15-213 Recitation 5: Attack Lab

26 Sept 2016

Agenda

- Reminders
- Stacks
- Attack Lab Activities

Reminders

- **Bomb lab is due tomorrow!**
  - "But if you wait until the last minute, it only takes a minute!" - ***NOT!***
  - Don't waste your grace days on this assignment
- **Attack lab will be released tomorrow!**

Stacks

- **Last-In, First-Out**
  - just like a stack of plates
  - pushes and pops to prese
- **x86 stack grows down**
  - lowest address is “top”

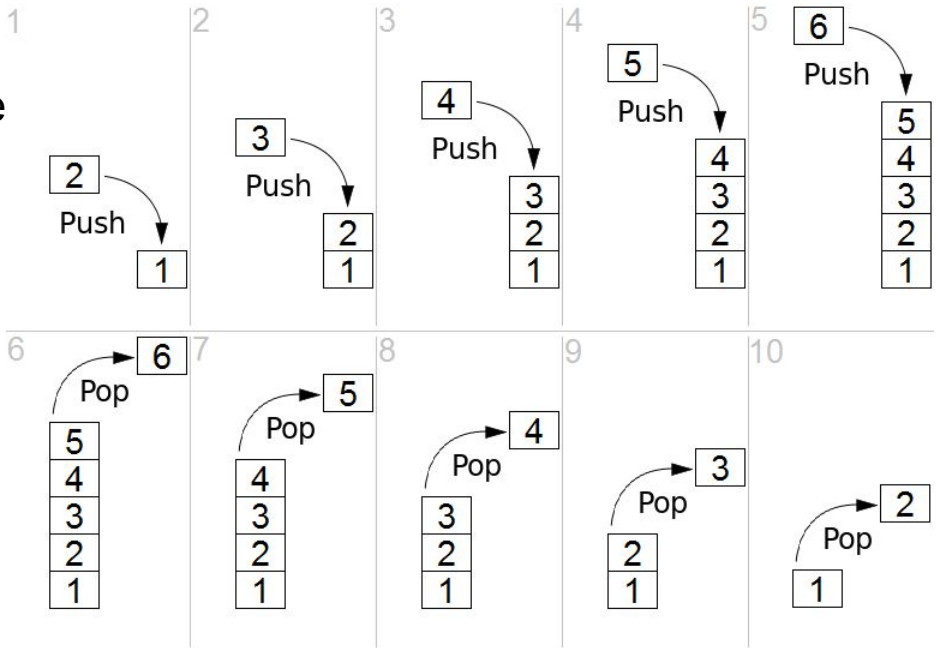


Image credit: Wikimedia Commons

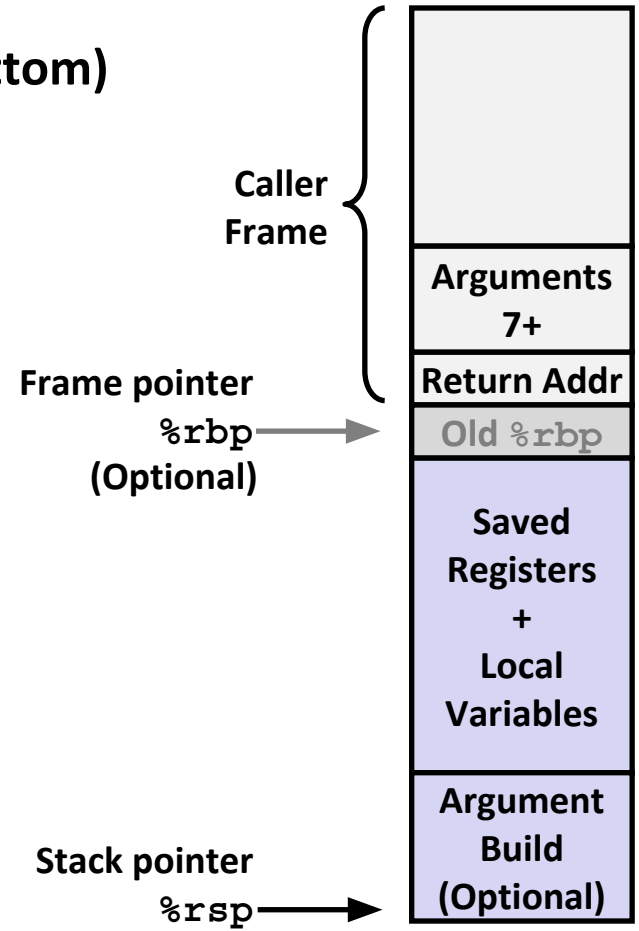
Stack

- **Stack space is allocated in "frames"**
  - Represents the state of a single function invocation
- **Used primarily for two things:**
  - Storing callee save registers
  - Storing the return address of a function
- **Can also store:**
  - Local variables that don't fit in registers
  - Function arguments 7+

x86-64/Linux Stack Frame

- **Current Stack Frame ("Top" to Bottom)**
  - "Argument build:"
    Parameters for function about to call
  - Local variables
    If can't keep in registers
  - Saved register context
  - Old frame pointer (optional)

- **Caller Stack Frame**
  - Return address
    - Pushed by `call` instruction
  - Arguments for this call



Stack Maintenance

- **Functions free their frame before returning**

- **Return instruction looks for the return address at the top of the stack**
  - What if the return address has been changed?

Attack Lab Activities

- **Three activities**
  - Each relies on a specially crafted assembly sequence to purposefully overwrite the stack

- **Activity 1 – Overwrites the return addresses**
- **Activity 2 – Writes an assembly sequence onto the stack**
- **Activity 3 – Uses byte sequences in libc as the instructions**

Form pairs

- One student needs a laptop
- Login to a shark machine

```
$ wget http://www.cs.cmu.edu/~213/activities/rec5.tar
$ tar xf rec5.tar
$ cd rec5
$ make
$ gdb act1
```

Activity 1

(gdb) break clobber

(gdb) run

(gdb) x $rsp

(gdb) backtrace

Q. Does the value at the top of the stack match any frame?

(gdb) x /2gx $rdi // Here are the two key values

(gdb) stepi // Keep doing this until

```
(gdb)
clobber () at support.s:16
16              ret
```

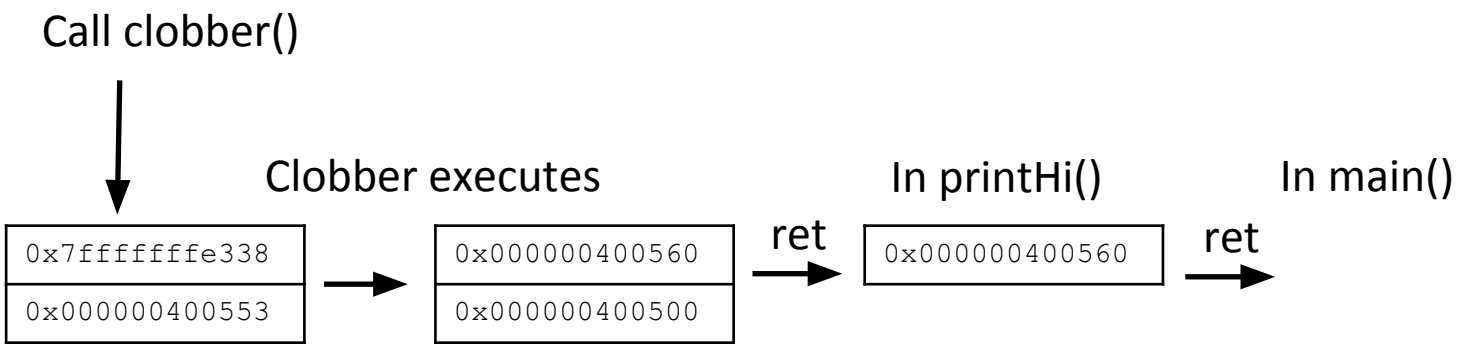
(gdb) x $rsp

Q. Has the return address changed?

(gdb) fin

Activity 1 Post

- **In this activity, we overwrote part of the stack**
  - Placing two return addresses onto the stack
  - Return to printHi()
  - Return to main



Activity 2

$gdb act2

(gdb) break clobber

(gdb) run

(gdb) x $rsp

Q. What is the address of the stack and the return address?

(gdb) x /4gx $rdi

Q. What will the new return address be?
(i.e., what is the first value?)

(gdb) x/5i $rdi + 8 // Display as instructions

Q. Why rdi + 8?

Q. What are the three addresses?

(gdb) break puts

(gdb) break exit

Q. Do these addresses look familiar?

Activity 2 Post

- **Normally programs cannot execute instructions on the stack**
  - Main used mprotect to change the memory protection for this activity
- **Clobber wrote a return address of the stack to the stack**
  - And a sequence of instructions
  - Three addresses: “Hi\n”, puts(), exit()
- **Why callq *%rsi?**
  - As the attacklab writeup notes, calling functions is hard.
  - Return oriented programming is much easier.

Activity 3

$gdb act3

(gdb) break clobber

(gdb) run

(gdb) x /5gx $rdi

Q. Which value will be first on the stack?

Q. At the end of clobber, where will it return?

(gdb) x /2i <return address>

Q. What does this sequence do?

Q. Do the same for the other addresses. Note that some are return addresses and some are for data. When you continue, what will the code now do?

How was it constructed?

- **Think of possible executions**
- **What are the bytes of the instructions?**
  - Write short assembly into foo.s
  - gcc -c foo.s
  - objdump -d foo.o
  - OR: Convert them to byte sequences (Attacklab write-up has a table)
    - Also important so you can switch between register names
- **After determining the desired instruction(s)**
  - Use the Linux tool xxd to dump the raw bytes to a file
  - Or: Objdump -d rtarget (or act3 or …)
  - Search the file

If You Get Stuck

- Please read the writeup. *Please read the writeup. Please read the writeup.* ***Please read the writeup!***
- CS:APP Chapter 3
- View lecture notes and course FAQ at http://www.cs.cmu.edu/~213
- Office hours Sunday through Thursday 5:00-9:00pm in WeH 5207
- Post a private question on Piazza
- man gdb, gdb's help command

Remember...

Appendix

Attack Lab Tools

- **gcc -c file.s**
  - convert the assembly code in file.s to object code in file.o
- **objdump -d file.o**
  - disassemble the code in file.o; shows the actual bytes for the instructions
- **./hex2raw**
  - convert hex codes into raw ASCII strings to pass to targets
- **gdb**
  - determine stack addresses
- **paper and pencil**
  - for drawing stack diagrams

More Useful GDB Commands

x/[*n*]i <address>	disassemble *n* instructions at <address>
b <loc> if <cond>	conditional breakpoint, stop only if <cond> true
cond <bp> <cond>	add condition to existing breakpoint <bp>
commands <bp>	execute commands when breakpoint <bp> hit
tbreak <loc>	set temporary breakpoint – auto-deletes when hit!
finish	run until current frame (function) returns, and print return value
layout asm	split the screen into separate disassembly and command windows
layout reg	show register window as well (after layout