Examining Buffer Overflow with GDB
Reminders

- PA1 deadline is tonight by 11:59 PM PDT
- PA2 goes out on Monday and is on Buffer Overflows!
Memory Layout

- high address
- command-line arguments and environment variables
- stack
- initialized by exec
- read from program file by exec
- heap
- uninitialized data (bss)
- initialized by exec
- initialized data
- initialized text
- low address
Stack Layout

High Memory

- arguments
- saved %eip
- saved %ebp
- other saved registers
- local variables

Low Memory

%eip is a register pointing to the instruction that CPU will execute in next cycle.

saved %eip references to a 4-byte address value stored on stack.

saved %eip is stored on stack when a function call is made. It has the address of where to resume execution in the caller function.

When a function returns, the saved %eip value will be popped into the register %eip → control will transfer to where saved %eip points to.

return address == saved return address == saved %eip == %ebp+4
Basic Buffer Overflow

int main(int argc, char *argv[]) {
    char name[5];
    gets(name);
    printf("Hi %s!", name);
    exit(0);
}

- How does `gets` copy `stdin` into `name`?
- When does `gets` stop copying?
# Executing Shell Code

```c
int main(int argc, char *argv[]) {
    char name[100];
    gets(name);
    printf("Hi %s!", name);
    exit(0);
}
```

- Where do I place the shellcode?
- How does ensure the shellcode is always executed?

**High Memory**

- saved %eip
- saved %ebp
- saved registers
- Shellcode

**Low Memory**
Helpful links

- https://www.youtube.com/watch?v=1S0aBV-Waeo
- https://www.youtube.com/watch?v=hJ8lwyhqzD4