Global-Buffer-Overflow in function dwarf::line_table::line_table at dwarf/line.cc:107 #48

Open **xiaoxiongwang** opened this issue on Aug 15, 2020 · 1 comment

xiaoxiongwang commented on Aug 15, 2020 • edited

Tested in Ubuntu 16.04, 64bit.

The tested program is the example program dump_line.

The testcase is dump_line_global_buffer_overflow.

I use the following command:

```
/path-to-libelfin/examples/dump-lines dump_line_global_buffer_overflow
```

and get:

```
terminate called after throwing an instance of 'dwarf::format_error'
  what():  expected 858944595 arguments for line number opcode 16, got 2
--- <0>
Aborted (core dumped)
```

I use **valgrind** to analysis the bug and get the below information (absolute path information omitted):

```
valgrind /path-to-libelfin/examples/dump-lines dump_line_global_buffer_overflow
==9235== Memcheck, a memory error detector
==9235== Copyright (C) 2002-2015, and GNU GPL'd, by Julian Seward et al.
==9235== Using Valgrind-3.11.0 and LibVEX; rerun with -h for copyright info
==9235== Command: /path-to-libelfin/examples/dump-lines dump_line_global_buffer_overflow
==9235==
terminate called after throwing an instance of 'dwarf::format_error'
  what():  expected 858944595 arguments for line number opcode 16, got 2
--- <0>
==9235==
==9235== Process terminating with default action of signal 6 (SIGABRT)
==9235==    at 0x546A428: raise (raise.c:54)
==9235==    by 0x546C029: abort (abort.c:89)
==9235==    by 0x4ED3DDD: ??? (in /usr/lib/x86_64-linux-gnu/libstdc++.so.6.0.28)
==9235==    by 0x4EDF895: ??? (in /usr/lib/x86_64-linux-gnu/libstdc++.so.6.0.28)
==9235==    by 0x4EDF900: std::terminate() (in /usr/lib/x86_64-linux-gnu/libstdc++.so.6.0.28)
==9235==    by 0x4EDFB54: __cxa_throw (in /usr/lib/x86_64-linux-gnu/libstdc++.so.6.0.28)
==9235==    by 0x48226F: dwarf::line_table::line_table(std::shared_ptr<dwarf::section> const&, unsigned long, unsigned int, std::__cxx11::basic_string<char, std::char_traits<char>,
std::allocator<char> > const&, std::__cxx11::basic_string<char, std::char_traits<char>, std::allocator<char> > const&) (line.cc:116)
==9235==    by 0x413558: dwarf::compilation_unit::get_line_table() const (dwarf.cc:304)
==9235==    by 0x402CB7: main (dump-lines.cc:41)
==9235==
==9235== HEAP SUMMARY:
==9235==     in use at exit: 81,960 bytes in 75 blocks
==9235==   total heap usage: 139 allocs, 64 frees, 90,129 bytes allocated
==9235==
==9235== LEAK SUMMARY:
==9235==    definitely lost: 0 bytes in 0 blocks
==9235==    indirectly lost: 0 bytes in 0 blocks
==9235==      possibly lost: 144 bytes in 1 blocks
==9235==    still reachable: 81,816 bytes in 74 blocks
==9235==                       of which reachable via heuristic:
==9235==                         stdstring          : 86 bytes in 1 blocks
==9235==         suppressed: 0 bytes in 0 blocks
==9235== Rerun with --leak-check=full to see details of leaked memory
==9235==
==9235== For counts of detected and suppressed errors, rerun with: -v
==9235== ERROR SUMMARY: 0 errors from 0 contexts (suppressed: 0 from 0)
Aborted (core dumped)
```

I use **AddressSanitizer** to build ffjpeg and running it with the following command:

```
/path-to-libelfin/examples/dump-lines dump_line_global_buffer_overflow
```

This is the ASAN information (absolute path information omitted):

```
/path-to-libelfin-address/examples/dump-lines dump_line_global_buffer_overflow
=================================================================
==9296==ERROR: AddressSanitizer: global-buffer-overflow on address 0x00000045f374 at pc 0x00000043db90 bp 0x7fff57889ea0 sp 0x7fff57889e90
READ of size 4 at 0x00000045f374 thread T0
    #0 0x43db8f in dwarf::line_table::line_table(std::shared_ptr<dwarf::section> const&, unsigned long, unsigned int, std::__cxx11::basic_string<char, std::char_traits<char>,
std::allocator<char> > const&, std::__cxx11::basic_string<char, std::char_traits<char>, std::allocator<char> > const&) /path-to-libelfin-address/dwarf/line.cc:107
    #1 0x40f67b in dwarf::compilation_unit::get_line_table() const /path-to-libelfin-address/dwarf/dwarf.cc:304
    #2 0x403356 in main /path-to-libelfin-address/examples/dump-lines.cc:41
    #3 0x7f0bb309682f in __libc_start_main (/lib/x86_64-linux-gnu/libc.so.6+0x2082f)
    #4 0x403888 in _start (/path-to-libelfin-address/examples/dump-lines+0x403888)

0x00000045f374 is located 0 bytes to the right of global variable 'opcode_lengths' defined in 'line.cc:15:18' (0x45f340) of size 52
SUMMARY: AddressSanitizer: global-buffer-overflow /path-to-libelfin-address/dwarf/line.cc:107 dwarf::line_table::line_table(std::shared_ptr<dwarf::section> const&, unsigned long,
unsigned int, std::__cxx11::basic_string<char, std::char_traits<char>, std::allocator<char> > const&, std::__cxx11::basic_string<char, std::char_traits<char>, std::allocator<char>
> const&)
Shadow bytes around the buggy address:
  0x000080083e10: f9 f9 f9 f9 00 00 00 00 03 f9 f9 f9 f9 f9 f9 f9
  0x000080083e20: 00 00 00 00 00 00 00 00 04 f9 f9 f9 f9 f9 f9 f9
  0x000080083e30: 00 00 00 00 00 04 f9 f9 f9 f9 f9 f9 00 02 f9 f9
  0x000080083e40: f9 f9 f9 f9 00 00 00 00 03 f9 f9 f9 f9 f9 f9 f9
  0x000080083e50: 07 f9 f9 f9 f9 f9 f9 00 00 00 00 00 00 00 00 00
=>0x000080083e60: 04 f9 f9 f9 f9 f9 f9 f9 00 00 00 00 00 00[04]f9
  0x000080083e70: f9 f9 f9 f9 00 00 00 00 00 00 00 00 00 00 00 00
  0x000080083e80: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
  0x000080083e90: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
  0x000080083ea0: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
  0x000080083eb0: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
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
  Heap right redzone:      fb
  Freed heap region:       fd
  Stack left redzone:      f1
  Stack mid redzone:       f2
  Stack right redzone:     f3
  Stack partial redzone:   f4
  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
==9296==ABORTING
```

An attacker can exploit this vulnerability by submitting a malicious elf file that exploits this bug which will result in a Denial of Service (DoS) even buffer overflow.

👍 1

fgeek commented on Aug 6, 2021

CVE-2020-24824 has been assigned for this issue.

Assignees

No one assigned

Labels

None yet

Projects

None yet

Milestone

No milestone

Development

No branches or pull requests

2 participants