segmentation fault in lzo_decompress_buf, stream.c 589, incomplete fix of CVE-2017-8845 and CVE-2019-10654 #163

Closed **5hadowblad3** opened this issue on Aug 26, 2020 · 18 comments

5hadowblad3 commented on Aug 26, 2020 • edited

Hi, there.

There is invalid memory access in lzo_decompress_buf, stream.c 589 in the newest branch `597be1f` .
According to the trace, it seems to be an incomplete fix of CVE-2017-8845 and CVE-2019-10654.
System:

```
DISTRIB_ID=Ubuntu
DISTRIB_RELEASE=16.04
DISTRIB_CODENAME=xenial
DISTRIB_DESCRIPTION="Ubuntu 16.04.6 LTS"
```

To reproduce, run:

```
lrzip -t poc
```

POC:
seg-stream589.zip

which is made from this original tar file compressed by lrzip
test.tar.zip

This is the output from the terminal:

```
Decompressing...
Segmentation fault
```

This is the trace reported by ASAN:

```
==177389==ERROR: AddressSanitizer: SEGV on unknown address 0x606000010000 (pc 0x7f19986a0144 bp 0x62100001cd54 sp 0x7f1994afed60 T1)
    #0 0x7f19986a0143 in lzo1x_decompress (/lib/x86_64-linux-gnu/liblzo2.so.2+0x13143)
    #1 0x43faff in lzo_decompress_buf ../stream.c:589
    #2 0x43faff in ucompthread ../stream.c:1529
    #3 0x7f199804d6b9 in start_thread (/lib/x86_64-linux-gnu/libpthread.so.0+0x76b9)
    #4 0x7f199747f41c in clone (/lib/x86_64-linux-gnu/libc.so.6+0x10741c)

AddressSanitizer can not provide additional info.
SUMMARY: AddressSanitizer: SEGV ??:0 lzo1x_decompress
Thread T1 created by T0 here:
    #0 0x7f19988e51e3 in pthread_create (/usr/lib/x86_64-linux-gnu/libasan.so.2+0x361e3)
    #1 0x451505 in create_pthread ../stream.c:133
    #2 0x451505 in fill_buffer ../stream.c:1694
    #3 0x451505 in read_stream ../stream.c:1781
    #4 0x18  (<unknown module>)

==177389==ABORTING
```

5hadowblad3 changed the title ~~segmentation fault in lzo_decompress_buf, stream.c 589~~ segmentation fault in lzo_decompress_buf, stream.c 589, incomplete fix of CVE-2017-8845 and CVE-2019-10654 on Aug 26, 2020

pete4abw commented on Aug 26, 2020 Contributor

I'm sorry. What you did was convoluted. test.tar.zip is a zip of an lrz file. This is why the zip file had 0 compression benefit.
lrzip cannot test or decompress a direct pkzip file. It just cant. seg-stream589.zip is also a zip of an lrz file. The lrz file is incomplete. Not sure what you were trying to do here. Looks like you manually edited out the stream.

You lrzip file, then zip it, then lrzip it again? To prove what? And in any event, I do not see your error.

unzip test.tar.zip --> test.tar
test.tar is an lrz file

```
peter@tommyv:/share/docs/downloads$ lrzip.631 -d -o testorig.tar test.tar
Using configuration file /home/peter/.lrzip/lrzip.conf
The following options are in effect for this DECOMPRESSION.
Threading is ENABLED. Number of CPUs detected: 8
Detected 16567353344 bytes ram
Compression level 7
Nice Value: 19
Show Progress
Verbose
Output Filename Specified: testorig.tar
Temporary Directory set as: ./
Output filename is: testorig.tar
Detected lrzip version 0.6 file.
MD5 being used for integrity testing.
Decompressing...
100%      20.00 /     20.00 KB
```

```
Average DeCompression Speed:  0.000MB/s
MD5: e460a17be4767ec46bfe7eae7424559d
Output filename is: testorig.tar: [OK] - 20480 bytes
Total time: 00:00:00.33
peter@tommyv:/share/docs/downloads$ tar -tf testorig.tar
home/heqing/c_test_case/
home/heqing/c_test_case/case1.bc
home/heqing/c_test_case/.tar.bzip2-lrz
home/heqing/c_test_case/cfg.lala1.dot
home/heqing/c_test_case/cfg.lala3.dot
home/heqing/c_test_case/case1.c
home/heqing/c_test_case/cfg.lala2.dot
home/heqing/c_test_case/cfg.main.dot
home/heqing/c_test_case/callgraph.dot
```

and to lrzip the testorig.tar file

```
peter@tommyv:/share/docs/downloads$ lrzip.631 testorig.tar
Using configuration file /home/peter/.lrzip/lrzip.conf
The following options are in effect for this COMPRESSION.
Threading is ENABLED. Number of CPUs detected: 8
Detected 16567353344 bytes ram
Compression level 7
Nice Value: 19
Show Progress
Verbose
Temporary Directory set as: ./
Compression mode is: LZMA. LZO Compressibility testing enabled
Heuristically Computed Compression Window: 105 = 10500MB
Output filename is: testorig.tar.lrz
File size: 20480
Will take 1 pass
Beginning rzip pre-processing phase
MD5: e460a17be4767ec46bfe7eae7424559d
testorig.tar - Compression Ratio: 3.861. Average Compression Speed:  0.000MB/s.
Total time: 00:00:00.08
peter@tommyv:/share/docs/downloads$ lrzip.631 -t testorig.tar.lrz
Using configuration file /home/peter/.lrzip/lrzip.conf
Threading is ENABLED. Number of CPUs detected: 8
Detected 16567353344 bytes ram
Compression level 7
Nice Value: 19
Show Progress
Verbose
Test file integrity
Temporary Directory set as: ./
Detected lrzip version 0.6 file.
MD5 being used for integrity testing.
Decompressing...
100%      20.00 /     20.00 KB
Average DeCompression Speed:  0.000MB/s
MD5: e460a17be4767ec46bfe7eae7424559d
[OK] - 20480 bytes
Total time: 00:00:00.05
```

5hadowblad3 commented on Aug 27, 2020 • edited Author

Oh, the reason why I zip these files is that GitHub does not allow uploading a tar file in the issue (but zip is ok).

To reproduce this segmentation fault, you should unzip these two files first, especially the seg-stream589.zip.

seg-stream589 is what you get from unzipping the seg-stream589.zip, which is the exploitable input.
test.tar is the original compressed file I use to craft the malformed input (seg-stream589) for helping you debug.

Specifically, the reproducing command is

```
lrzip -t seg-stream589
```

pete4abw commented on Aug 27, 2020 Contributor

Well, intentionally stripping an lrzip file is certainly not recommended. Interestingly, using `lrzip -i` refuses to progress and detects the malformed file.

```
peter@tommyv:/share/docs/downloads$ lrzip.631 -ivv seg-stream589
Using configuration file /home/peter/.lrzip/lrzip.conf
Detected lrzip version 0.6 file.
Rzip chunk 1:
Chunk byte width: 2
Chunk size: 10240
Stream: 0
Offset: 28
Block   Comp    Percent Size
1       lzo     100.0%  55 / 55 Offset: 0       Head: 65527
Offset greater than archive size, likely corrupted/truncated archive.
Fatal error - exiting
```

The routine to unpack a stream should test for a bad file like `lrzip -i` does.

pete4abw commented on Aug 27, 2020 Contributor

As a **longtime** contributor to this project, I have to say that since this is a manufactured event, I have it as a low priority. `lrzip`, whether this version or my enhanced fork has been proven useful and very stable since inception. If used as designed and intended, I think, and I know **@ckolivas** would agree, you will find `lrzip` suitable for backup purposes.

If you feel so inclined, you can examine the code of `bool get_fileinfo(rzip_control *control)` in `lrzip.c` and see if you would like to apply it to as each stream is called for decompression. I warn you, however, that corruption in a file can occur anywhere, so there could be many checks needed in different places (and it is important to note that there are already **many, many** checks).

And, defending from **intentional** corruption of a file would mean many different types of checks all throughout the program.The magic header could be changed, stream headers could be changed, invalid stream pointers could be inserted, and if the lrzip file is encrypted, there are even more issues to check for.

Good luck.

5hadowblad3 commented on Aug 27, 2020 • edited — Author

Well, this malicious input is made by my new fuzzing tool for automatically detecting/verifying incomplete fix in the program.

Since this segmentation fault is related to two CVEs that lasts for more than three years, which are caused by a series of incomplete fixes for the initial issue, I think it could cause some severe impacts if some adversaries already know the methods to exploit this bug.

Still, I appreciate the developers' efforts in making the tools better.

pete4abw commented on Aug 27, 2020 — Contributor

I read the CVE's and they are specific to `lrzip`. But the risk is not really clear. You munge a file, the program segfaults. Any application that uses LZO can be made to fail too.The fix, if you are interested, would be to traverse the entire `lrz` file stream by stream before decompression starts, and if an error is found along the way, stop and report. This is what the info function does. for large lrz files, this will add to decompression time. However, this may not work with encrypted files unless you have the password.Best to package the info checks into its own function that returns a 1 or 0 for error or no error or simply abort.

As I said, this is not a dangerous error, no malicious code can be executed since it is a read only process. I'm disinclined to pursue this.

pete4abw commented on Aug 28, 2020 — Contributor

FYI, This has nothing to do with LZO decompression. This error will occur with any mode, lzma, zpaq, gzip, bz2, etc. The bug is misplaced and tyhe CVEs incorrect, pointing to the wrong function. If you're really motivated, you can take a look here:

stream.c

```
/* fill a buffer from a stream - return -1 on failure */
static int fill_buffer(....)
...
...
        /* Check for invalid data and that the last_head is actually moving
         * forward correctly. */
        if (unlikely(c_len < 1 || u_len < 1 || last_head < 0 || (last_head && last_head <= s->last_head))) {
                fatal_return(("Invalid data compressed len %lld uncompressed %lld last_head %lld\n",
                              c_len, u_len, last_head), -1);
        }
```

This block checks for bad data in a number of ways. Because the header chain is broken, the last_head and s->last_head values are bad. But figuring out how to check on that is tough. Tough, because you don't want to exclude valid data in the checks. We've modified this in the past. Maybe it needs another look.

pete4abw commented on Aug 30, 2020 • edited — Contributor

Intentionally breaking a file can always lead to failures. Here, the fix was to check that the stream pointer did not point past the end of the current chunk. This may not catch everything, because some wacko could decide to push extraneous data anywhere in the file. Try the patch and see. This really needs to be tested thoroughly because there may be unintended consequences. I uncompressed small files and a 20GB compressed file. All seemed to work. Addresses #68

Here's some annotation.

```
        /* Check for invalid data and that the last_head is actually moving forward correctly.
         * Check for:
         *      compressed length (c_len)
         *      uncompressed length (u_len)
         *      invalid current stream pointer (last_head < 0)
         *      stream pointer extending beyond chunk (last_head > sinfo->size)
         *      stream pointer less than last stream pointer (i.e. pointing backwards!)
         */
```

Also changed the `fatal_return` call to `failure_return`, because fatal_return used `perror()` which showed SUCCESS!
stream.c_589_segfault_fix.patch.gz

5hadowblad3 commented on Aug 31, 2020 • edited — Author

Thanks for the update!

Unfortunately, I applied this patch, and the problem still exists.

This is the reproduced input (unzip first).
stream_589_seg_incomplete.zip

pete4abw commented on Aug 31, 2020 — Contributor

You found another way to break the file. This is my point. No matter what I devise, you can still come up with a way to break this. Since stream offsets are zeroed on each chunk, even checking against compressed filesize is not foolproof. I don't have more time for this. Happy coding.

pete4abw mentioned this issue on Oct 29, 2020

Fixes to Corrupt File errors and segfaults #171

Closed

ckolivas commented on Feb 13, 2021 — Owner

Fixed in git master.

ckolivas closed this as completed on Feb 13, 2021

5hadowblad3 commented on Jun 9, 2021 Author

This is assigned with CVE-2020-25467.

pete4abw mentioned this issue on Jun 9, 2021

CVEs 2020-25467 and 2021-27345 and 2021-27347 pete4abw/lrzip-next#30

Closed

ckolivas commented on Jun 9, 2021 Owner

> This is assigned with CVE-2020-25467.

Why are you commenting on this issue? It was closed.

pete4abw commented on Jun 9, 2021 Contributor

> > This is assigned with CVE-2020-25467.
>
> Why are you commenting on this issue? It was closed.

https://cve.mitre.org/cgi-bin/cvekey.cgi?keyword=Lrzip

ckolivas commented on Jun 9, 2021 Owner

That doesn't answer my question as they're all referring to an older version and all those CVEs have been fixed.

pete4abw commented on Jun 9, 2021 Contributor

> That doesn't answer my question as they're all referring to an older version and all those CVEs have been fixed.

Beats me. Boredom? **@5hadowblad3**

5hadowblad3 commented on Jun 10, 2021 • edited Author

> > This is assigned with CVE-2020-25467.
>
> Why are you commenting on this issue? It was closed.

The id is just assigned after I have submitted the application for almost a year.

They told me to update the id in the related links for public reference and ensure it is well-fixed before the full details publicized in their database.

> https://cve.mitre.org/cgi-bin/cvekey.cgi?keyword=Lrzip

This is also the reason why it cannot be found in this link yet.

carnil commented on Apr 9

For cross-reference the fixing commit should be `e74a11c`

SkewedZeppelin added a commit to Divested-Mobile/FOSS_Apps_List that referenced this issue on Jul 31

`Denote known security issues` ··· `e932b34`

Assignees

No one assigned

Labels

None yet

Projects

None yet

Milestone

No milestone

Development

No branches or pull requests

4 participants