Buflab

Recitation - 09/20/2010

By sseshadr

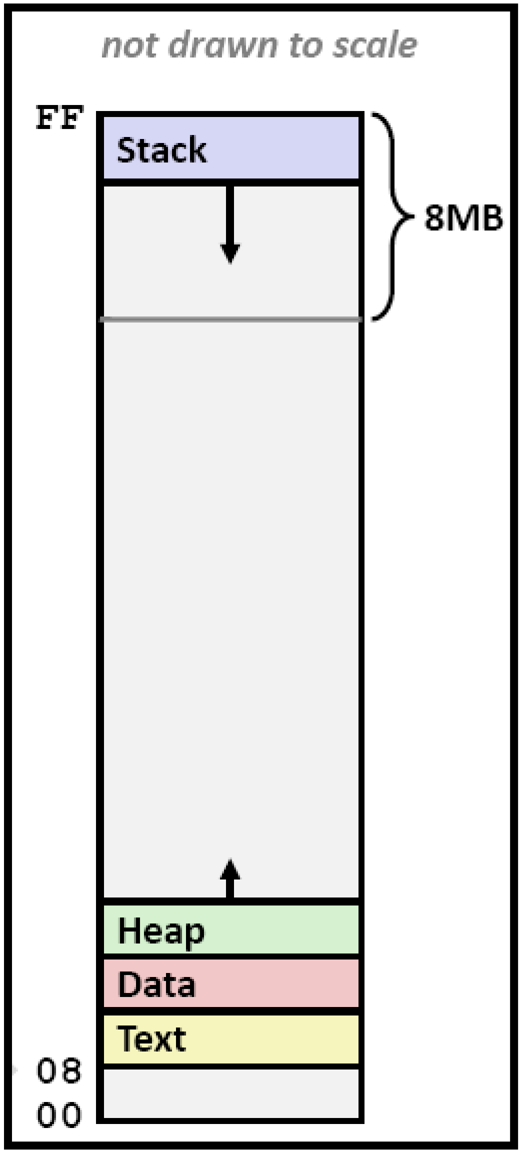
Agenda

- Reminders
  - Bomblab should be finished up
  - Exam 1 is on Tuesday 09/28/2010
- Stack Discipline
- Buflab
  - One of you will get lucky
- Datalab Handouts
  - Overall style OK. See comments on ink

Stack?

- What is the stack?
  - It's NOT
    - The memory regions returned by malloc()
    - The memory where your program itself is loaded
    - Where the bits for general purpose register are stored
    - Where the return value of a function is stored
  - It IS
    - Where you can often find a function's local variables
    - How parameters are passed in 32-bit x86
    - Where the return ADDRESS is stored
    - A highly structured (easily corruptible) data structure essential to the execution of your code!

Virtual Address Space



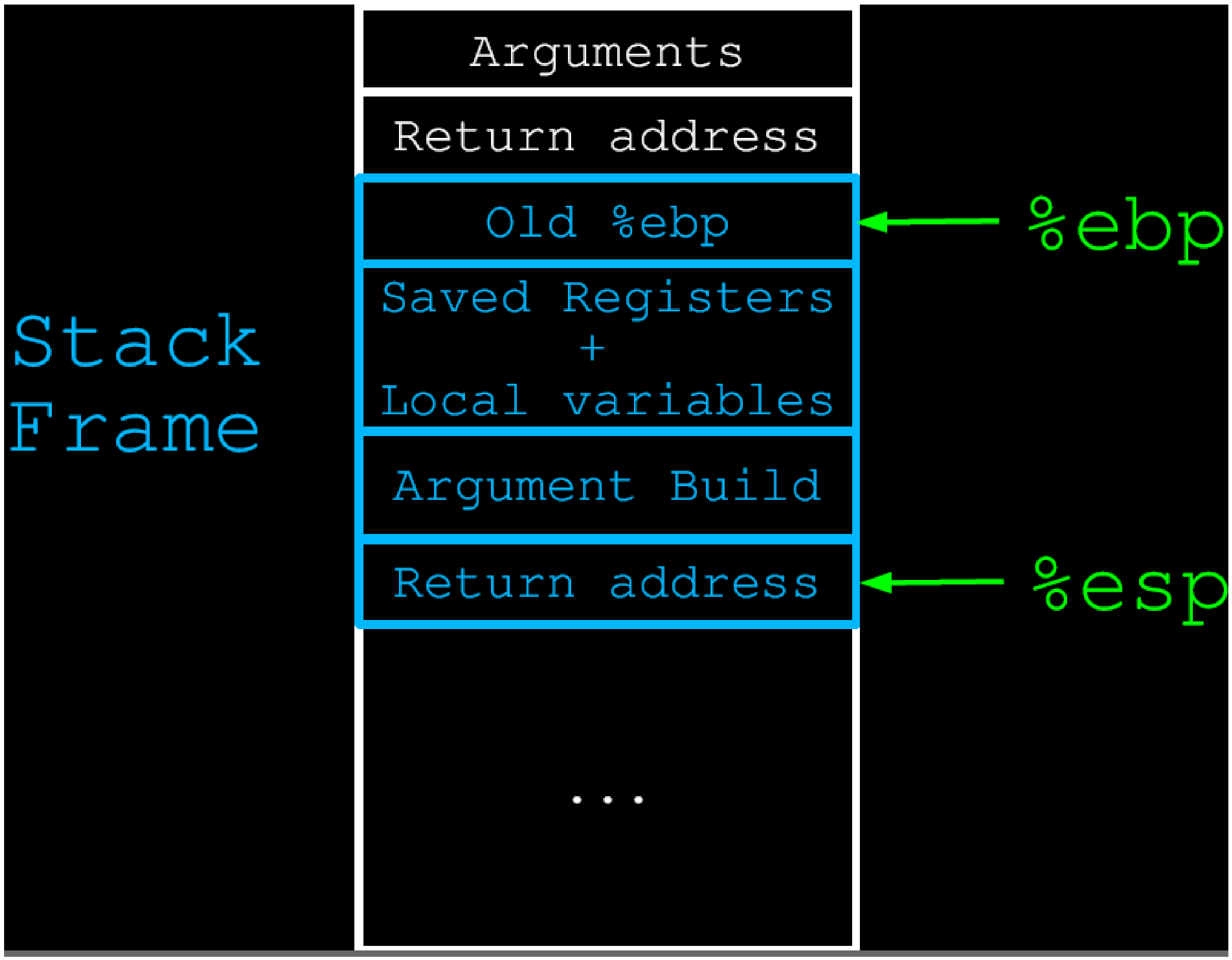
The stack starts very close to `0xFFFFFFFF` (for 32-bit) and grows DOWN

Stack Discipline

- Each function has a stack frame
  - Local variables
  - Saved registers
  - Anything that function wants to put in its stack

- Functions CALL other functions
  - Arguments
  - Return address
  - Base pointer

Stack Discipline



Stack Discipline

- What happens when `ret` is executed?
  - Pop the stack and "`jmp`" to that address
  - In Java, think: `eip = stack.pop();`
- Where can I find the return address of my function?
  - `*(ebp + 4)`
- How can I find the second parameter to my function?
  - `*(ebp + 12) == the second parameter`
- How can I find the return address of the function that CALLED me?
  - Remember that `*ebp == old ebp`
  - `*(*ebp + 4)`
- How are arguments pushed onto the stack?
  - Reverse order! Stacks are LIFO

Stack Discipline

- How will the "transition stack" look like for a function `foo` which makes the function call `printf("str = %s, num = %d", name, 16)`

Values	Description
…	Stack frame for `foo`
`0x00000010`	Arguments pushed in REVERSE order
<address of `name`>	
<address of format string in .rodata>	All hard-coded strings are put into .rodata before runtime
return address (where EIP should return in `foo` after the printf call)	Return address
`foo`'s ebp	Old base pointer
…	Stack frame for printf

Buflab

- It's a hack.
  - Overflow the buffer to write over the return address
- We will go over how to solve the first phase.
- Whoever can answer this next question will get a head start
  - Only answer if you haven't yet started buflab

x86 Review Question:

- `%eax` at the start of this has the value `0x01000000`
- What will `%eax` have after executing this code?

```
mov 4(%eax), %eax   →   eax = *(eax+4) = 0x10203040
lea 4(%eax), %eax   →   eax =  (eax+4) = 0x10203044
```

Memory	Value
0x00FFFFFC	0xDEADBEEF
0x01000000	0x01020304
0x01000004	0x10203040
0x10203040	0x12345678
0x10203044	0xBEEFBABE

Buflab Demo

Pick up your datalabs!