_conn_info *conn)
 {
     struct lo_data *lo = (struct lo_data *)userdata;
@@ -684,8 +707,7 @@ static void lo_setattr(fuse_req_t req, fuse_ino_t ino,
struct stat *attr,
         if (fi) {
             truncfd = fd;
         } else {
-            sprintf(procname, "%i", ifd);
-            truncfd = openat(lo->proc_self_fd, procname, O_RDWR);
+            truncfd = lo_inode_open(lo, inode, O_RDWR);
             if (truncfd < 0) {
                 goto out_err;
             }
@@ -1725,7 +1747,6 @@ static struct lo_inode_plock
*lookup_create_plock_ctx(struct lo_data *lo,
                                                      pid_t pid, int *err)
 {
     struct lo_inode_plock *plock;
-    char procname[64];
     int fd;

     plock =
@@ -1742,12 +1763,10 @@ static struct lo_inode_plock
*lookup_create_plock_ctx(struct lo_data *lo,
     }

     /* Open another instance of file which can be used for ofd locks. */
-    sprintf(procname, "%i", inode->fd);
-
     /* TODO: What if file is not writable? */
-    fd = openat(lo->proc_self_fd, procname, O_RDWR);
-    if (fd == -1) {
-        *err = errno;
+    fd = lo_inode_open(lo, inode, O_RDWR);
+    if (fd < 0) {
+        *err = -fd;
         free(plock);
         return NULL;
     }
```

```
@@ -1894,18 +1913,24 @@ static void lo_open(fuse_req_t req, fuse_ino_t ino,
struct fuse_file_info *fi)
 {
     int fd;
     ssize_t fh;
-    char buf[64];
     struct lo_data *lo = lo_data(req);
+    struct lo_inode *inode = lo_inode(req, ino);

     fuse_log(FUSE_LOG_DEBUG, "lo_open(ino=%" PRIu64 ", flags=%d)\n", ino,
              fi->flags);

+    if (!inode) {
+        fuse_reply_err(req, EBADF);
+        return;
+    }
+
     update_open_flags(lo->writeback, lo->allow_direct_io, fi);

-    sprintf(buf, "%i", lo_fd(req, ino));
-    fd = openat(lo->proc_self_fd, buf, fi->flags & ~O_NOFOLLOW);
-    if (fd == -1) {
-        return (void)fuse_reply_err(req, errno);
+    fd = lo_inode_open(lo, inode, fi->flags & ~O_NOFOLLOW);
+    if (fd < 0) {
+        lo_inode_put(lo, &inode);
+        fuse_reply_err(req, -fd);
+        return;
     }

     pthread_mutex_lock(&lo->mutex);
@@ -1913,6 +1938,7 @@ static void lo_open(fuse_req_t req, fuse_ino_t ino,
struct fuse_file_info *fi)
     pthread_mutex_unlock(&lo->mutex);
     if (fh == -1) {
         close(fd);
+        lo_inode_put(lo, &inode);
         fuse_reply_err(req, ENOMEM);
         return;
     }
@@ -1923,6 +1949,7 @@ static void lo_open(fuse_req_t req, fuse_ino_t ino,
struct fuse_file_info *fi)
     } else if (lo->cache == CACHE_ALWAYS) {
         fi->keep_cache = 1;
     }
+    lo_inode_put(lo, &inode);
     fuse_reply_open(req, fi);
 }

@@ -1982,39 +2009,40 @@ static void lo_flush(fuse_req_t req, fuse_ino_t ino,
struct fuse_file_info *fi)
 static void lo_fsync(fuse_req_t req, fuse_ino_t ino, int datasync,
                      struct fuse_file_info *fi)
 {
+    struct lo_inode *inode = lo_inode(req, ino);
+    struct lo_data *lo = lo_data(req);
     int res;
     int fd;
-    char *buf;

     fuse_log(FUSE_LOG_DEBUG, "lo_fsync(ino=%" PRIu64 ", fi=0x%p)\n", ino,
              (void *)fi);

+    if (!inode) {
+        fuse_reply_err(req, EBADF);
+        return;
+    }
+
     if (!fi) {
-        struct lo_data *lo = lo_data(req);
-
-        res = asprintf(&buf, "%i", lo_fd(req, ino));
-        if (res == -1) {
-            return (void)fuse_reply_err(req, errno);
-        }
-
-        fd = openat(lo->proc_self_fd, buf, O_RDWR);
-        free(buf);
-        if (fd == -1) {
-            return (void)fuse_reply_err(req, errno);
+        fd = lo_inode_open(lo, inode, O_RDWR);
+        if (fd < 0) {
+            res = -fd;
+            goto out;
         }
     } else {
         fd = lo_fi_fd(req, fi);
     }

     if (datasync) {
-        res = fdatasync(fd);
+        res = fdatasync(fd) == -1 ? errno : 0;
     } else {
-        res = fsync(fd);
+        res = fsync(fd) == -1 ? errno : 0;
     }
     if (!fi) {
         close(fd);
     }
-    fuse_reply_err(req, res == -1 ? errno : 0);
+out:
+    lo_inode_put(lo, &inode);
+    fuse_reply_err(req, res);
 }

 static void lo_read(fuse_req_t req, fuse_ino_t ino, size_t size, off_t offset,
--
2.29.2
```

---

reply via email to
Stefan Hajnoczi

[Prev in Thread] **Current Thread** [Next in Thread]

- **[PATCH] virtiofsd: prevent opening of special files (CVE-2020-35517)**, *Stefan Hajnoczi* <=
  - **Re: [PATCH] virtiofsd: prevent opening of special files (CVE-2020-35517)**, *Daniel P . Berrangé*, 2021/01/21
    - **Re: [PATCH] virtiofsd: prevent opening of special files (CVE-2020-35517)**, *Vivek Goyal*, 2021/01/22
    - **Re: [PATCH] virtiofsd: prevent opening of special files (CVE-2020-35517)**, *Stefan Hajnoczi*, 2021/01/25
  - **Re: [PATCH] virtiofsd: prevent opening of special files (CVE-2020-35517)**, *Alex Xu*, 2021/01/21
  - **Re: [PATCH] virtiofsd: prevent opening of special files (CVE-2020-35517)**, *Laszlo Ersek*, 2021/01/21
    - **Re: [PATCH] virtiofsd: prevent opening of special files (CVE-2020-35517)**, *Alex Xu*, 2021/01/21
      - **Re: [PATCH] virtiofsd: prevent opening of special files (CVE-2020-35517)**, *Laszlo Ersek*, 2021/01/21
  - **Re: [PATCH] virtiofsd: prevent opening of special files (CVE-2020-35517)**, *Dr. David Alan Gilbert*, 2021/01/21
  - **Re: [PATCH] virtiofsd: prevent opening of special files (CVE-2020-35517)**, *Vivek Goyal*, 2021/01/22
    - **Re: [PATCH] virtiofsd: prevent opening of special files (CVE-2020-35517)**, *Stefan Hajnoczi*, 2021/01/26

- Prev by Date: **Re: [PATCH 0/8] s390x/pci: Fixing s390 vfio-pci ISM support**
- Next by Date: **Re: [PATCH 0/8] s390x/pci: Fixing s390 vfio-pci ISM support**
- Previous by thread: **[PATCH] meson: convert wixl detection to Meson**
- Next by thread: **Re: [PATCH] virtiofsd: prevent opening of special files (CVE-2020-35517)**