I have been using AFL++ with libdislocator to test the software at my company. As described in the libdislocator README, this library interposes `malloc` and `free` to make all allocations occur at the end of a page boundary, with the subsequent page `mprotect`ed to `PROT_NONE`. This causes any read or write beyond the end of the `malloc`ed buffer to immediately segfault.

During testing, AFL reported a crash in `__strncmp_avx2` as follows:

```
$ LD_PRELOAD=/home/dmendenhall/AFLplusplus/libdislocator.so strace <prog>
...
mmap(NULL, 8192, PROT_READ|PROT_WRITE, MAP_PRIVATE|MAP_ANONYMOUS, -1, 0) =
0x7f89e3789000
mprotect(0x7f89e378a000, 4096, PROT_NONE) = 0
--- SIGSEGV {si_signo=SIGSEGV, si_code=SEGV_ACCERR, si_addr=0x7f89e378a000} ---
+++ killed by SIGSEGV (core dumped) +++
Segmentation fault
$ gdb <prog> core
...
Program terminated with signal SIGSEGV, Segmentation fault.
#0  0x00007f89e289c7b3 in __strncmp_avx2 () from /lib64/libc.so.6
...
(gdb) disassemble
...
   0x00007f89e289c7a5 <+421>:   vpminub %ymm1,%ymm4,%ymm0
   0x00007f89e289c7a9 <+425>:   vmovdqa 0x60(%rax),%ymm3
   0x00007f89e289c7ae <+430>:   vpcmpeqb 0x40(%rdx),%ymm2,%ymm5
=> 0x00007f89e289c7b3 <+435>:   vpcmpeqb 0x60(%rdx),%ymm3,%ymm6
   0x00007f89e289c7b8 <+440>:   vpminub %ymm2,%ymm5,%ymm5
   0x00007f89e289c7bc <+444>:   vpminub %ymm3,%ymm6,%ymm6
   0x00007f89e289c7c0 <+448>:   vpminub %ymm5,%ymm0,%ymm0
...
(gdb) info registers
rax            0x7f89e3797f80      140230203637632
rbx            0x0                 0/
rcx            0xfffffffc1         4294967233
rdx            0x7f89e3789f81      140230203580289
rsi            0x1f                31
rdi            0x0                 0
rbp            0x7ffe3fa19540      0x7ffe3fa19540
rsp            0x7ffe3fa19518      0x7ffe3fa19518
r8             0x0                 0
r9             0x0                 0
r10            0xffffffffffffffff  -1
r11            0x7f                127
r12            0x405370            4215664
r13            0x7ffe3fa1a690      140729965979280
r14            0x0                 0
r15            0x0                 0
rip            0x7f89e289c7b3      0x7f89e289c7b3 <__strncmp_avx2+435>
eflags         0x10246             [ PF ZF IF RF ]
```

```
cs             0x33               51
ss             0x2b               43
ds             0x0                0
es             0x0                0
fs             0x0                0
gs             0x0                0
```

The crashing instruction is attempting to read 32 bytes starting at `0x7f89e3789fe1` (%rdx+0x60). This will be bytes `0x7f89e3789fe1` through `0x7f89e378a000`, inclusive. As we can see from the `strace` output, the page starting at address `0x7f89e378a000` has been marked `PROT_NONE`. That means the code is reading one byte into unreadable memory, causing the segfault.

Of course, normally in this situation the problem is the calling code, not glibc, so I traced through the crash in gdb.

```
$ gdb <prog>
...
(gdb) set exec-wrapper env
'LD_PRELOAD=/home/dmendenhall/AFLplusplus/libdislocator.so'
(gdb) break __strncmp_avx2
Breakpoint 1 at 0x7ffff7037600
(gdb) run

Breakpoint 1, 0x00007ffff7037600 in __strncmp_avx2 () from /lib64/libc.so.6
(gdb) info registers
rax            0x7ffff7f29fe8      140737353261032
rbx            0x0                 0
rcx            0xd1aa              53674
rdx            0x80                128
rsi            0x7ffff7f1ff80      140737353219968
rdi            0x7ffff7f2df7f      140737353277311
rbp            0x7fffffffc6e0      0x7fffffffc6e0
rsp            0x7fffffffc6b8      0x7fffffffc6b8
r8             0x80                128
r9             0x0                 0
r10            0x0                 0
r11            0x7ffff7037600      140737337587200
r12            0x405370            4215664
r13            0x7fffffffd830      140737488345136
r14            0x0                 0
r15            0x0                 0
rip            0x7ffff7037600      0x7ffff7037600 <__strncmp_avx2>
eflags         0x206               [ PF IF ]
cs             0x33                51
ss             0x2b                43
ds             0x0                 0
es             0x0                 0
fs             0x0                 0
gs             0x0                 0
```

As we can see from the registers %rdi, %rsi, and %rdx, the arguments to `strncmp` were `0x7ffff7f2df7f`, `0x7ffff7f1ff80`, and 128. These pointers are, respectively, 129 and 128 bytes away from their page boundaries, so a `strncmp` of length 128 should be fine. It is not shown here, but the 129 byte string is NUL terminated while the 128 byte string is not. The strings are otherwise identical.

As I traced through the `__strncmp_avx2` assembly, program flow was as follows:

- `ENTRY (STRCMP)`
- `L(cross_page)`
- `L(cross_page_1_vector)`
- `L(cross_page_1_xmm)`
- `L(cross_page_8bytes)`
- `L(cross_page_4bytes)`
- `L(cross_page_loop)`
    - Repeated 129 times
- `L(main_loop_header)`
- `...`

In other words, the algorithm correctly determined that the strings were at page boundaries and fell back to the one-byte-at-a-time loop. However, the last iteration of `cross_page_loop` did not proceed to `L(zero)` as it should have but instead went to `main_loop_header`. The reason is that 128, in addition to being the length passed to `strncmp`, is also `(VEC_SIZE * 4)`. Here is the relevant code from `cross_page_loop`:

```
        addl    $SIZE_OF_CHAR, %edx
        cmpl    $(VEC_SIZE * 4), %edx
        je      L(main_loop_header)
# ifdef USE_AS_STRNCMP
        cmpq    %r11, %rdx
        jae     L(zero)
# endif
```

`$SIZE_OF_CHAR` is 1, so on the 129th iteration this is added to %edx of 0x79 to yield %edx of 0x80. This is then compared to `$(VEC_SIZE * 4)`. The values match, so the code jumps to `L(main_loop_header)`. Waiting immediately after this code is a comparison to %r11, which is also 0x80. Had this test executed first, `cross_page_loop` would have correctly determined that the first 128 bytes of the two strings were identical and exited the function. Instead, we proceed to the main loop which eventually reads past the end of the string.

To correct this bug, the order of the two tests should be inverted as follows:

```
--- glibc-2.28.orig/sysdeps/x86_64/multiarch/strcmp-avx2.S      2020-05-06
15:00:55.475787638 -0700
+++ glibc-2.28/sysdeps/x86_64/multiarch/strcmp-avx2.S   2020-05-06
16:26:40.209796315 -0700
@@ -637,12 +637,12 @@
 # endif
        jne     L(different)
        addl    $SIZE_OF_CHAR, %edx
-       cmpl    $(VEC_SIZE * 4), %edx
```

```
-       je      L(main_loop_header)
 # ifdef USE_AS_STRNCMP
        cmpq    %r11, %rdx
        jae     L(zero)
 # endif
+       cmpl    $(VEC_SIZE * 4), %edx
+       je      L(main_loop_header)
 # ifdef USE_AS_WCSCMP
        movl    (%rdi, %rdx), %eax
        movl    (%rsi, %rdx), %ecx
```

System information

```
$ uname -a
Linux spenseweed 4.18.0-147.8.1.el8_1.x86_64 #1 SMP Thu Apr 9 13:49:54 UTC 2020
x86_64 x86_64 x86_64 GNU/Linux
$ /lib64/libc.so.6 --version
GNU C Library (GNU libc) stable release version 2.28.
Copyright (C) 2018 Free Software Foundation, Inc.
This is free software; see the source for copying conditions.
There is NO warranty; not even for MERCHANTABILITY or FITNESS FOR A
PARTICULAR PURPOSE.
Compiled by GNU CC version 8.3.1 20190507 (Red Hat 8.3.1-4).
libc ABIs: UNIQUE IFUNC ABSOLUTE
For bug reporting instructions, please see:
<http://www.gnu.org/software/libc/bugs.html>.
```