HW2.10. Reading Memory

Consider the following code: `memory.c`.

You may find this reference useful on the difference between structs and unions.

Q1.1: During the function call `q1()`, what two lines get printed? You may find an ASCII table (like this one) useful.

Line 1:	ASCII is fun	? ✓ 100%
Line 2:	3 1 4 1 5 9 3	? ✓ 100%

We run GDB on `memory.c` on a **64-bit little-endian system**, up to the line marked `Breakpoint 1`. We then run several GDB commands; the commands and their outputs are here: `memoryGDBoutput.txt`

Quick reference guide:

`print expr` evaluates expr and prints out the result.

`x/20xw expr` prints out 20 words of memory, starting at the given memory address, as if memory was an array of word-size values, in hexadecimal.

`$sp` is a pointer to the bottom of the stack.

Q2.1: Given the information from the GDB memory dump, what two lines get printed by `q2()`? Hint: Try and find where the two constants are in the stack. What information in the GDB dump could you have used to find those constants, if you didn't know their actual values?

Line 3:	is fun	? ✓ 100%
Line 4:	6 28 496 8128 33550336 28 28	? ✓ 100%

Try a new variant

Homework 2

Assessment overview

Total points:	100/100
Score:	100%

Question

Value:	15
History:	15
Awarded points:	15/15

Report an error in this question

Previous question

Next question

Attached files

No attached files

Attach a file

Attach text

Correct answer

Q1.1: During the function call `q1()`, what two lines get printed? You may find an ASCII table (like this one) useful.
Line 1: `ASCII is fun` Line 2: `3 1 4 1 5 9 3`

We run GDB on `memory.c` on a **64-bit little-endian system**, up to the line marked `Breakpoint 1`. We then run several GDB commands; the commands and their outputs are here: `memoryGDBoutput.txt`

Quick reference guide:

`print expr` evaluates expr and prints out the result.

`x/20xw expr` prints out 20 words of memory, starting at the given memory address, as if memory was an array of word-size values, in hexadecimal.

`$sp` is a pointer to the bottom of the stack.

Q2.1: Given the information from the GDB memory dump, what two lines get printed by `q2()`? Hint: Try and find where the two constants are in the stack. What information in the GDB dump could you have used to find those constants, if you didn't know their actual values?

Line 3: ` is fun` Line 4: `6 28 496 8128 33550336 28 28`

The code that was given presents a sequence of tree operations. Note that our `struct node` contains a left pointer and a union of a right pointer and an integer. Effectively, there are two types of tree nodes; internal nodes would have a non-NULL left pointer, and a non-NULL right pointer, while leaves would have a NULL left pointer and an integer of data. We use the left pointer as an indicator if the node is a leaf; if the left pointer is NULL, we assume that it's a leaf. As a result, we never need to store both a right pointer and data, so we define it as a union in order to save space; within the struct, only 8 bytes get allocated for both the pointer and the data (since on a 64 bit system, pointers are 8 bytes long).

The second part of the question deals with actually reading through memory. We can see through the infodump exactly how the computer sees its internal data; there's no real way to distinguish pointer data from integer data or union data. During compilation, all information related to how to interpret each variable is removed, and is instead replaced with "hardcoded" sequences of operations that happen to use the data the way we want. C is exceptionally lax in type checking; if you ask the compiler to treat a string as an integer, it will happily do that for you with at most a warning.

While reading through memory, you may have noticed several things about how memory is laid out. Here are some highlights which are useful to know:

Variables on the stack can be rearranged by the compiler, though generally, they end up in a contiguous block. The compiler has full control over this, and can choose to remove variables entirely sometimes if it notices that the variable isn't used or is always a constant.

Uninitialized variables contain garbage, but often that garbage is just the result of not clearing out old data. In this case, the uninitialized char buffer ended up overlapping with the old stack frame of q1, and included parts of the string initialized there. We can see that stack-allocated variables get overwritten this way, but said overwriting could take quite a while before it actually happens. This also means that printing out uninitialized data could leak information about previous function calls; the security implications of this are covered in CS 161.

There's a decent amount of metadata floating around. The stack contains various pointers and seemingly random data (which will be covered in greater detail during the RISC-V section of this class), while every malloced block stores its length in the byte immediately before the block (the details of which are covered in CS 162).

Submitted answer 3 **correct: 100%**
Submitted at 2022-09-03 10:26:06 (PDT)

hide

Q1.1: During the function call `q1()`, what two lines get printed? You may find an ASCII table (like this one) useful.
Line 1: `ASCII is fun` ✓100% Line 2: `3 1 4 1 5 9 3` ✓100%

We run GDB on `memory.c` on a **64-bit little-endian system**, up to the line marked `Breakpoint 1`. We then run several GDB commands; the commands and their outputs are here: `memoryGDBoutput.txt`

Quick reference guide:

`print expr` evaluates expr and prints out the result.

`x/20xw expr` prints out 20 words of memory, starting at the given memory address, as if memory was an array of word-size values, in hexadecimal.

`$sp` is a pointer to the bottom of the stack.

Q2.1: Given the information from the GDB memory dump, what two lines get printed by `q2()`? Hint: Try and find where the two constants are in the stack. What information in the GDB dump could you have used to find those constants, if you didn't know their actual values?

Line 3: ` is fun` ✓100% Line 4: `6 28 496 8128 33550336 28 28` ✓100%

Submitted answer 2 **partially correct: 75%**

Submitted at 2022-09-03 10:19:17 (PDT)

Submitted answer 1 saved, not graded

Submitted at 2022-09-03 09:52:36 (PDT)

show