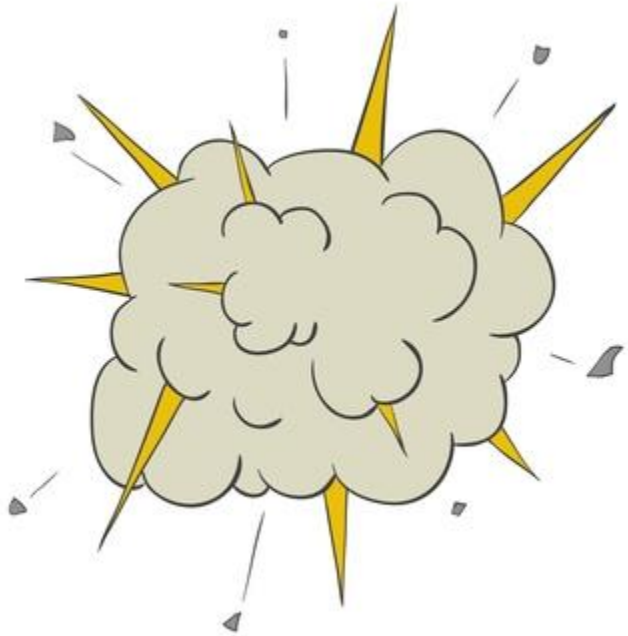
15-213 Recitation: Buffer Lab

Jack Biggs

22 Sep 2014

Reminder

- Bomb lab is due **tomorrow!**
- Buffer lab is released **tomorrow!!**



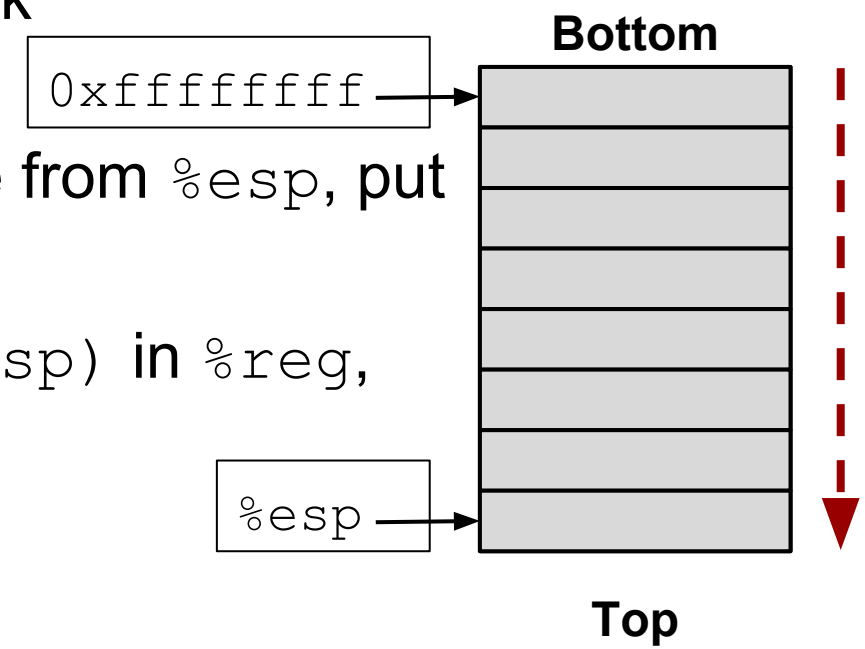
IA32: Register Conventions

- Arguments **not** saved in registers (passed on stack)
- Return value: `%eax`
- Callee-saved: `%eax, %ecx, %edx`
- Caller-saved: `%edx, %edi, %esi`
- Base pointer: `%ebp`
- Stack pointer: `%esp`
- Instruction pointer: `%eip`

Register? I barely knew ‘er!

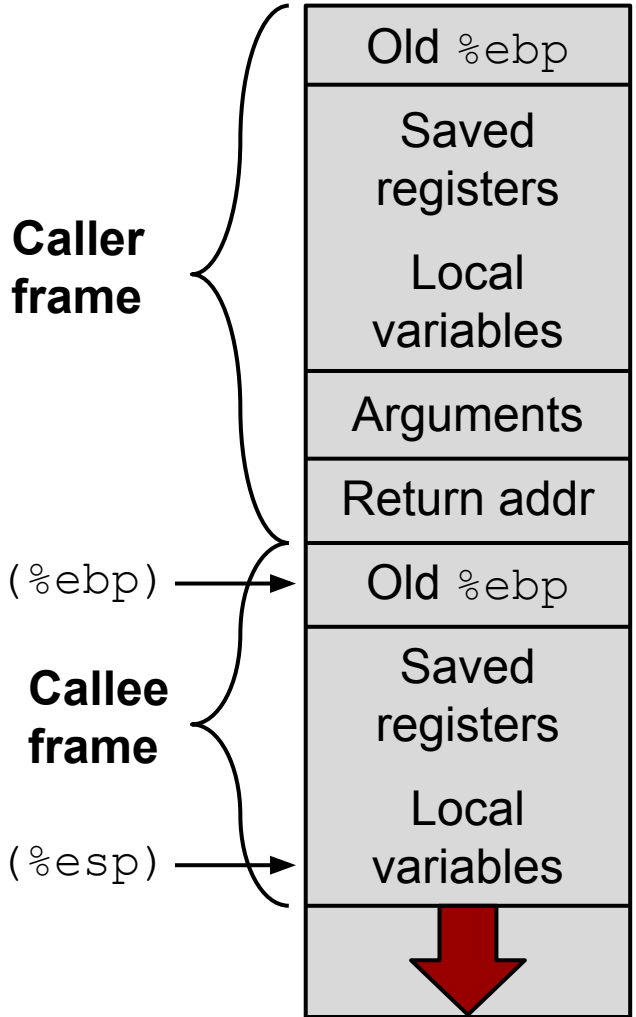
IA32: The Stack

- Grows **downward** towards **lower** memory addresses
- `%esp` points to **top** of stack
- `push %reg`: subtract size from `%esp`, put val in `%reg` at `(%esp)`
- `pop %reg`: put val at `(%esp)` in `%reg`, add size to `%esp`



IA32: Stack Frames

- Every function call has its own **stack frame**.
- Think of a frame as a workspace for each call.
  - Local variables (arrays, structs, scalars)
  - Callee & Caller-saved registers
  - Preparing arguments for a function call



IA32: Function Call Setup

- Push any caller-saved registers that are in use
- Push arguments to stack (in **reverse order**)
- `call foo`: push `%eip` to stack, jump to label `foo`
- Callee: `push %ebp; mov %esp, %ebp;` decrease `%esp` to make room for new frame

IA32: Function Call Return

- At end of function, execute `leave` followed by `ret`
  - `leave: movl %ebp, %esp; popl %ebp`
  - `ret: popl %eip`

Convention Example

```
int main() {
    int x = 3;
    int y = 5;
    foo(x, y);
    return 0;
}

int foo(int x, int y) {
    return x + y;
}
```

```
(gdb) disas main
Dump of assembler code for function main:
   0x080483a2 <+0>:     push   %ebp
   0x080483a3 <+1>:     mov    %esp,%ebp
   0x080483a5 <+3>:     sub    $0x18,%esp
   0x080483a8 <+6>:     movl   $0x3,-0x8(%ebp)
   0x080483af <+13>:    movl   $0x5,-0x4(%ebp)
   0x080483b6 <+20>:    movl   $0x5,0x4(%esp)
   0x080483be <+28>:    movl   $0x3,(%esp)
   0x080483c5 <+35>:    call   0x8048394 <foo>
   0x080483ca <+40>:    mov    $0x0,%eax
   0x080483cf <+45>:    leave
   0x080483d0 <+46>:    ret
End of assembler dump.
(gdb) disas foo
Dump of assembler code for function foo:
   0x08048394 <+0>:     push   %ebp
   0x08048395 <+1>:     mov    %esp,%ebp
   0x08048397 <+3>:     mov    0xc(%ebp),%eax
   0x0804839a <+6>:     mov    0x8(%ebp),%edx
   0x0804839d <+9>:     lea    (%edx,%eax,1),%eax
   0x080483a0 <+12>:    pop    %ebp
   0x080483a1 <+13>:    ret
End of assembler dump.
```

x86_64

- No frame pointers! `%ebp` is free!
- Arguments passed in registers!
- More use of registers in general!
  - Less of stack, because the stack sucks!
- Harder to exploit than IA32. Consider yourselves lucky.

Buffer Lab Overview

- Exploit IA32 by overwriting the stack
- Overflow a buffer, overwrite return address
- Brush up on your IA32 conventions!
- Find out how length of input, what string, etc
  - **Use `gdb`!!!!!!!**

Buffer Lab Tips

- Stack Canaries are special values on the stack
  - Detect overrun of buffer if changed
  - Placed immediately after a buffer
- `nop` sleds
  - `nop` does nothing (no operation)
  - Pad instructions if stack addresses randomize

Buffer Lab Tools

- `./makecookie <andrewid>`
  - Make your cookie appear where it shouldn't
- `./hex2raw`
  - Pass raw ASCII strings to bufbomb
- `./bufbomb -t <andrewid>`
  - Your bomb!
  - Don't worry, it won't explode and cost you points.
- `gcc -m32 <file.c>`
  - Compile exploit code (later on)

Also...

Demonstration!