Stack Overflows

An Intro

Stuart Nevans Locke

Background

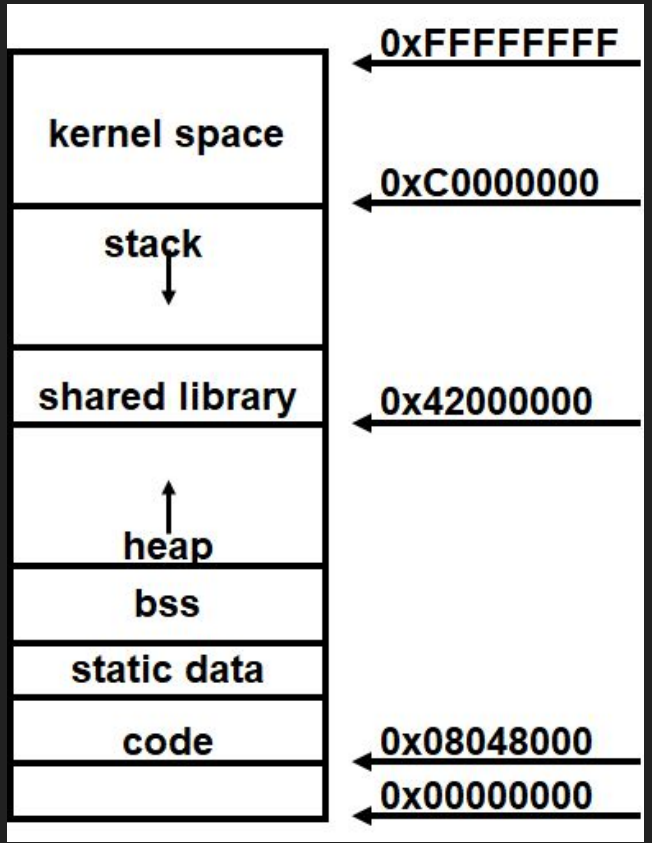
- Ability to read C code
- Minimal knowledge of assembly (mov, push, pop, call, ret)
- Rough understanding of the stack

Overview

- Memory and Stack Layout
- C Calling Conventions
- Stack Overflows
  - Demo
- Mitigations
  - DEP
  - ASLR
  - Stack Canaries
- Tools
  - GDB
  - Cutter (radare2 gui)
- More Demos

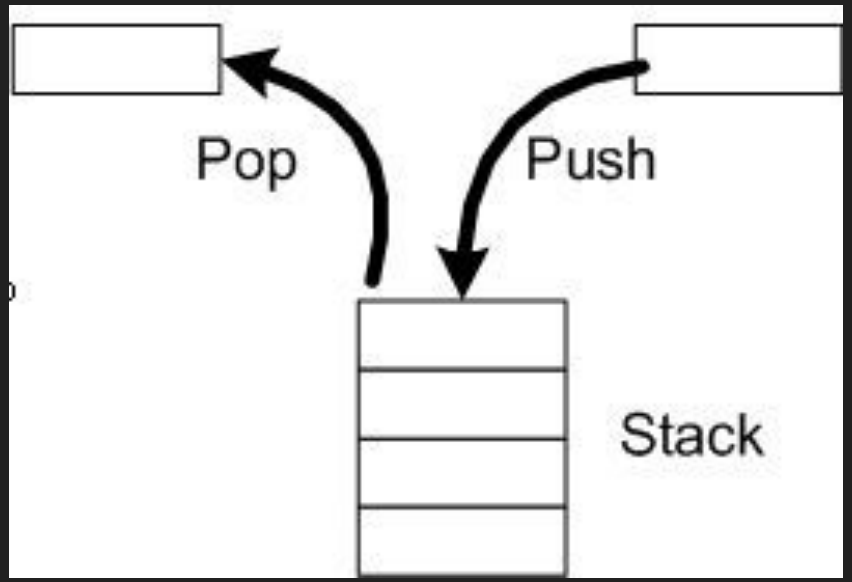
Memory Layout

- Important Stuff:
  - The stack
  - Code section (.text)
  - Data sections
    - Later: the heap



Introduction to the Stack

- Interesting Registers:
  - RBP (Base Pointer) , RSP (Stack Pointer)
- Interesting Instructions
  - *push register*
    - rsp-=8
    - mov [rsp], register
  - *pop register*
    - mov register, [rsp]
    - rsp+=8



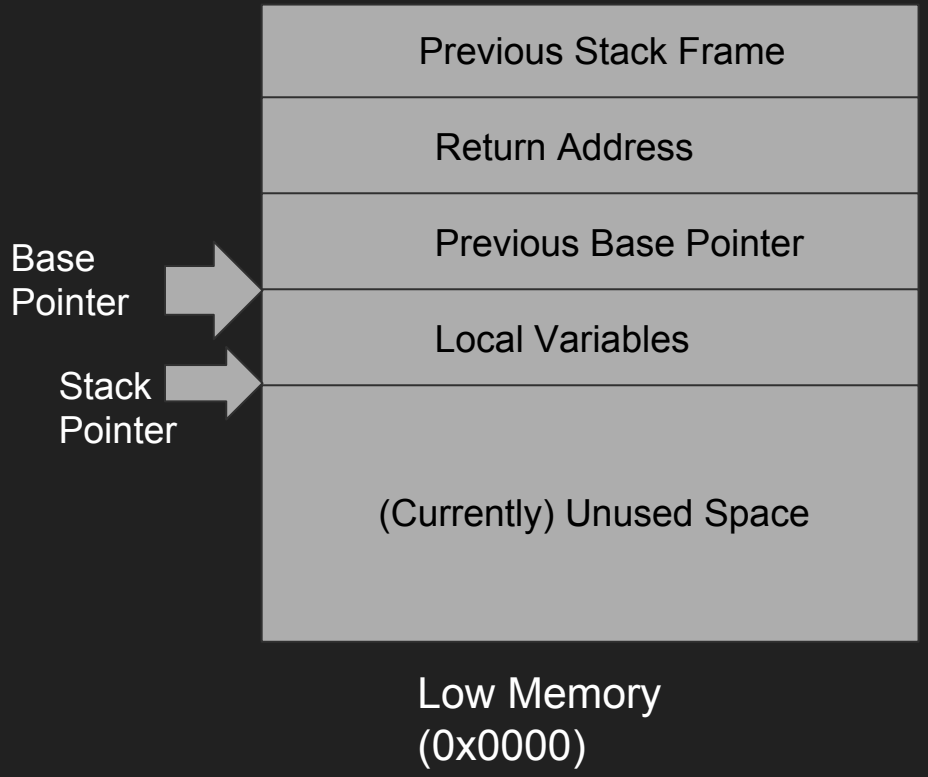
Stack Frames

```
void func(){
    char x[16];
    puts("Stack Frames!");
    x[0]=1;
    x[15]=15;
}
```

```
push    rbp
mov     rbp, rsp
sub     rsp, 16
mov     edi, OFFSET FLAT:.LC0
call    puts
mov     BYTE PTR [rbp-16], 1
mov     BYTE PTR [rbp-1], 15
nop
leave
ret
```

Stack Frames

```
push    rbp
mov     rbp, rsp
sub     rsp, 16
mov     edi, OFFSET FLAT:.LC0
call    puts
mov     BYTE PTR [rbp-16], 1
mov     BYTE PTR [rbp-1], 15
nop
leave
ret
```



C Calling Conventions

__cdecl

- All arguments go on the stack
- Caller cleans up
- Ex:
  - push rdi
  - call puts
  - add rsp,8

__stdcall

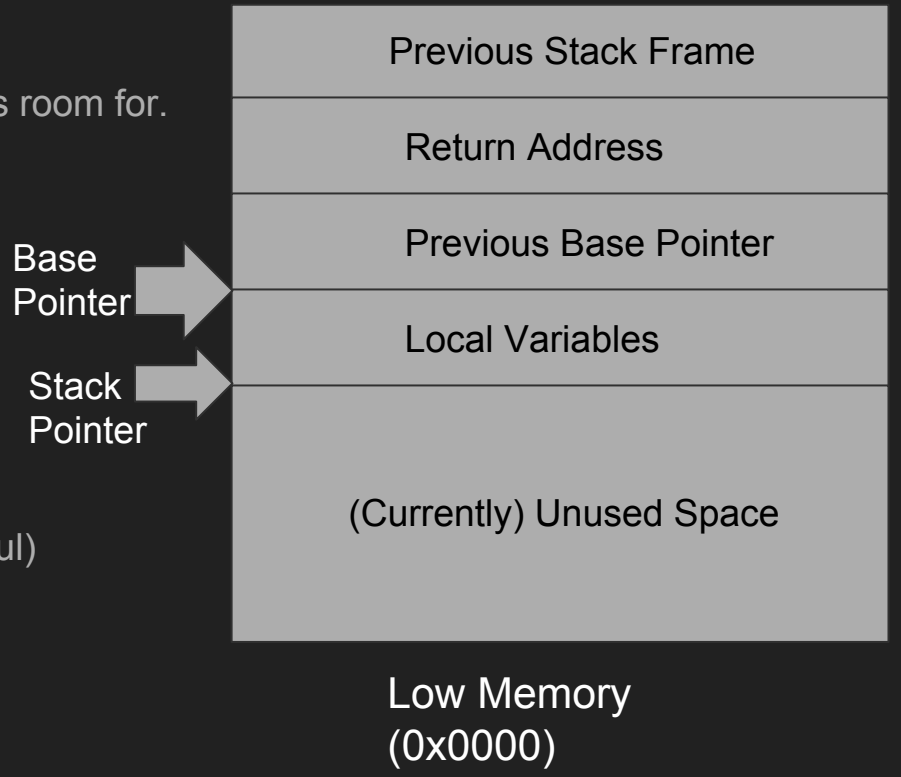
- All arguments go on the stack
- Callee cleans up
- Ex:
  - push rdi
  - call puts

__fastcall

- Tries to put arguments in registers
- Ex:
  - mov rdi, 0xaddress
  - call puts

Stack Overflows

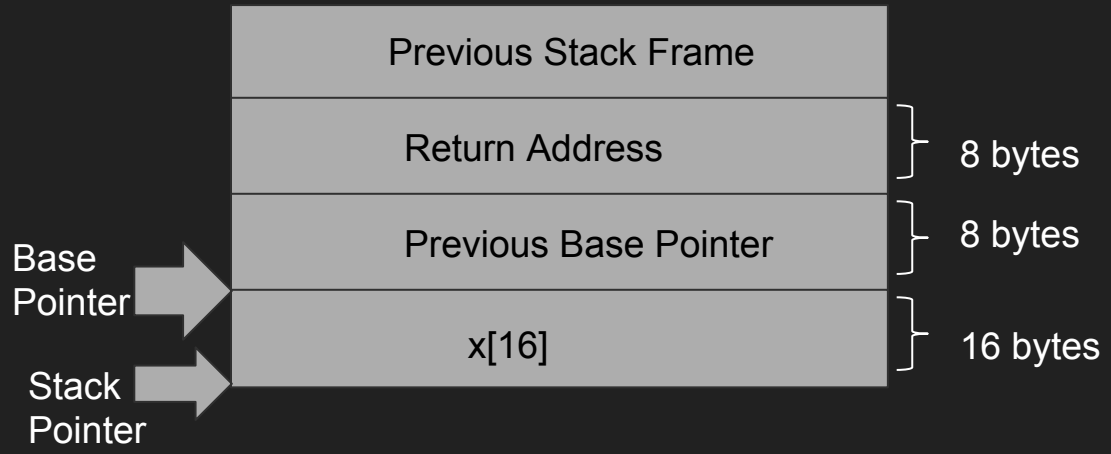
- What is a stack overflow?
  - More data is read onto the stack than there is room for.
- Example:
  - char x[16];
  - fgets(x, 32, stdin);
  - We just read 32 bytes into a 16 byte buffer
- What can we overwrite?
  - Local Variables
  - Return Address
  - Previous base pointer (seldom (never?) useful)



Demo

```
void callme(){
	system("/bin/bash");
}
int main(){
	char x[16];
	fgets(x,32,stdin);
	return 1;
}
```

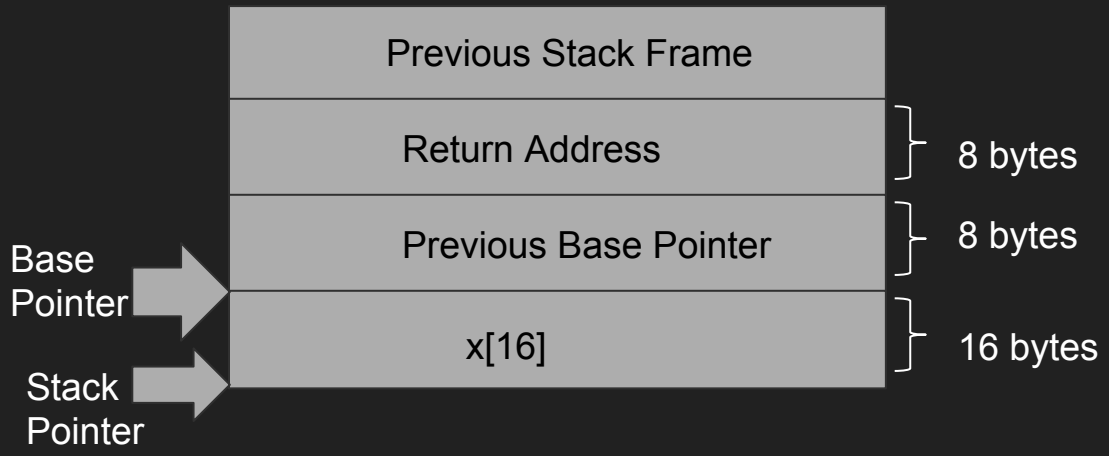
- We want to execute callme
- How can we do that?
  - Overwrite the return address
  - GDB tip:
    - Print callme
    - Prints the address of callme



Demo

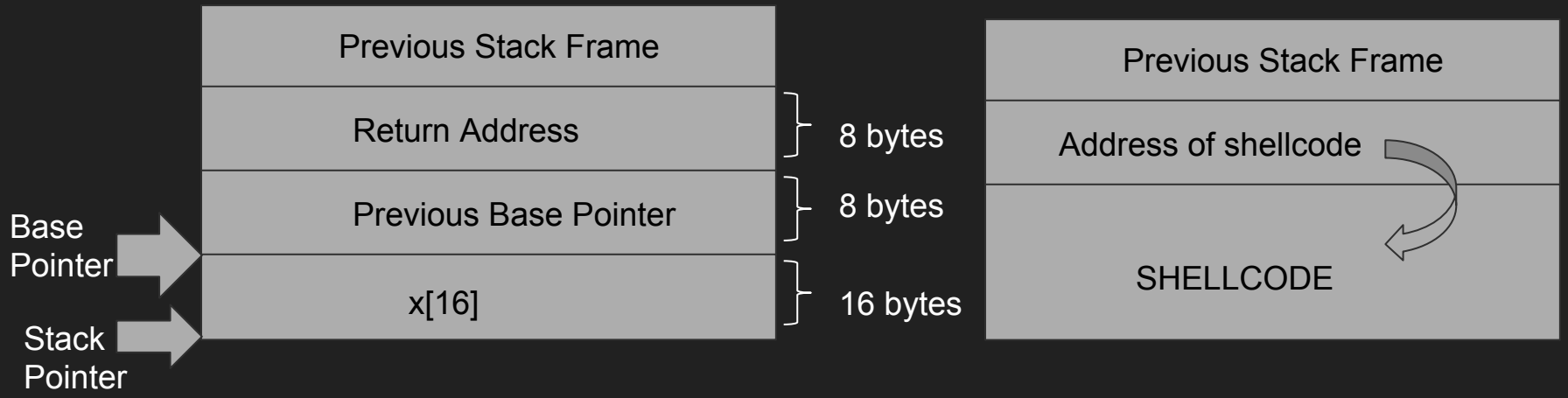
```
void callme(){
	system("/bin/bash");
}
int main(){
	char x[16];
	fgets(x,32,stdin);
	return 1;
}
```

- Writing an exploit
  - How can we write unprintable characters?
    - python -c “print ‘string’ + ‘\xda\x2f\x00\x2f’ ”
  - Problem we overwrite return address but can’t interact with the shell
    - To allow yourself to interact, do:
    - (python -c “print ‘exploit’” ; cat) | ./binary



Stack Overflow Cont.

- In the real world, there's no callme
- What do we do?
  - Send shellcode
    - Code that if run will run a shell
  - Instead of returning to a function, we return to wherever we put the shellcode

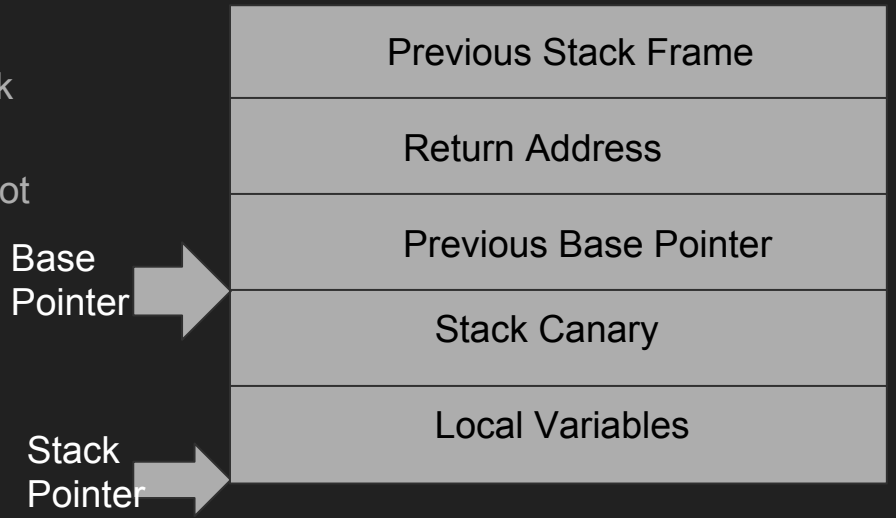


Mitigations

- DEP
  - Data Execution Prevention
  - Also known as NX(Non Executable), W^X(Writeable XOR Executable)
  - Makes the stack and heap (and other data sections) non executable
- ASLR
  - Address Space Layout Randomization
  - Loads the stack and heap into random locations
  - DOES NOT NECESSARILY APPLY TO TARGET BINARY
    - callme would always have to same address
- Both of these prevent us from merely putting shellcode on the stack and returning into that
- For bypassing them, see next presentation

Mitigations (Cont.)

- Stack Canaries
  - Detects stack overflows and aborts program if found
  - In every function
    - When called
      - Loads a secret value onto the stack
    - Before returning
      - Checks that the secret value has not changed
  - When overwriting the return address, the stack canary's value is overwritten
- No great bypasses
  - Leak the value
  - Change the local variables, hope one of them is really important



Tools - GDB

Load a binary	gdb /path/to/file
Set breakpoint	b func, b *0xaddress
See registers	info registers, (i r), i r rax
View Stack Frame (Useful for overflows)	info frame
Examine Memory	x Lots of options x/2 $rax shows 2 words at the address of rax x/2xs 0xaddress shows 2 strings x/xg shows a giant word(8 bytes)(64 bit pointer)
View Assembly Default View	layout asm Ctrl+X+A

Tools - Cutter

- Used for static analysis
- GUI for radare2
- Nicer than gdb for reading assembly
- Graph Mode
  - Space to activate
- Pseudocode Window
  - Very much a work in progress

Questions?
stnevans.me/binex/1/