Starred by 2 users

Owner:	jannh@google.com
CC:	proje...@google.com
Status:	Fixed *(Closed)*
Components:	----
Modified:	Dec 1, 2020

Deadline-90
Vendor-Linux
Severity-low
CCProjectZeroMembers
Finder-jannh
Product-Linux
Methodology-source-review
Reported-2020-Aug-13
Product-udisks
Vendor-udisks
Fixed-2020-Aug-26
CVE-2020-29371

Issue 2077: udisks + Linux kernel: udisks permits users to mount romfs, romfs leaks uninit memory to userspace

Reported by jannh@google.com on Thu, Aug 13, 2020, 4:57 PM EDT Project Member

# Introduction

This bug report describes a combination of two issues; together, they allow an attacker who runs in the context of an "active session on a local console" (in policykit/logind parlance) to leak pages of uninitialized memory to userspace. That's not particularly severe, but I think it points to an interesting disparity of assumptions about security guarantees.

# udisks

Several Linux distributions ship a DBus service called "udisks" (https://www.freedesktop.org/wiki/Software/udisks/) in their default configuration. Its APIs are gated using polkit checks, based on the policy <https://github.com/storaged-project/udisks/blob/master/data/org.freedesktop.UDisks2.policy.in>. Any action in that policy with "<allow_active>yes</allow_active>" can be performed by any process running inside an "active session on a local console". In a server setting, that applies to almost no processes; but in the context of a desktop system, all the processes a locally-signed-in user launches have this level of access.

In particular, udisks allows such processes to set up new loop devices and mount them. udisks restricts mount options and flags (e.g. enforcing that the mount is nosuid and nodev), but happily mounts even very esoteric file system types, as long as the kernel supports them.

As an example, you could mount a filesystem image like this:

```
    udisksctl loop-setup -f fs.img
    udisksctl mount -b /dev/disk/by-label/0123456789abcde
```

# romfs

I started browsing through random file systems that the kernel supports, looking for simple security bugs to demonstrate that the ability to mount arbitrary file systems is dangerous.

romfs is a very minimal read-only file system.
It has a helper romfs_dev_read() for reading data from the underlying block device (or MTD). The helper returns zero on success, or negative values on failure.
romfs_dev_read() silently truncates the caller-specified length based on

the filesystem size stored in the superblock:

```
int romfs_dev_read(struct super_block *sb, unsigned long pos,
                   void *buf, size_t buflen)
{
        size_t limit;

        limit = romfs_maxsize(sb);
        if (pos >= limit)
                return -EIO;
        if (buflen > limit - pos)
                buflen = limit - pos;

[...]
        if ([true])
                return romfs_blk_read(sb, pos, buf, buflen);
}
```

This means that, by crafting a filesystem whose last inode is a file of
size 0x1000, but whose data actually starts one byte before
romfs_maxsize(sb), we can make romfs_readpage() believe that
romfs_dev_read() will fill the entire page (0x1000 bytes) while it
actually only fills the first byte. This leaves the remaining 0xfff
bytes uninitialized.

# Putting it together

I have attached a PoC that, on a Debian 10 system (that's the current
Debian stable), spits out a hexdump of uninitialized memory every time
it is executed. Usage:

```
user@deb10:~/romfs-udisk$ gcc -o romfs-test romfs-test.c
user@deb10:~/romfs-udisk$ ./romfs-test
creating romfs image...
creating loop device...
Mapped file romfs.img as /dev/loop2.
mounting loop device...
Mounted /dev/loop2 at /media/user/0123456789abcde1.
testing access...
total 0
-rw-r--r-- 1 root root 4096 Jan  1  1970 foobar
dumping file contents...
00000000  58 5c 19 ec e8 55 00 00  ff ff ff ff ff ff ff ff  |X\...U..........|
00000010  02 00 00 00 00 00 00 00  30 ad 16 ec e8 55 00 00  |........0....U..|
00000020  1d 54 19 ec e8 55 00 00  02 00 00 00 ff ff ff ff  |.T...U..........|
00000030  00 00 00 00 00 00 00 00  b0 72 16 ec e8 55 00 00  |.........r...U..|
00000040  7e 72 18 ec e8 55 00 00  02 00 00 00 ff ff ff ff  |~r...U..........|
[...]
00000fe0  42 5c 19 ec e8 55 00 00  b7 02 18 ec e8 55 00 00  |B\...U.......U..|
00000ff0  30 56 19 ec e8 55 00 00  00 00 00 00 00 00 00 00  |0V...U..........|
00001000
removing mount...
Unmounted /dev/loop2.
tearing down loop device...
user@deb10:~/romfs-udisk$
```

(It doesn't work on Ubuntu 20.04.1 because they enable automatic heap zeroing.)

# Conclusion

As far as I know, so far, Linux file system maintainers have not treated
kernel memory safety bugs reachable by malicious file system images as
security issues, based on the assumption that an attacker would not
typically be able to convince the kernel to mount their malicious images.
This bug is just one example of such a bug, and I would not be surprised
at all if you could find some memory corruption that can be used to get
root privileges with more effort.

With udisks added to the mix, that assumption no longer holds.

While the romfs bug should probably be fixed, I think udisks should also
avoid exposing all this extra kernel attack surface to non-root users.
If it is necessary to allow such users to mount file system images and
such, one solution would be to use libguestfs (https://libguestfs.org/),
which will run the file system driver inside a QEMU guest and expose the
file system to the host through FUSE.

**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 2020-11-11. Disclosure at an earlier date is possible if
agreed upon by all parties.
(This language is not intended to restrict when you can ship a fix or
announce the bug publicly; it only limits when we publish our copy of
the bug report.)

romfs-test.c
2.9 KB View Download

Comment 1 by jannh@google.com on Thu, Oct 1, 2020, 10:56 PM EDT Project Member

Status: Fixed (was: New)
Labels: -Restrict-View-Commit

Upstream Linux fixed the infoleak (https://git.kernel.org/pub/scm/linux/kernel/git/torvalds/linux.git/commit/?id=2935e0a3cec1ffa558eea90db6279cff83aa3592) in:

v5.9-rc2
v5.8.4
v5.4.61
v4.19.142
v4.14.195
v4.9.234

v4.4.234

while the udisks folks see the udisks part of this as more of a hardening opportunity than a security bug. I'm marking this as fixed, but there are likely more kernel issues in that area that could be exploited through udisks.

Comment 2 by marcu...@googlemail.com on Fri, Oct 2, 2020, 5:36 AM EDT

is google requesting the CVE?

(Mitre does not allow requesting CVEs for google found isues, as this overlaps with the Google CNA.)

Comment 3 by yasho...@gmail.com on Fri, Oct 2, 2020, 6:52 AM EDT

Its's Fixed , great learning thanks

Comment 4 by jannh@google.com on Fri, Oct 2, 2020, 12:44 PM EDT Project Member

@marcusmeissner: Ugh, I hadn't realized that this was Google's turf by MITRE rules. That kinda sucks. While I don't particularly like assigning CVE identifiers to issues I find - because I fear that CVEs will not only be used to track patch propagation, but will also be abused to fast-track fixes for bugs I reported more than e.g. memory corruption fixes in response to syzkaller reports -, I guess a situation where distros aren't allowed to assign CVEs to these issues also sucks.

I'll try to make some time soon-ish to look at this more (and will probably obtain some CVEs for this and some other older issues from Google' CNA, or something like that).

Comment 5 by jannh@google.com on Mon, Nov 16, 2020, 3:33 PM EST Project Member

Labels: Fixed-2020-Aug-26

Comment 6 by jannh@google.com on Tue, Dec 1, 2020, 9:55 AM EST Project Member

Labels: CVE-2020-29371