

ldns heap Out-of-bound Read

1. Version:

1.7.1

2. Vulnerability detect

The fuzz info is as follow:

INFO: Seed: 499159114

INFO: Loaded 2 modules (7085 inline 8-bit counters): 6823 [0x7f28727f5d60, 0x7f28727f7807), 262 [0x374d20, 0x374e26),

INFO: Loaded 2 PC tables (7085 PCs): 6823 [0x7f28727f7808,0x7f2872812278), 262 [0x374e28,0x375e88),

../ldns-verify-zone: Running 1 inputs 1 time(s) each.

Running: ./w16wcrash-b6944107c6d8af77d22189cdb7f2b18e7ef8a188

=====

===

==18590==ERROR: AddressSanitizer: heap-buffer-overflow on address 0x616000001192 at pc 0x7f287278a0ae bp 0x7fff1aa7ef90 sp 0x7fff1aa7ef88

READ of size 1 at 0x616000001192 thread T0

#0 0x7f287278a0ad in ldns_rr_new_frm_str_internal
/home/work/build-ldns171-with-clang/./ldns-1.7.1/rr.c:368:11

#1 0x7f2872788973 in ldns_rr_new_frm_str
/home/work/build-ldns171-with-clang/./ldns-1.7.1/rr.c:669:9

#2 0x7f287278c605 in ldns_rr_new_frm_fp_l
/home/work/build-ldns171-with-clang/./ldns-1.7.1/rr.c:780:8

#3 0x7f28727d59b7 in ldns_zone_new_frm_fp_l
/home/work/build-ldns171-with-clang/./ldns-1.7.1/zone.c:227:7

#4 0x7f2872725093 in ldns_dnssec_zone_new_frm_fp_l
/home/work/build-ldns171-with-clang/./ldns-1.7.1/dnssec_zone.c:645:11

#5 0x36637a in real_main
/home/work/build-ldns171-with-clang/./ldns-1.7.1/examples/ldns-verify-zone.c:790:6

#6 0x367c87 in LLVMFuzzerTestOneInput
/home/work/build-ldns171-with-clang/./ldns-1.7.1/examples/ldns-verify-zone.c:790:6

e.c:879:13

#7 0x26f44a in fuzzer::Fuzzer::ExecuteCallback(unsigned char const*,
unsigned long)

(/home/work/build-ldns171-with-clang/fuzz/ldns-verify-zone+0x26f44a)

#8 0x25e737 in fuzzer::RunOneTest(fuzzer::Fuzzer*, char const*, unsigned
long)

(/home/work/build-ldns171-with-clang/fuzz/ldns-verify-zone+0x25e737)

#9 0x264661 in fuzzer::FuzzerDriver(int*, char***, int (*)(unsigned char
const*, unsigned long))

(/home/work/build-ldns171-with-clang/fuzz/ldns-verify-zone+0x264661)

#10 0x28afe2 in main
(/home/work/build-ldns171-with-clang/fuzz/ldns-verify-zone+0x28afe2)

#11 0x7f2870b69b96 in __libc_start_main
(/lib/x86_64-linux-gnu/libc.so.6+0x21b96)

#12 0x25d029 in _start
(/home/work/build-ldns171-with-clang/fuzz/ldns-verify-zone+0x25d029)

0x616000001192 is located 0 bytes to the right of 530-byte region
[0x61600000f80,0x616000001192)

allocated by thread T0 here:

#0 0x3368d3 in malloc
(/home/work/build-ldns171-with-clang/fuzz/ldns-verify-zone+0x3368d3)

#1 0x7f28726f47db in ldns_buffer_new_frm_data
/home/work/build-ldns171-with-clang/./ldns-1.7.1/buffer.c:48:18

#2 0x7f287278954d in ldns_rr_new_frm_str_internal
/home/work/build-ldns171-with-clang/./ldns-1.7.1/rr.c:260:2

#3 0x7f2872788973 in ldns_rr_new_frm_str
/home/work/build-ldns171-with-clang/./ldns-1.7.1/rr.c:669:9

#4 0x7f287278c605 in ldns_rr_new_frm_fp_l
/home/work/build-ldns171-with-clang/./ldns-1.7.1/rr.c:780:8

#5 0x7f28727d59b7 in ldns_zone_new_frm_fp_l
/home/work/build-ldns171-with-clang/./ldns-1.7.1/zone.c:227:7

#6 0x7f2872725093 in ldns_dnssec_zone_new_frm_fp_l

```

/home/work/build-ldns171-with-clang/./ldns-1.7.1/dnssec_zone.c:645:11
#7          0x36637a          in          real_main
/home/work/build-ldns171-with-clang/./ldns-1.7.1/examples/ldns-verify-zone.c:790:6
#8          0x367c87          in          LLVMFuzzerTestOneInput
/home/work/build-ldns171-with-clang/./ldns-1.7.1/examples/ldns-verify-zone.c:879:13
#9 0x26f44a in fuzzer::Fuzzer::ExecuteCallback(unsigned char const*,
unsigned long)
(/home/work/build-ldns171-with-clang/fuzz/ldns-verify-zone+0x26f44a)
#10 0x25e737 in fuzzer::RunOneTest(fuzzer::Fuzzer*, char const*, unsigned
long)
(/home/work/build-ldns171-with-clang/fuzz/ldns-verify-zone+0x25e737)
#11 0x264661 in fuzzer::FuzzerDriver(int*, char***, int (*)(unsigned char
const*, unsigned long))
(/home/work/build-ldns171-with-clang/fuzz/ldns-verify-zone+0x264661)
#12          0x28afe2          in          main
(/home/work/build-ldns171-with-clang/fuzz/ldns-verify-zone+0x28afe2)
#13          0x7f2870b69b96     in          __libc_start_main
(/lib/x86_64-linux-gnu/libc.so.6+0x21b96)

```

```

SUMMARY:          AddressSanitizer:          heap-buffer-overflow
/home/work/build-ldns171-with-clang/./ldns-1.7.1/rr.c:368:11          in
ldns_rr_new_frm_str_internal

```

Shadow bytes around the buggy address:

```

0x0c2c7fff81e0: fa fa fa fa fa fa fa fa fa fa fa fa fa fa fa fa
0x0c2c7fff81f0: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
0x0c2c7fff8200: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
0x0c2c7fff8210: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
0x0c2c7fff8220: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
=>0x0c2c7fff8230: 00 00[02]fa fa fa fa fa fa fa fa fa fa fa fa fa
0x0c2c7fff8240: fa fa fa fa fa fa fa fa fa fa fa fa fa fa fa fa
0x0c2c7fff8250: fd fd fd fd fd fd fd fd fd fd fd fd fd fd fd fd

```

0x0c2c7fff8260: fd fd fd fd fd fd fd fd fd fd fd fd fd fd fd fd

0x0c2c7fff8270: fd fd fd fd fd fd fd fd fd fd fd fd fd fd fd fd

0x0c2c7fff8280: 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

==18590==ABORTING

3. Analysis:

From the log of libfuzzer, the problem lies in the `ldns_rr_new_frm_str_internal` function of `rr.c`, offset by 368 lines, as follows:

```
365         while (ldns_buffer_remaining(rd_buf) > 0) {
366
367             /* skip spaces */
368             while (*(ldns_buffer_current(rd_buf)) == ' ') {
369                 ldns_buffer_skip(rd_buf, 1);
370             }
371         }
```

Here, when the rd_buf is read, an out-of-bounds access occurs. As long as rd_buf reads out '', the ldns_buffer_skip is called to perform the position offset 1. The definition of ldns_buffer_skip is as follows:

```
C buffer.h x C rr.c C server.c C ldns-testns.c C ldns-verify-z
ldns > C buffer.h > ldns_buffer_skip(ldns_buffer *, ssize_t)
170  INLINE void
171  ldns_buffer_set_position(ldns_buffer *buffer, ssize_t mark)
172  {
173      assert(mark <= buffer->_limit);
174      buffer->_position = mark;
175  }
176
177  /**
178   * changes the buffer's position by COUNT bytes. The position must
179   * be moved behind the buffer's limit or before the beginning of the
180   * buffer.
181   * \param[in] buffer the buffer
182   * \param[in] count the count to use
183   */
184  INLINE void
185  ldns_buffer_skip(ldns_buffer *buffer, ssize_t count)
186  {
187      assert(buffer->_position + count <= buffer->_limit);
188      buffer->_position += count;
189  }
190
191  /**
```

Can see that the offset of position is count. Follow the ldns_buffer_current function and find the following definition in line 293 of buffer.h:

```
292  INLINE uint8_t *
293  ldns_buffer_current(const ldns_buffer *buffer)
294  {
295      return ldns_buffer_at(buffer, buffer->_position); // 越界读
296  }
297
298  /**
```

It can be parsed by calling ldns_buffer_at, which is located in line 257 of buffer.h:

```
255  */
256  INLINE uint8_t *
257  ldns_buffer_at(const ldns_buffer *buffer, size_t at) // 漏洞函数
258  {
259      assert(at <= buffer->_limit); // 边界校验问题
260      return buffer->_data + at;
261  }
262
263
```

Here we see a problem when checking the boundary, comparing at with buffer->_limit, throwing an exception if it is greater, or returning the offset byte if it is less than or equal to. Leading to a 1 byte access, we continue to return to the ldns_rr_new_frm_str_internal function to see how the buffer is initialized. Here we find the initialization part of rd_buf, in line 260 of rr.c:

```
248  rr_buf, type, (LDNS_RR_TYPE, LDNS_SYNTAX_DATALEN)) {
249
250      status = LDNS_STATUS_SYNTAX_TYPE_ERR;
251      goto error;
252  }
253  }
254
255  if (ldns_bget_token(rr_buf, rdata, "\0", LDNS_MAX_PACKETLEN) == -1) {
256      /* apparently we are done, and it's only a question RR
257       * so do not set status and go to ldnserror here
258       */
259  }
260  ldns_buffer_new_frm_data(rd_buf, rdata, strlen(rdata)); // 从rdata里读入到rd_buf里
261
```

See here is to read rdata into rd_buf, follow up the function, in line 41 of buffer.c:

```
void
ldns_buffer_new_frm_data(ldns_buffer *buffer, const void *data, size_t size)
{
    assert(data != NULL);

    buffer->_position = 0; // 位置
    buffer->_limit = buffer->_capacity = size; // 结束
    buffer->_fixed = 0;
    buffer->_data = LDNS_XMALLOC(uint8_t, size);
    if(!buffer->_data) {
        buffer->_status = LDNS_STATUS_MEM_ERR;
        return;
    }
    memcpy(buffer->_data, data, size);
    buffer->_status = LDNS_STATUS_OK;

    ldns_buffer_invariant(buffer);
}

// Diagram annotations:
// - Red arrow from 'data' parameter to 'data' in memcpy
// - Red arrow from 'size' parameter to 'size' in memcpy
// - Red box around buffer initialization lines
// - Red box around memcpy line
// - Red box around error handling lines
```

Here we see that the `_limit` parameter is the size of the buffer, so it goes back to the vulnerability location.

```
255  */
256  INLINE uint8_t *
257  ldns_buffer_at(const ldns_buffer *buffer, size_t at) // 漏洞函数
258  {
259      assert(at <= buffer->_limit); // 边界校验问题
260      return buffer->_data + at;
261  }
262
263  /*
```

Here, when `at == buffer->_limit`, the returned value is `buffer[buffer->_limit]`, that is, reading one byte out of bounds. Continue to find the source of the variable `rdata` in `ldns_rr_new_frm_str_internal`, the source is as follows:

```
254
255      if (ldns_bget_token(rr_buf, rdata, "\\0", LDNS_MAX_PACKETLEN) == -1) {
256          /* apparently we are done, and it's only a question RR
257             * so do not set status and go to ldnserror here
258             */
259      }
```

It is read by `rr_buf` through the `ldns_bget_token` function, which is an analytic function, and `rr_buf` is assigned as follows:

```
177
178      ldns_buffer_new_frm_data(rr_buf, (char*)str, strlen(str));
179
180      /* split the rr in its parts -1 signals trouble */
181      if (ldns_bget_token(rr_buf, owner, "\\t\\n ", LDNS_MAX_DOMAINLEN) == -1){
182
183          status = LDNS_STATUS_SYNTAX_ERR;
184          goto error;
```

It is assigned by `str`, and `str` is the second parameter of `ldns_rr_new_frm_str_internal` as follows:

```
111  */
112  static ldns_status
113  ldns_rr_new_frm_str_internal(ldns_rr **newrr, const char *str,
114                             uint32_t default_ttl, const ldns_rdf *origin,
115                             ldns_rdf **prev, bool question)
116  {
117      ldns_rr *new;
118      const ldns_rr_descriptor *desc;
119      ldns_rr_type rr_type;
120      ldns_buffer *rr_buf = NULL;
121      ldns_buffer *rd_buf = NULL;
122      uint32_t ttl_val;
123      char *owner = NULL;
124      char *ttl = NULL;
125      ldns_rr_class clas_val;
126      char *clas = NULL;
```

View the caller of this function:

```
C buffer.h  C rr.c  x  C parse.c  C dnssec.c  C buffer.c
C rr.c > ...
661  return status;
662  }
663
664  ldns_status
665  ldns_rr_new_frm_str(ldns_rr **newrr, const char *str,
666                    uint32_t default_ttl, const ldns_rdf *origin,
667                    ldns_rdf **prev)
668  {
669      return ldns_rr_new_frm_str_internal(newrr,
670                                         str,
671                                         default_ttl,
672                                         origin,
673                                         prev,
674                                         false);
675  }
676
677  ldns_status
678  ldns_rr_new_question_frm_str(ldns_rr **newrr, const char *str, // 第二个参数
679                              const ldns_rdf *origin, ldns_rdf **prev)
680  {
681      return ldns_rr_new_frm_str_internal(newrr,
682                                         str,
683                                         0,
684                                         origin,
685                                         prev,
686                                         true);
687  }
688
```

Look at the caller `ldns_rr_new_frm_str` function, in `ldns-verify-zon.c`:


```
C rr.c x C server.c C ldns-testns.c C ldns-verify-zone.c C rr.h C dnssec.c
C rr.c > ...
769         }
770         s = LDNS_STATUS_SYNTAX_TTL;
771     } else if (strncmp(line, "$INCLUDE", 8) == 0) {
772         s = LDNS_STATUS_SYNTAX_INCLUDE;
773     } else if (!*ldns_strip_ws(line)) {
774         LDNS_FREE(line);
775         return LDNS_STATUS_SYNTAX_EMPTY;
776     } else {
777         if (origin && *origin) {
778             s = ldns_rr_new_frm_str(&rr, (const char*) line, ttl, *origin, prev);
779         } else {
780             s = ldns_rr_new_frm_str(&rr, (const char*) line, ttl, NULL, prev);
781         }
782     }
783     LDNS_FREE(line);
```

Its caller is: `ldns_rr_new_frm_fp_l`, which is called in `ldns-verify-zone.c` as follows:

```
ldns-verify-zone.c — ldns-develop
C rr.c C server.c C ldns-testns.c C ldns-verify-zone.c x C rr.h C
Users > qulewei > Desktop > C ldns-verify-zone.c > ...
52         fprintf(stream, "%s", de);
53     } else {
54         fprintf(stream, "TYPE%u", type);
55     }
56 }
57
58 static ldns_status
59 read_key_file(const char *filename, ldns_rr_list *keys)
60 {
61     ldns_status status = LDNS_STATUS_ERR;
62     ldns_rr *rr;
63     FILE *fp;
64     uint32_t my_ttl = 0;
65     ldns_rdf *my_origin = NULL;
66     ldns_rdf *my_prev = NULL;
67     int line_nr;
68
69     if (!(fp = fopen(filename, "r"))) {
70         return LDNS_STATUS_FILE_ERR;
71     }
72     while (!feof(fp)) {
73         status = ldns_rr_new_frm_fp_l(&rr, fp, &my_ttl, &my_origin,
74                                     &my_prev, &line_nr);
75     }
76     if (status == LDNS_STATUS_OK) {
```

That is, the final payload is the zone file. In summary: the process is as follows:

- a) `ldns_rr_new_frm_fp_l` read into the zone file for parsing;
- b) the parameter is passed to the `ldns_rr_new_frm_str` function;

- c) The parameter is assigned to str by the ldns_rr_new_frm_str_internal function;
- d) assign str to rr_buf via the ldns_buffer_new_frm_data function;
- e) assign rr_buf to rdata via the ldns_bget_token function;
- f) Use ldns_buffer_new_frm_data again to assign rdata to rd_buf
- g) Finally, the ldns_buffer_current is called in the loop, and the out-of-bounds access occurs when ldns_buffer_skip is performed.

5. Repair plan

The judgment is made in ldns_rr_new_frm_str_internal as follows:

```

/* skip spaces */
while (ldns_buffer_position(rd_buf) < ldns_buffer_limit(rd_buf) &&
*(ldns_buffer_current(rd_buf)) == ' ') {
ldns_buffer_skip(rd_buf, 1);
}
if (ldns_buffer_position(rd_buf) < ldns_buffer_limit(rd_buf) &&
*(ldns_buffer_current(rd_buf)) == '\\"') {
delimiters = "\\\"0";
ldns_buffer_skip(rd_buf, 1);
quoted = true;
} else if (ldns_rr_descriptor_field_type(desc, r_cnt)
== LDNS_RDF_TYPE_LONG_STR) {
status = LDNS_STATUS_SYNTAX_RDATA_ERR;
goto error;
}

```

If you want to solve the remaining problems, it is recommended to modify the constraint relationship of _position, _limit, _capacity in buffer.h.