memset() causing data abort

Asked 16 years, 4 months ago Modified 7 years, 4 months ago Viewed 5k times

4

I'm getting some strange, intermittent, data aborts (< 5% of the time) in some of my code, when calling `memset()`. The problem is that is usually doesn't happen unless the code is running for a couple days, so it's hard to catch it in the act.

I'm using the following code:

```
char *msg = (char*)malloc(sizeof(char)*2048);
char *temp = (char*)malloc(sizeof(char)*1024);
memset(msg, 0, 2048);
memset(temp, 0, 1024);
char *tempstr = (char*)malloc(sizeof(char)*128);

sprintf(temp, "%s %s/%s %s%s", EZMPPOST, EZMPTAG, EZMPVER, TYPETXT, EOL);
strcat(msg, temp);

//Add Data
memset(tempstr, '\0', 128);
wcstombs(tempstr, gdevID, wcslen(gdevID));
sprintf(temp, "%s: %s%s", "DeviceID", tempstr, EOL);
strcat(msg, temp);
```

As you can see, I'm not trying to use memset with a size larger that what's originally allocated with `malloc()`

Anyone see what might be wrong with this?

c++ c memory windows-mobile

Share
Improve this question
Follow

edited Aug 3, 2017 at 16:07

Error - Syntactical Remorse
7,869 ● 4 ● 28 ● 54

asked Aug 22, 2008 at 14:17

Adam Haile
31.3k ● 60 ● 195 ● 290

10 Answers

Sorted by: Highest score (default)

21

`malloc` can return `NULL` if no memory is available. You're not checking for that.

Share Improve this answer Follow

answered Aug 22, 2008 at 14:21

Joel Spolsky
33.6k ● 17 ● 90 ● 105

2 Note also that because of optimistic allocation, it is not because maloc() returns non-null that you do have enough memory. It will fail at the first actual access of the memory which will be here ... memset(). – Ben Jul 24, 2009 at 7:44

@Ben: You are very correct for OS's like desktop Linux. But I doubt that mobile OS like Windows Mobile does optimistic alloc. It does not make a lot of sense on a system with limited RAM and no virtual RAM. – Zan Lynx Sep 21, 2010 at 23:37

4

There's a couple of things. You're using `sprintf` which is inherently unsafe; unless you're 100% positive that you're not going to exceed the size of the buffer, you should almost *always* prefer `snprintf`. The same applies to `strcat`; prefer the safer alternative `strncat`.

Obviously this may not fix anything, but it goes a *long* way in helping spot what might otherwise be very annoying to spot bugs.

Share Improve this answer Follow

answered Aug 22, 2008 at 14:26

FreeMemory
8,604 ● 7 ● 38 ● 50

No don't use strncat. It fills the entire buffer. This kills performance if you're using a large buffer. It hit the top of my profiler graph on one project that used 8K buffer for URLs. – Zan Lynx Jul 30, 2009 at 3:40

@Zan Lynx: what's the alternative for `strncat`? – Cristian Ciupitu Sep 21, 2010 at 22:23

1 @Cristian: strlcat is one option. You may have to copy the code into your project though, it isn't always in the dev environment. Tracking your string length (or using pointers to buffer start, current position and buffer end) in source and destination and using memcpy is my favorite method. – Zan Lynx Sep 21, 2010 at 23:35

@Zan Lynx: `strlcat` might be nicer than `strncat` from a semantic point of view, but I still don't understand how its performance could be better. The paper presenting strlcpy and strlcat mentions only `strncpy`: "Finally, strncpy() zero-fills the remainder of the destination string, incurring a performance penalty". By the way my Linux `strncat` man page says nothing about a performance penalty. – Cristian Ciupitu Sep 22, 2010 at 1:36

@Cristian: D'oh. Just now that you write it I see that I meant strncpy all along. That's the one you should avoid, and my previous comment is now silly. – Zan Lynx Sep 22, 2010 at 1:40

3

> malloc can return NULL if no memory is available. You're not checking for that.

Right you are... I didn't think about that as I was monitoring the memory and it there was enough free. Is there any way for there to be available memory on the system but for malloc to fail?

> Yes, if memory is fragmented. Also, when you say "monitoring memory," there may be something on the system which occasionally consumes a lot of memory and then releases it before you notice. If your call to `malloc` occurs then, there won't be any memory available. -- **Joel**

Either way...I will add that check :)

Share
Improve this answer
Follow

edited Aug 22, 2008 at 14:42

Joel Spolsky
33.6k ● 17 ● 90 ● 105

answered Aug 22, 2008 at 14:26

Adam Haile
31.3k ● 60 ● 195 ● 290

1

`wcstombs` doesn't get the size of the destination, so it can, in theory, buffer overflow.

And why are you using `sprintf` with what I assume are constants? Just use:

```
EZMPPOST" " EZMPTAG "/" EZMPVER " " TYPETXT EOL
```

C and C++ combines string literal declarations into a single string.

Share
Improve this answer
Follow

edited Jul 3, 2012 at 15:02

user142162

answered Aug 22, 2008 at 16:31

Mat Noguchi
1,080 ● 6 ● 7

0

Have you tried using Valgrind? That is usually the fastest and easiest way to debug these sorts of errors. If you are reading or writing outside the bounds of allocated memory, it will flag it for you.

Share Improve this answer Follow

answered Aug 22, 2008 at 14:18

Doug
1,083 ● 1 ● 10 ● 16

0

> You're using sprintf which is inherently unsafe; unless you're 100% positive that you're not going to exceed the size of the buffer, you should almost always prefer snprintf. The same applies to strcat; prefer the safer alternative strncat.

Yeah..... I mostly do .NET lately and old habits die hard. I likely pulled that code out of something else that was written before my time...

But I'll try not to use those in the future ;)

Share Improve this answer Follow

answered Aug 22, 2008 at 14:28

Adam Haile

31.3k ● 60 ● 195 ● 290

0

You know it might not even be your code... Are there any other programs running that could have a memory leak?

Share Improve this answer Follow

answered Aug 22, 2008 at 14:28

TK.

47.8k ● 47 ● 121 ● 148

0

It could be your processor. Some CPUs can't address single bytes, and require you to work in words or chunk sizes, or have instructions that can only be used on word or chunk aligned data.

Usually the compiler is made aware of these and works around them, but sometimes you can malloc a region as bytes, and then try to address it as a structure or wider-than-a-byte field, and the compiler won't catch it, but the processor will throw a data exception later.

It wouldn't happen unless you're using an unusual CPU. ARM9 will do that, for example, but i686 won't. I see it's tagged windows mobile, so maybe you do have this CPU issue.

Share Improve this answer Follow

answered Sep 20, 2008 at 6:40

davenpcj

12.7k ● 5 ● 42 ● 38

Instead of doing `malloc` followed by `memset`, you should be using `calloc` which will clear the newly allocated memory for you. Other than that, do what Joel said.

Share Improve this answer Follow

answered Sep 20, 2008 at 6:46

1800 INFORMATION

135k • 30 • 163 • 242

0

NB borrowed some comments from other answers and integrated into a whole. The code is all mine...

- Check your error codes. E.g. malloc can return NULL if no memory is available. This could be causing your data abort.
- sizeof(char) is 1 by definition
- Use snprintf not sprintf to avoid buffer overruns
  - If EZMPPOST etc are constants, then you don't need a format string, you can just combined several string literals as STRING1 " " STRING2 " " STRING3 and strcat the whole lot.
- You are using much more memory than you need to.
- With one minor change, you don't need to call memset in the first place. Nothing really requires zero initialisation here.

This code does the same thing, safely, runs faster, and uses less memory.

```
    // sizeof(char) is 1 by definition. This memory does not require zero
    // initialisation. If it did, I'd use calloc.
    const int max_msg = 2048;
    char *msg     = (char*)malloc(max_msg);
    if(!msg)
    {
       // Allocaton failure
       return;
    }
    // Use snprintf instead of sprintf to avoid buffer overruns
    // we write directly to msg, instead of using a temporary buffer and then
calling
    // strcat. This saves CPU time, saves the temporary buffer, and removes the
need
    // to zero initialise msg.
    snprintf(msg, max_msg, "%s %s/%s %s%s", EZMPPOST, EZMPTAG, EZMPVER,
TYPETXT, EOL);

   //Add Data
   size_t len = wcslen(gdevID);
   // No need to zero init this
```

0

```
   char* temp = (char*)malloc(len);
   if(!temp)
   {
      free(msg);
      return;
   }
   wcstombs(temp, gdevID, len);
   // No need to use a temporary buffer - just append directly to the msg,
protecting
   // against buffer overruns.
   snprintf(msg + strlen(msg),
           max_msg - strlen(msg), "%s: %s%s", "DeviceID", temp, EOL);
   free(temp);
```

Share

Improve this answer

Follow

edited Jul 24, 2009 at 7:33

answered Sep 24, 2008 at 17:55

Airsource Ltd

32.6k ● 13 ● 74 ● 76