Attack Lab

Agenda

- Stacks
- Attack Lab

Stacks

- **Last-in, first-out**
- **x86 stack grows down**
  - lowest address is “top”
  - $rsp contains the address of the topmost element in the stack
- **Uses the pushq and popq instructions to push and pop registers/constants onto and off the stack**

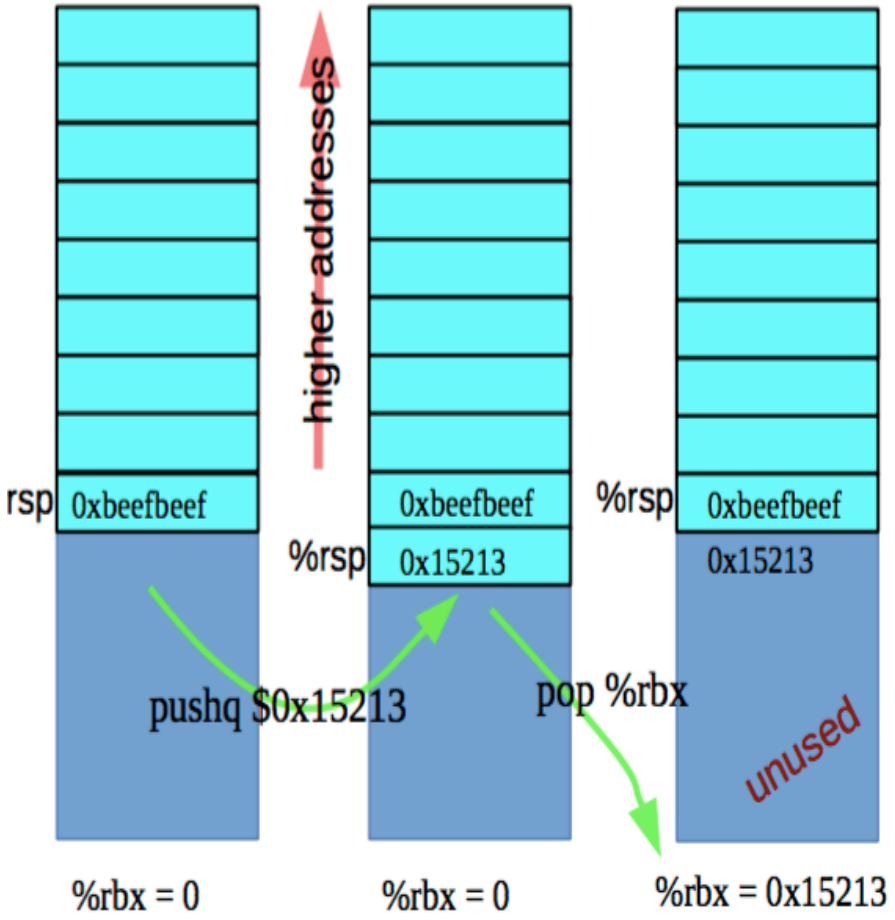
Stack – pushq & popq

- **pushq {value} is equivalent to**

  sub $8, %rsp
  mov {value}, (%rsp)

- **popq {reg} is equivalent to**

  mov (%rsp), {reg}
  add $8, %rsp



Stack – Caller vs. Callee

- **Function A calls function B**
  - A is the caller
  - B is the callee
- **Stack space is allocated in “frames”**
  - Represents the state of a single function invocation
- **Frame used primarily for two things:**
  - Storing callee saved registers
  - Storing the return address of a funciton

Registers – Caller-saved vs. Callee-saved

- **Caller-saved**
  - Registers used for function arguments are always caller-saved
  - $rax is also caller-saved
  - Called function may do as it wishes with the registers
  - Must save/restore register in caller's stack frame if it still needs the value after a function call
- **Callee-saved**
  - If the function wants to change the register, it must save the original value in its stack frame and restore it before returning
  - The calling function may store temporary values in callee-saved registers

x86-64 Register Usage Conventions

%rax	return value
%rbx	callee saves
%rcx	argument #4
%rdx	argument #3
%rsi	argument #2
%rdi	argument #1
%rsp	stack pointer
%rbp	callee saves

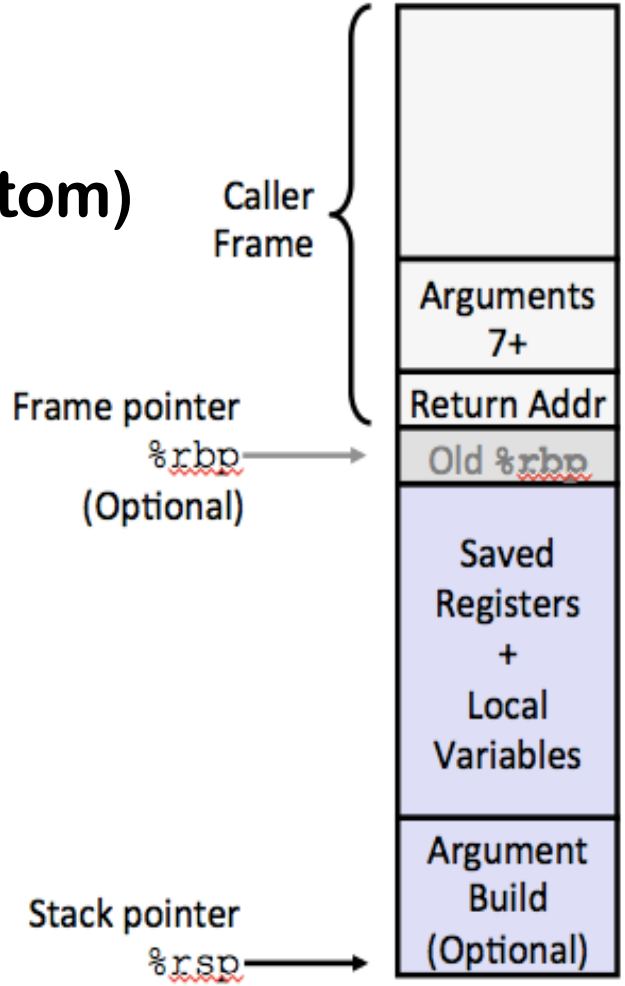
%r8	argument #5
%r9	argument #6
%r10	caller saves
%r11	caller saves
%r12	callee saves
%r13	callee saves
%r14	callee saves
%r15	callee saves

Registers – Caller-saved vs. Callee-saved

- **Before function call**
  - rdi = first argument
  - rsi = second argument
  - rax = some temporary value
  - rbx = some important number to use later (15213)
  - rsp = pointer to some important buffer (0x7fffffffaaaa)
- **After function call**
  - rdi = garbage
  - rsi = garbage
  - rax = return value
  - rbx = some important number to use later (15213)
  - rsp = pointer to some important buffer (0x7fffffffaaaa)

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

- We're letting you hijack programs(ctarget,rtarget) by running buffer overflow attacks on them.
  - Is that not justification enough?
- To understand stack discipline and stack frames
- To defeat relatively secure programs with return oriented programming

Attack Lab Tools

- **$gcc –c** test.s

  $objdump –d test.o > test.asm

  Compiles the assembly code in test.s and then shows the actual bytes for the instructions

- **$./hex2raw** < exploit.txt > exploit.bin

  Convert hex codes in exploit.txt into raw binary strings to pass to targets

- (gdb) **display /12gx** $rsp

  (gdb) **display /2i** $rip

  Displays 12 elements on the stack and the next 2 instructions to run

GDB is also useful to for tracing to see if an exploit is working

If you get stuck

- **Please read the writeup carefully.** Not everything will make sense on the first read-through.
- Other resources you can make use of:
  - CS:APP Chapter 3
  - Lecture slides and videos
  - x86-64 and GDB cheat sheets under Resources