21-hll-p3-notes

High-Level Languages: Wrap-up

Agenda

- 0. Re-Orienting
- 1. Getting Strange
- 2. Data Structures
- 3. Stacks and Data Structures
- 4. Buffer Overflow
- 5. What's Next

Reading:

- 3.7 through 3.9
- (note gdb in 3.11)
- 3.12

---

slides.pdf (https://ssl.cs.dartmouth.edu/~sws/cs51-s15/21-hll-p3/slides.pdf)

---

0. Re-Orienting

From thinking about software down to assembly/machine language for an ISA.

From electricity up to a microarchitecture implementing that ISA.

The correspondence between C and the ISA

- using an industrial-strength compiler for an industrial-strength processor family!

How the higher-level operations (e.g., math, logic, control flow) map onto assembly instructions

How a lot of it all comes together when we think about stack frames and function calls. Typically:

- frame pointer at bottom (highest) address
- stack pointer at top (lowest) address

- locals (sometimes) at the bottom, accessed via negative offset from frame pointer
- arguments to a subroutine at top, accessed via nonnegative offset from stack pointer
- Since stacks grow to smaller addresses.... we access arguments from our caller by using a positive offset from the frame pointer, to reach down into the arg-building space of the caller's stack frame.

Implications of the hardware-software interface

---

1. Getting Strange

Do we have to use %ebp as a frame pointer?

Can locals live in registers?

---

2. Data Structures

Hardware view

how does the structure get laid out in memory?

```
        typedef struct {
          unsigned short short1;
          int int1;
          unsigned char char1;
          int int2;

        } foo_t;

        foo_t foo;

        main() {



          printf("foo        lives at %p\n", &foo);
          printf("foo.short1 lives at %p\n", &(foo.short1));
```

```
      printf("foo.int1   lives at %p\n", &(foo.int1));
      printf("foo.char1  lives at %p\n", &(foo.char1));
      printf("foo.int2   lives at %p\n", &(foo.int2));
      printf("sizeof(foo) = %d\n", sizeof(foo));

    }
```

[pack.c (https://ssl.cs.dartmouth.edu/~sws/cs51-s15/21-hll-p3/demo/pack.c)](https://ssl.cs.dartmouth.edu/~sws/cs51-s15/21-hll-p3/demo/pack.c)

- gcc
- gcc -fpack-struct

---

3. Stacks and Data Structures

Can we pass **data structures** as arguments? As return values?

Should we?

[struct_argB.c (https://ssl.cs.dartmouth.edu/~sws/cs51-s15/22-hll-p3/demo/struct_argB.c)](https://ssl.cs.dartmouth.edu/~sws/cs51-s15/22-hll-p3/demo/struct_argB.c)

Walk through (and draw on board):

- where foo lives, compared to main's stack frame
- what happens in the fun1 call, and why
- what happens in the fun2 call, and why. (where is fval?)

[struct_val1B.c (https://ssl.cs.dartmouth.edu/~sws/cs51-s15/22-hll-p3/demo/struct_val1B.c)](https://ssl.cs.dartmouth.edu/~sws/cs51-s15/22-hll-p3/demo/struct_val1B.c)

Walk through (and draw on board):

- where foo lives, compared to main's stack frame
- what happens in the fun3 call, and why. (where is fval?)
- what happens to foop after the fun3 call, and why
- what happens the second time we look?

[struct_val2B.c (https://ssl.cs.dartmouth.edu/~sws/cs51-s15/22-hll-p3/demo/struct_val2B.c)](https://ssl.cs.dartmouth.edu/~sws/cs51-s15/22-hll-p3/demo/struct_val2B.c)

Walk through (and draw on board):

- where foo lives, compared to main's stack frame

- what happens in the fun4 call, and why. (where is fval?)
- does foo see the change?
- can the return value come back in %eax?

In **struct_val2c.c (https://ssl.cs.dartmouth.edu/~sws/cs51-s15/22-hll-p3/demo/struct_val2C.c)** , we can see in the caller

- the data structure sent as a parameter being stashed on the stack frame
- space for the returned data structure being reserved on the stack frame
- the address of that space being stashed with the parameter

And the callee using a 64-bit move instruction and a 128-bit xmm register to copy the returned data structure to the reserved space

---

4. Buffer Overflow

A long-running class of security holes that follow from the interactions between C and the architecture!

Including the Heartbleed bug.

---

bo.c (https://ssl.cs.dartmouth.edu/~sws/cs51-s15/21-hll-p3/demo/bo.c) shows a working example.

(Draw stack frame, work through three examples)

Next step: what if the evil string **included** the code?

There's a long line of countermeasures...and ways to work around countermeasures.

---

5. What's Next

The mess vs speed graph

Big move number 1: pipelining

Big move number 2: caching