[contents] [usage] [execution] [stack] [breakpoints] [watchpoints] [advanced]

7.2 Example Debugging Session: Segmentation Fault Example

We are going to use gdb to figure out why the following program causes a segmentation fault. The program is meant to read in a line of text from the user and print it. However, we will see that in it's current state it doesn't work as expected...

```
1 : #include <stdio.h>
2 : #include <stdlib.h>

3 : int main(int argc, char **argv)
4 : {
5 :   char *buf;
6 :
7 :   buf = malloc(1<<31);
8 :
9 :   fgets(buf, 1024, stdin);
10:   printf("%s\n", buf);
11:
12:   return 1;
13: }
```

The first step is to compile the program with debugging flags:

```
prompt> gcc -g segfault.c
```

Now we run the program:

```
prompt > a.out
Hello World!
Segmentation fault
prompt >
```

This is not what we want. Time to fire up gdb:

```
prompt > gdb a.out
GNU gdb 5.0
Copyright 2000 Free Software Foundation, Inc.
GDB is free software, covered by the GNU General Public License, and you are
welcome to change it and/or distribute copies of it under certain conditions.
Type "show copying" to see the conditions.
There is absolutely no warranty for GDB.  Type "show warranty" for details.
This GDB was configured as "i686-pc-linux-gnu"...
(gdb)
```

We'll just run it and see what happens:

```
(gdb) run
Starting program: /home/dgawd/cpsc/363/a.out
test string

Program received signal SIGSEGV, Segmentation fault.
0x4007fc13 in _IO_getline_info () from /lib/libc.so.6
```

So we received the SIGSEGV signal from the operating system. This means that we tried to access an invalid memory address. Let's take a backtrace:

```
(gdb) backtrace
#0  0x4007fc13 in _IO_getline_info () from /lib/libc.so.6
#1  0x4007fb6c in _IO_getline () from /lib/libc.so.6
#2  0x4007ef51 in fgets () from /lib/libc.so.6
#3  0x80484b2 in main (argc=1, argv=0xbffffaf4) at segfault.c:10
#4  0x40037f5c in __libc_start_main () from /lib/libc.so.6
```

We are only interested in our own code here, so we want to switch to stack frame 3 and see where the program crashed:

```
(gdb) frame 3
#3  0x80484b2 in main (argc=1, argv=0xbffffaf4) at segfault.c:10
10        fgets(buf, 1024, stdin)
```

We crashed inside the call to fgets. In general, we can assume that library functions such as fgets work properly (if this isn't the case, we are in a lot of trouble). So the problem must be one of our arguments. You may not know that 'stdin' is a global variable that is created by the stdio libraries. So we can assume this one is ok. That leaves us with 'buf':

```
(gdb) print buf
$1 = 0x0
```

The value of buf is 0x0, which is the NULL pointer. This is not what we want - buf should point to the memory we allocated on line 8. So we're going to have to find out what happened there. First we want to kill the currently-running invocation of our program:

```
(gdb) kill
Kill the program being debugged? (y or n) y
```

Now set a breakpoint on line 8:

```
(gdb) break segfault.c:8
Breakpoint 1 at 0x8048486: file segfault.c, line 8.
```

Now run the program again:

```
(gdb) run
Starting program: /home/dgawd/cpsc/363/a.out

Breakpoint 1, main (argc=1, argv=0xbffffaf4) at segfault.c:8
8         buf = malloc(1<<31);
```

We're going to check the value of buf before the malloc call. Since buf wasn't initialized, the value should be garbage, and it is:

```
(gdb) print buf
$2 = 0xbfffaa8 "Èúÿ¿#\177\003@t`\001@\001"
```

Now step over the malloc call and examine buf again:

```
(gdb) next
10        fgets(buf, 1024, stdin);
(gdb) print buf
$3 = 0x0
```

After the call to malloc, buf is NULL. If you were to go check the man page for malloc, you would discover that malloc returns NULL when it cannot allocate the amount of memory requested. So our malloc must have failed. Let's go back and look at it again:

```
7 :   buf = malloc(1<<31);
```

Well, the value of the expression 1 << 31 (the integer 1 right-shifted 31 times) is 429497295, or 4GB (gigabytes). Very few machines have this kind of memory - mine only

has 256MB. So of cousre malloc would fail. Furthermore, we are only reading in 1024 bytes in the fgets call. All that extra space would be wasted, even if we could allocate it. Change the 1<<31 to 1024 (or 1<<9), and the program will work as expected:

```
prompt >
Hello World!
Hello World!

prompt >
```

So now you know how to debug segmentation faults with gdb. This is extremely useful (I use it more often then I care to admit). The example also illustrated another very important point: ALWAYS CHECK THE RETURN VALUE OF MALLOC! Have a nice day.

[contents] [usage] [execution] [stack] [breakpoints] [watchpoints] [advanced]

Questions? Comments? Flames? email rms@unknownroad.com