Intro Stack Buffer Overflow Workshop

UC Davis Cybersecurity Club
Nate Buttke

May 2, 2022

Icebreaker

What do you remember from last time?

OR

How familiar are you with assembly?

Key points from last time

- **Endianness**
  - Little endian: least significant byte has the lowest address
  - Big endian: Most significant byte has the lowest address
- Local variables should be next to each other in memory (on the stack). `gets` and `strcpy` are not secure.
- **gdb**
  - `disassemble fn` to view a function in assembly.
  - set breakpoints by copying and pasting the hex for instructions
  - To print stack data: `x/24wx $esp` (`x/<number><width><format> <address>`)
  - `x/10i $rip` prints 10 instructions after IP
  - `define hook-stop` to set commands that will run after each pause (end your input with `end`).

How do functions work?

- If the computer starts at some point in memory (`%eip`), and executes instructions sequentially, how can we redirect control flow?

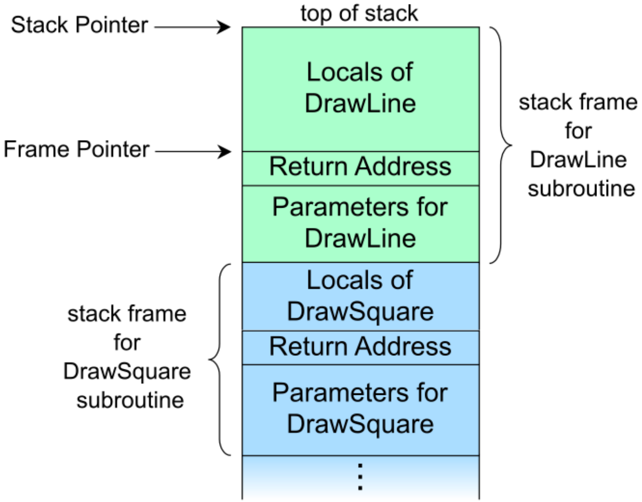
How do functions work?

- If the computer starts at some point in memory (`%eip`), and executes instructions sequentially, how can we redirect control flow?
- Modify `%eip`! Just put the address of the function's first instruction into `%eip`. In assembly, we keep track of function locations with labels, e.g. `jmp label`.
- We also need to know where to return to. So, we push the return address onto a part of memory called *the stack*.

The stack

- The stack is a conventional part of computer memory
- It is a first-in-first-out (FIFO) data structure
- Contains return addresses, local variables, arguments, and *stack frames* for each function. This allows backtracing, among other things.
- In assembly programming, the stack is used mostly through `push` and `pop` instructions
- On Intel, the stack grows toward 0. So, the stack includes `(%esp)` and all higher addresses.

The stack (grows *toward 0*)



R. S. Shaw, Public domain, via Wikimedia Commons

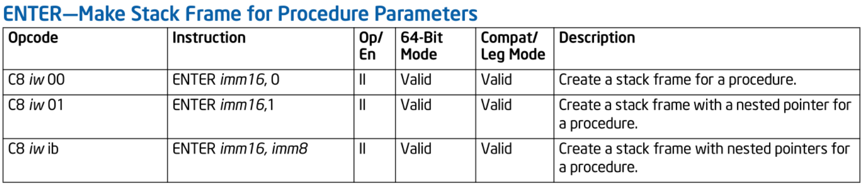
This makes more sense in action (Let's work through some assembly)

```
#max call
  pushl cats
  pushl dogs
  call max
  popl %edx
  popl %edx
```

this function has *no local variables.* Look at the enter instruction.

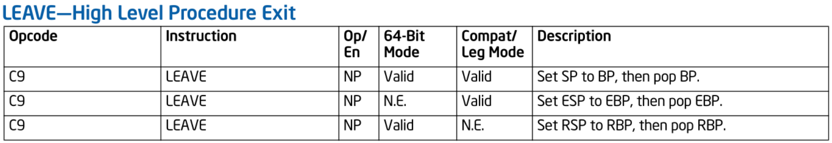
```
#max function
max:
  enter $0, $0
  movl 8(%ebp), %eax #arg1
  cmpl %eax, 12(%ebp) #arg2
  jg max_true
  max_false:
    movl 8(%ebp), %eax
    jmp max_end
  max_true:
    movl 12(%ebp), %eax
    jmp max_end
  max_end:
    leave
    ret
```

enter and leave



ENTER—Make Stack Frame for Procedure Parameters

Opcode	Instruction	Op/ En	64-Bit Mode	Compat/ Leg Mode	Description
C8 *iw* 00	ENTER *imm16*, 0	II	Valid	Valid	Create a stack frame for a procedure.
C8 *iw* 01	ENTER *imm16*,1	II	Valid	Valid	Create a stack frame with a nested pointer for a procedure.
C8 *iw* ib	ENTER *imm16*, *imm8*	II	Valid	Valid	Create a stack frame with nested pointers for a procedure.



LEAVE—High Level Procedure Exit

Opcode	Instruction	Op/ En	64-Bit Mode	Compat/ Leg Mode	Description
C9	LEAVE	NP	Valid	Valid	Set SP to BP, then pop BP.
C9	LEAVE	NP	N.E.	Valid	Set ESP to EBP, then pop EBP.
C9	LEAVE	NP	Valid	N.E.	Set RSP to RBP, then pop RBP.