aclements / libelfin Public

<> Code ⊙ Issues 31 Pull requests 9 Actions Projects Wiki ···

New issue

Jump to bottom

SEGV in function dwarf::to_string at dwarf/value.cc:300 #51

⊙ 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-tree.

The testcase is dump_tree_segv2.

I use the following command:

```
/path-to-libelfin/examples/dump-tree dump_tree_segv2
```

and get:

```
Segmentation fault (core dumped)
```

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

```
valgrind /path-to-libelfin/examples/dump-tree dump_tree_segv2
==22094== Memcheck, a memory error detector
==22094== Copyright (C) 2002-2015, and GNU GPL'd, by Julian Seward et al.
==22094== Using Valgrind-3.11.0 and LibVEX; rerun with -h for copyright info
==22094== Command: /path-to-libelfin/examples/dump-tree dump_tree_segv2
==22094==
==22094== Invalid read of size 1
==22094==    at 0x44CE58: dwarf::to_string[abi:cxx11](dwarf::value const&) (value.cc:300)
==22094==    by 0x4031B0: dump_tree (dump-tree.cc:19)
==22094==    by 0x4031B0: main (dump-tree.cc:43)
==22094==  Address 0x402a000 is not stack'd, malloc'd or (recently) free'd
==22094==
==22094==
==22094== Process terminating with default action of signal 11 (SIGSEGV)
==22094==  Access not within mapped region at address 0x402A000
==22094==    at 0x44CE58: dwarf::to_string[abi:cxx11](dwarf::value const&) (value.cc:300)
==22094==    by 0x4031B0: dump_tree (dump-tree.cc:19)
==22094==    by 0x4031B0: main (dump-tree.cc:43)
==22094==  If you believe this happened as a result of a stack
==22094==  overflow in your program's main thread (unlikely but
==22094==  possible), you can try to increase the size of the
==22094==  main thread stack using the --main-stacksize= flag.
==22094==  The main thread stack size used in this run was 8388608.
--- <0>
<b> DW_TAG_compile_unit
      DW_AT_producer
      DW_AT_language 12 byte block: cb 0 0 0 12 0 0 0 26 5 40 0
      DW_AT_name long unsigned int
==22094==
==22094== HEAP SUMMARY:
==22094==     in use at exit: 111,921 bytes in 68 blocks
==22094==   total heap usage: 145 allocs, 77 frees, 150,879 bytes allocated
==22094==
==22094== LEAK SUMMARY:
==22094==    definitely lost: 0 bytes in 0 blocks
==22094==    indirectly lost: 0 bytes in 0 blocks
==22094==      possibly lost: 0 bytes in 0 blocks
==22094==    still reachable: 111,921 bytes in 68 blocks
==22094==         suppressed: 0 bytes in 0 blocks
==22094== Rerun with --leak-check=full to see details of leaked memory
==22094==
==22094== For counts of detected and suppressed errors, rerun with: -v
==22094== ERROR SUMMARY: 1 errors from 1 contexts (suppressed: 0 from 0)
Segmentation fault (core dumped)
```

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

```
/path-to-libelfin/examples/dump-tree dump_tree_segv2
Segmentation fault (core dumped)
```

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

```
/path-to-libelfin-address/examples/dump-tree dump_tree_segv2
=================================================================
==22134==ERROR: AddressSanitizer: unknown-crash on address 0x7f6f8b233000 at pc 0x000000428213 bp 0x7ffd7ae677d0 sp 0x7ffd7ae677c0
READ of size 1 at 0x7f6f8b233000 thread T0
    #0 0x428212 in dwarf::to_string[abi:cxx11](dwarf::value const&) /path-to-libelfin-address/dwarf/value.cc:300
    #1 0x403aec in dump_tree(dwarf::die const&, int) /path-to-libelfin-address/examples/dump-tree.cc:19
    #2 0x403361 in main /path-to-libelfin-address/examples/dump-tree.cc:43
    #3 0x7f6f8971282f in __libc_start_main (/lib/x86_64-linux-gnu/libc.so.6+0x2082f)
    #4 0x403878 in _start (/path-to-libelfin-address/examples/dump-tree+0x403878)

AddressSanitizer can not describe address in more detail (wild memory access suspected).
SUMMARY: AddressSanitizer: unknown-crash /path-to-libelfin-address/dwarf/value.cc:300 dwarf::to_string[abi:cxx11](dwarf::value const&)
Shadow bytes around the buggy address:
  0x0fee7163e5b0: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
  0x0fee7163e5c0: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
  0x0fee7163e5d0: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
  0x0fee7163e5e0: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
  0x0fee7163e5f0: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
=>0x0fee7163e600:[fe]fe fe fe fe fe fe fe fe fe fe fe fe fe fe fe
  0x0fee7163e610: fe fe fe fe fe fe fe fe fe fe fe fe fe fe fe fe
  0x0fee7163e620: fe fe fe fe fe fe fe fe fe fe fe fe fe fe fe fe
  0x0fee7163e630: fe fe fe fe fe fe fe fe fe fe fe fe fe fe fe fe
  0x0fee7163e640: fe fe fe fe fe fe fe fe fe fe fe fe fe fe fe fe
  0x0fee7163e650: fe fe fe fe fe fe fe fe fe fe fe fe fe fe fe fe
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
==22134==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).

fgeek commented on Aug 6, 2021

CVE-2020-24823 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