New issue

Concurrency use-after-free between clear_rulist and lzma_decompress_buf #262

Closed

ZIIIR0 opened on Aug 12, 2025

Hi guys,

We found a concurrency uaf between the `clear_rulist` and `lzma_decompress_buf` functions in the latest master branch 1242aec.

The root cause of this vulnerability is that `sinfo->ucthreads` may be freed (in `clear_rulist`) and used (in `lzma_decompress_buf`) concurrently, leading to a use-after-free. I think it is probably due to an imcomplete fix of #206.

Reproduction

Although our tool can reproduce this vulnerability reliably, to help developers better understand the vulnerability's root cause, we'll inject several delays (e.g., `sleep()`) in the source code .

To increase the probability of a successful reproduction, we recommend inserting a delay after `dealloc(sinfo->ucthreads)` within the `clear_rulist` function in lrzip.c and another before `ucthread->s_buf = c_buf` within the `lzma_decompress_buf` function in stream.c.
Furthermore, to ensure a complete AddressSanitizer report is generated, it is highly recommended to insert an additional delay after the `clear_rulist(control)` within the `decompress_file` function.

```
// in lrzip.c, insert a delay after 'dealloc(sinfo->ucthreads)' to extend uaf window.
static void clear_rulist(rzip_control *control)
{
    while (control->ruhead) {
        struct runzip_node *node = control->ruhead;
        struct stream_info *sinfo = node->sinfo;

        dealloc(sinfo->ucthreads);
```

```
        sleep(1); // delay here!!!
        dealloc(node->pthreads);
        dealloc(sinfo->s);
        dealloc(sinfo);
        control->ruhead = node->prev;
        dealloc(node);
    }
}

// in steam.c, insert a sufficient delay before 'ucthread->s_buf = c_buf' to ensure that 'uct
static int lzma_decompress_buf(rzip_control *control, struct uncomp_thread *ucthread)
{
    ...
out:
    if (ret == -1) {
        dealloc(ucthread->s_buf);
        sleep(2); // delay here!!!
        ucthread->s_buf = c_buf;
    }
    ...
}

// in lrzip.c, insert a delay after 'clear_rulist(control)' to guarantee a full ASAN report i
bool decompress_file(rzip_control *control)
{
    ...
    if (unlikely(runzip_fd(control, fd_in, fd_hist, expected_size) < 0)) {
        clear_rulist(control);
        sleep(1);  // delay here!!!
        return false;
    }
    ...

}
```

Compile Command:

```
./autogen.sh
CC="gcc -fsanitize=address -fno-omit-frame-pointer -g -O0" CXX="g++ -fsanitize=address -fno-
make -j4
```

PoC file:

A crafted PoC is available here, please unzip first.

Run Command:

```
./lrzip -t -p2 ./PoC_UAF
```

And there is the ASAN report:

```
=================================================================
==5093==ERROR: AddressSanitizer: heap-use-after-free on address 0x6100000000a0 at pc
0x5604dee94d4e bp 0x7f68f9684c30 sp 0x7f68f9684c20
WRITE of size 8 at 0x6100000000a0 thread T3
    #0 0x5604dee94d4d in lzma_decompress_buf
/home/ziiiro/work/eval/vul_repro/lrzip/stream.c:562
    #1 0x5604dee94d4d in ucompthread /home/ziiiro/work/eval/vul_repro/lrzip/stream.c:1554
    #2 0x7f68fca92ac2 in start_thread nptl/pthread_create.c:442
    #3 0x7f68fcb2484f  (/lib/x86_64-linux-gnu/libc.so.6+0x12684f)

0x6100000000a0 is located 96 bytes inside of 192-byte region
[0x610000000040,0x610000000100)
freed by thread T0 here:
    #0 0x7f68fd080537 in __interceptor_free
../../../../src/libsanitizer/asan/asan_malloc_linux.cpp:127
    #1 0x5604dee7878c in clear_rulist /home/ziiiro/work/eval/vul_repro/lrzip/lrzip.c:786
    #2 0x5604dee7effc in decompress_file
/home/ziiiro/work/eval/vul_repro/lrzip/lrzip.c:952
    #3 0x5604dee74284 in main /home/ziiiro/work/eval/vul_repro/lrzip/main.c:720
    #4 0x7f68fca27d8f in __libc_start_call_main ../sysdeps/nptl/libc_start_call_main.h:58

previously allocated by thread T0 here:
    #0 0x7f68fd080a57 in __interceptor_calloc
../../../../src/libsanitizer/asan/asan_malloc_linux.cpp:154
    #1 0x5604dee9a472 in open_stream_in
/home/ziiiro/work/eval/vul_repro/lrzip/stream.c:1101
    #2 0x5604dee8ec42 in runzip_chunk /home/ziiiro/work/eval/vul_repro/lrzip/runzip.c:309
    #3 0x5604dee8ec42 in runzip_fd /home/ziiiro/work/eval/vul_repro/lrzip/runzip.c:387
    #4 0x5604dee7dfe3 in decompress_file
/home/ziiiro/work/eval/vul_repro/lrzip/lrzip.c:952
    #5 0x5604dee74284 in main /home/ziiiro/work/eval/vul_repro/lrzip/main.c:720
    #6 0x7f68fca27d8f in __libc_start_call_main ../sysdeps/nptl/libc_start_call_main.h:58

Thread T3 created by T0 here:
    #0 0x7f68fd024685 in __interceptor_pthread_create
../../../../src/libsanitizer/asan/asan_interceptors.cpp:216
    #1 0x5604dee9c8b6 in create_pthread
/home/ziiiro/work/eval/vul_repro/lrzip/stream.c:125
    #2 0x5604dee9c8b6 in fill_buffer /home/ziiiro/work/eval/vul_repro/lrzip/stream.c:1725
    #3 0x5604dee9c8b6 in read_stream /home/ziiiro/work/eval/vul_repro/lrzip/stream.c:1812
    #4 0x5604dee8f361 in unzip_literal /home/ziiiro/work/eval/vul_repro/lrzip/runzip.c:162
    #5 0x5604dee8f361 in runzip_chunk /home/ziiiro/work/eval/vul_repro/lrzip/runzip.c:325
    #6 0x5604dee8f361 in runzip_fd /home/ziiiro/work/eval/vul_repro/lrzip/runzip.c:387
    #7 0x5604dee7dfe3 in decompress_file
/home/ziiiro/work/eval/vul_repro/lrzip/lrzip.c:952
    #8 0x5604dee74284 in main /home/ziiiro/work/eval/vul_repro/lrzip/main.c:720
    #9 0x7f68fca27d8f in __libc_start_call_main ../sysdeps/nptl/libc_start_call_main.h:58

SUMMARY: AddressSanitizer: heap-use-after-free
/home/ziiiro/work/eval/vul_repro/lrzip/stream.c:563 in lzma_decompress_buf
Shadow bytes around the buggy address:
  0x0c207fff7fc0: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
  0x0c207fff7fd0: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
  0x0c207fff7fe0: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
  0x0c207fff7ff0: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
  0x0c207fff8000: fa fa fa fa fa fa fa fa fd fd fd fd fd fd fd fd
```

```
=>0x0c207fff8010: fd fd fd fd[fd]fd fd fd fd fd fd fd fd fd fd fd
  0x0c207fff8020: fa fa fa fa fa fa fa fa fd fd fd fd fd fd fd fd
  0x0c207fff8030: fd fd fd fd fd fd fd fd fd fd fd fd fd fd fd fd
  0x0c207fff8040: fa fa fa fa fa fa fa fa fa fa fa fa fa fa fa fa
  0x0c207fff8050: fa fa fa fa fa fa fa fa fa fa fa fa fa fa fa fa
  0x0c207fff8060: fa fa fa fa fa fa fa fa fa fa fa fa fa fa fa fa
Shadow byte legend (one shadow byte represents 8 application bytes):
  Addressable:           00
  Partially addressable: 01 02 03 04 05 06 07
  Heap left redzone:       fa
  Freed heap region:       fd
  Stack left redzone:      f1
  Stack mid redzone:       f2
  Stack right redzone:     f3
  Stack after return:      f5
  Stack use after scope:   f8
  Global redzone:          f9
  Global init order:       f6
  Poisoned by user:        f7
  Container overflow:      fc
  Array cookie:            ac
  Intra object redzone:    bb
  ASan internal:           fe
  Left alloca redzone:     ca
  Right alloca redzone:    cb
  Shadow gap:              cc
==5093==ABORTING
```

I believe this concurrency-related memory corruption vulnerability could have serious consequences, for example, with a carefully crafted exploit it could allow arbitrary command execution. So I recommend fixing it as soon as possible.

Thanks

ZIIIR0 mentioned this on Aug 12, 2025

Concurrency null-pointer-dereference in ucompthread() of stream.c #263

ckolivas on Feb 12 — Owner

To be clear, this issue only occurs on lzma_decompress_buf failing I assume? Investigating, thanks.

ckolivas on Feb 12 — Owner

Whilst not a perfect solution, this has been addressed in `96931e7`

ckolivas closed this as completed on Feb 12

Metadata

Assignees

No one assigned

Labels

No labels

Projects

No projects

Milestone

No milestone

Relationships

None yet

Development

No branches or pull requests

Participants