New issue

Jump to bottom

Multiple concurrency UAF bug between `zpaq_decompress_buf()` and `clear_rulist()` function #206

Closed wcventure opened this issue on Aug 2, 2021 · 4 comments

wcventure commented on Aug 2, 2021

Dear all,

Our tool report that there would be multiple concurrency use-after-free between `zpaq_decompress_buf()` function and `clear_rulist()` function, in the newest master branch `465afe8` .

Brief Explanation

The related code simplified from `stream.c` and `runzip.c` are shown as follow:

```
// Thread T0                                    | // Thread T1
// in clear_rulist() in runzip.c                | // in zpaq_decompress_buf() in steam.c
...                                             | ...
struct runzip_node *node = control->ruhead;     | if (unlikely(dlen != ucthread->u_len)) {
struct stream_info *sinfo = node->sinfo;        |     ret = -1;
                                                | } else
dealloc(sinfo->ucthreads);                      |     dealloc(c_buf);
dealloc(sinfo);                                 | out:
                                                |     if (ret == -1) {
                                                |         dealloc(ucthread->s_buf);
                                                |         ucthread->s_buf = c_buf;
                                                |     }
```

Both thread T0 and thread T1 operate on a shared variable `ucthread` (i.e., T0 dealloc the a ucthread through `dealloc(sinfo->ucthreads);` , and T1 use the ucthread in all statements `if (unlikely(dlen != ucthread->u_len))` , `dealloc(ucthread->s_buf);` , and `ucthread->s_buf = c_buf;`).
However, a use-after-free can occur if the deallocation of ucthread before the use of ucthread.
For example, the following three thread interleaving can trigger three different UAFs:

Interleaving (a)

```
// Thread T0                                    | // Thread T1
                                                |
...                                             |
struct runzip_node *node = control->ruhead;     |
struct stream_info *sinfo = node->sinfo;        |
                                                |
dealloc(sinfo->ucthreads);                      |
dealloc(sinfo);                                 |
------------------------------------------------------------------------------------------------
                                                | ...
                                                | if (unlikely(dlen != ucthread->u_len)) { // UAF here
                                                |     ret = -1;
                                                | } else
                                                |     dealloc(c_buf);
                                                | out:
                                                |     if (ret == -1) {
                                                |         dealloc(ucthread->s_buf);
                                                |         ucthread->s_buf = c_buf;
                                                |     }
```

Interleaving (b)

```
// Thread T0                                    | // Thread T1
                                                |
                                                | ...
                                                | if (unlikely(dlen != ucthread->u_len)) {
                                                |     ret = -1;
                                                | } else
                                                |     dealloc(c_buf);
                                                | out:
                                                |     if (ret == -1) {
------------------------------------------------------------------------------------------------
...                                             |
struct runzip_node *node = control->ruhead;     |
struct stream_info *sinfo = node->sinfo;        |
                                                |
dealloc(sinfo->ucthreads);                      |
dealloc(sinfo);                                 |
------------------------------------------------------------------------------------------------
                                                |         dealloc(ucthread->s_buf);  // UAF occur here
                                                |         ucthread->s_buf = c_buf;
                                                |     }
```

Interleaving (c)

```
// Thread T0                                    | // Thread T1
                                                |
                                                | ...
```

```
                                        | if (unlikely(dlen != ucthread->u_len)) {
                                        |     ret = -1;
                                        | } else
                                        |     dealloc(c_buf);
                                        | out:
                                        |     if (ret == -1) {
                                        |         dealloc(ucthread->s_buf);
-------------------------------------------------------------------------------------------------
...                                     |
struct runzip_node *node = control->ruhead; |
struct stream_info *sinfo = node->sinfo;    |
                                        |
dealloc(sinfo->ucthreads);              |
dealloc(sinfo);                         |
-------------------------------------------------------------------------------------------------
                                        |         ucthread->s_buf = c_buf; // UAF occur here
                                        |     }
```

Reproduce through delay injection

To reproduce those use-after-free errors, we can insert two delays (e.g., `sleep(1)`) into the original source code.

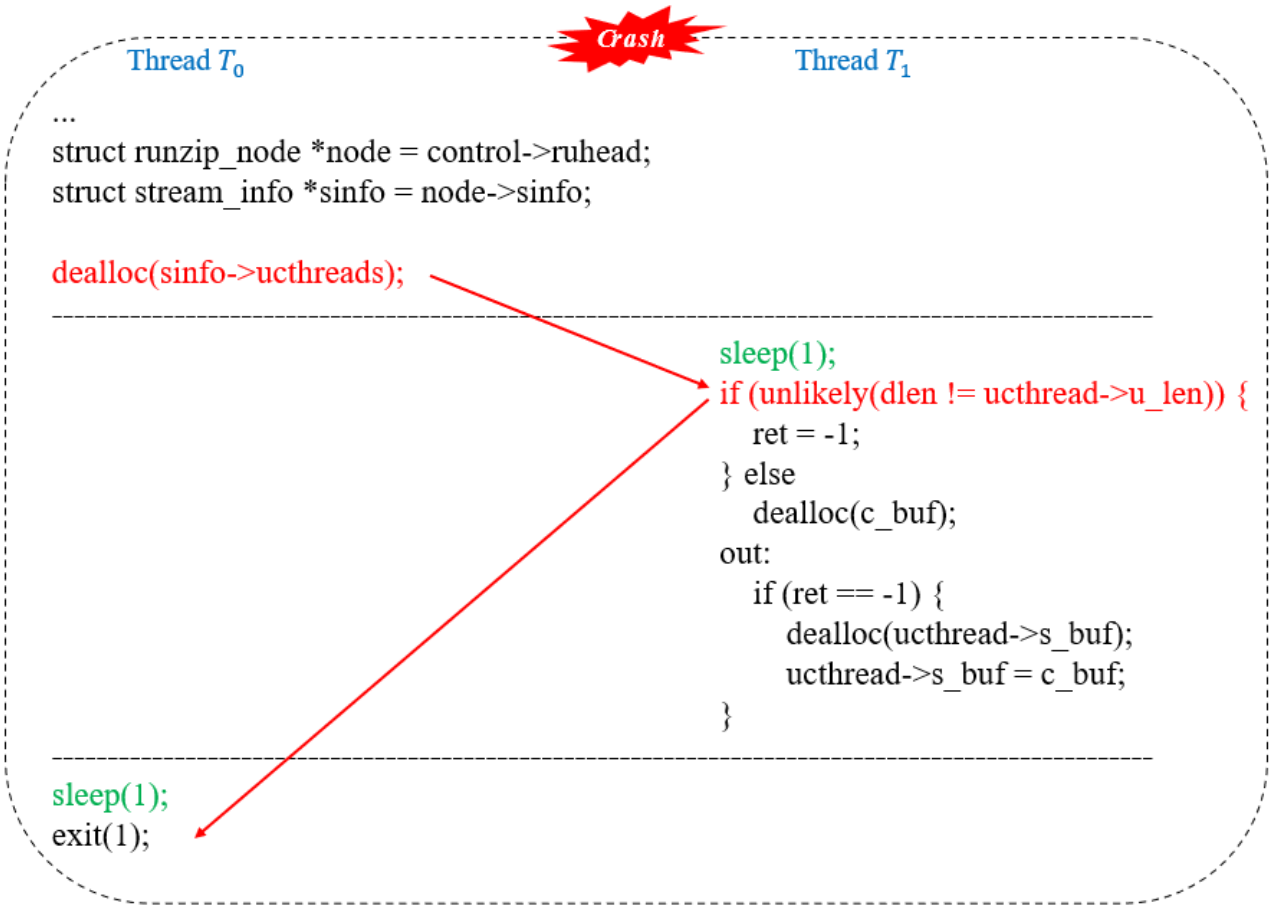
For example, to reproduce interleaving (a) as mentioned earlier, you can insert a delay before `dealloc(sinfo->ucthreads);` statement in function in `steam.c`, and also a delay after, as shown as follows.

```
// In runzip.c, insert a delay after `dealloc(sinfo->ucthreads);`
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 delay after `dealloc(sinfo->ucthreads);`
static int zpaq_decompress_buf(rzip_control *control __UNUSED__, struct uncomp_thread *ucthread, long
{
        ...
        zpaq_decompress(ucthread->s_buf, &dlen, c_buf, ucthread->c_len,
                        control->msgout, SHOW_PROGRESS ? true: false, thread);
        sleep(1); // delay here !!!!!!!!!!
        if (unlikely(dlen != ucthread->u_len)) {
```

```
                print_err("Inconsistent length after decompression. Got %ld bytes, expected %lld\n",
                ret = -1;
        } else
                dealloc(c_buf);
    ...
}
```



compile the program:

```
CC=gcc CXX=g++ CFLAGS="-g -O0 -fsanitize=address" CXXFLAGS="-g -O0 -fsanitize=address" ./configure --
make
```

Download the testcase (I upload the POC here, please unzip first).
POC.zip

Run with the testcase with the following command:

```
./lrzip -t -p2 POC
```

Then, you will see the use-after-free bug report. Here is the trace reported by ASAN:

```
=================================================================
==33325==ERROR: AddressSanitizer: heap-use-after-free on address 0x61d0000000e8 at pc 0x000000512b00

READ of size 8 at 0x61d0000000e8 thread T3
    #0 0x512aff in zpaq_decompress_buf /workdir/lrzip/stream.c:449:6
    #1 0x510381 in ucompthread /workdir/lrzip/stream.c:1554:11
    #2 0x7f9a3541b6da in start_thread (/lib/x86_64-linux-gnu/libpthread.so.0+0x76da)
    #3 0x7f9a3479771e in clone (/lib/x86_64-linux-gnu/libc.so.6+0x12171e)

0x61d0000000e8 is located 104 bytes inside of 2400-byte region [0x61d000000080,0x61d0000009e0)
freed by thread T0 here:
    #0 0x494e1d in free /home/brian/src/final/llvm-project/compiler-rt/lib/asan/asan_malloc_linux.cpp
    #1 0x4faa0d in clear_rulist /workdir/lrzip/runzip.c:255:3
    #2 0x4f7ab2 in runzip_chunk /workdir/lrzip/runzip.c:384:2
    #3 0x4f4aae in runzip_fd /workdir/lrzip/runzip.c:404:7
    #4 0x4d84ce in decompress_file /workdir/lrzip/lrzip.c:845:6
    #5 0x4cb98b in main /workdir/lrzip/main.c:706:4
    #6 0x7f9a34697bf6 in __libc_start_main (/lib/x86_64-linux-gnu/libc.so.6+0x21bf6)

previously allocated by thread T0 here:
    #0 0x495212 in calloc /home/brian/src/final/llvm-project/compiler-rt/lib/asan/asan_malloc_linux.c
    #1 0x5003e1 in open_stream_in /workdir/lrzip/stream.c:1084:33
    #2 0x4f6fa5 in runzip_chunk /workdir/lrzip/runzip.c:322:7
    #3 0x4f4aae in runzip_fd /workdir/lrzip/runzip.c:404:7
    #4 0x4d84ce in decompress_file /workdir/lrzip/lrzip.c:845:6
    #5 0x4cb98b in main /workdir/lrzip/main.c:706:4
    #6 0x7f9a34697bf6 in __libc_start_main (/lib/x86_64-linux-gnu/libc.so.6+0x21bf6)

Thread T3 created by T0 here:
    #0 0x47fe4a in pthread_create /home/brian/src/final/llvm-project/compiler-rt/lib/asan/asan_interc
    #1 0x4fbbd0 in create_pthread /workdir/lrzip/stream.c:125:6
    #2 0x507033 in fill_buffer /workdir/lrzip/stream.c:1713:6
    #3 0x504184 in read_stream /workdir/lrzip/stream.c:1800:8
    #4 0x4f9c88 in unzip_literal /workdir/lrzip/runzip.c:162:16
    #5 0x4f731c in runzip_chunk /workdir/lrzip/runzip.c:338:9
    #6 0x4f4aae in runzip_fd /workdir/lrzip/runzip.c:404:7
    #7 0x4d84ce in decompress_file /workdir/lrzip/lrzip.c:845:6
    #8 0x4cb98b in main /workdir/lrzip/main.c:706:4
    #9 0x7f9a34697bf6 in __libc_start_main (/lib/x86_64-linux-gnu/libc.so.6+0x21bf6)

SUMMARY: AddressSanitizer: heap-use-after-free /workdir/lrzip/stream.c:449:6 in zpaq_decompress_buf
Shadow bytes around the buggy address:
  0x0c3a7fff7fc0: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
  0x0c3a7fff7fd0: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
  0x0c3a7fff7fe0: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
  0x0c3a7fff7ff0: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
  0x0c3a7fff8000: fa fa fa fa fa fa fa fa fa fa fa fa fa fa fa fa
=>0x0c3a7fff8010: fd fd fd fd fd fd fd fd fd fd fd fd fd[fd]fd fd
  0x0c3a7fff8020: fd fd fd fd fd fd fd fd fd fd fd fd fd fd fd fd
  0x0c3a7fff8030: fd fd fd fd fd fd fd fd fd fd fd fd fd fd fd fd
  0x0c3a7fff8040: fd fd fd fd fd fd fd fd fd fd fd fd fd fd fd fd
  0x0c3a7fff8050: fd fd fd fd fd fd fd fd fd fd fd fd fd fd fd fd
  0x0c3a7fff8060: fd fd fd fd fd fd fd fd fd fd fd fd fd fd fd fd
```

```
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
==33325==ABORTING
```

I'm not sure if these use-after-free bugs could cause serious harm. I hope you can check whether it is necessary to fix these bugs.

Thanks.

pete4abw commented on Aug 3, 2021 — Contributor

Totally broken file.I use a validation function prior to decompress or test. It will catch this broken file before it even starts. Interesting academic exercise, but not worth it, IMHO.

```
$ ps2lrz -i POC
Showing file info only
POC is an lrzip version 0.6 file
POC is not encrypted
POC uncompressed file size is 5476377696771506395 bytes
Dumping magic header 24 bytes
Byte Offset      Description/Content
===========      ==================
Magic Bytes 0-3: 4C 52 5A 49 LRZI
Bytes 4-5:       LRZIP Major, Minor version: 00, 06
Bytes 6-13:      LRZIP Uncompressed Size bytes: DB 00 F0 07 80 00 00 4C
Bytes 14 and 15: unused
Bytes 16-20:     LZMA Properties Bytes; 49 00 00 00 1B lc=1, lp=3, pb=1, Dictionary Size=452984832
Byte  21:        MD5 Sum at EOF: yes
Byte  22:        File is encrypted: no
Byte  23:        unused
```

wcventure commented on Aug 3, 2021 Author

Actually, the uploaded file is a well-crafted input. Given an illegal input file, concurrency use-after-free may occur. Because once the program failed to read chunk_bytes size in runzip_chunk function, the program will release the resources (i.e., free) and terminate. However, other threads still could access the released resources before termination.

pete4abw commented on Aug 5, 2021 Contributor

All decompress functions perform the same way. As **@ckolivas** likes to say, patches welcome. `if (!ptr) fatal` ...

```
 431 static int zpaq_decompress_buf(rzip_control *control __UNUSED__, struct uncomp_thread
*ucthread, long thread)
. . .
 449         if (unlikely(dlen != ucthread->u_len)) {
 450                 print_err("Inconsistent length after decompression. Got %ld bytes, expected
%lld\n", dlen, ucthread->u_len);
 451                 ret = -1;
 452         } else
 453                 dealloc(c_buf);
. . .
 462 static int bzip2_decompress_buf(rzip_control *control __UNUSED__, struct uncomp_thread
*ucthread)
. . .
 483         if (unlikely(dlen != ucthread->u_len)) {
 484                 print_err("Inconsistent length after decompression. Got %d bytes, expected
%lld\n", dlen, ucthread->u_len);
 485                 ret = -1;
 486         } else
 487                 dealloc(c_buf);
. . .
 496 static int gzip_decompress_buf(rzip_control *control __UNUSED__, struct uncomp_thread
*ucthread)
. . .
 517         if (unlikely((i64)dlen != ucthread->u_len)) {
 518                 print_err("Inconsistent length after decompression. Got %ld bytes, expected
%lld\n", dlen, ucthread->u_len);
 519                 ret = -1;
 520         } else
 521                 dealloc(c_buf);
. . .
 530 static int lzma_decompress_buf(rzip_control *control, struct uncomp_thread *ucthread)
. . .
 554         if (unlikely((i64)dlen != ucthread->u_len)) {
 555                 print_err("Inconsistent length after decompression. Got %lld bytes, expected
%lld\n", (i64)dlen, ucthread->u_len);
 556                 ret = -1;
 557         } else
```

```
 558                 dealloc(c_buf);
. . .
 567 static int lzo_decompress_buf(rzip_control *control __UNUSED__, struct uncomp_thread
*ucthread)
. . .
 588         if (unlikely((i64)dlen != ucthread->u_len)) {
 589                 print_err("Inconsistent length after decompression. Got %lu bytes, expected
%lld\n", (unsigned long)dlen, ucthread->u_len);
 590                 ret = -1;
 591         } else
 592                 dealloc(c_buf);
```

ckolivas commented on Feb 25 — Owner

Fixed in 4b39421 , thanks.

ckolivas closed this as completed on Feb 25

Assignees

No one assigned

Labels

None yet

Projects

None yet

Milestone

No milestone

Development

No branches or pull requests

3 participants