Buffer Overrun Review
Process Layout in Memory

- **Stack**
  - grows towards decreasing addresses.
  - is initialized at run-time.

- **Heap**
  - grow towards increasing addresses.
  - is initialized at run-time.

- **BSS section**
  - size fixed at compile-time.
  - is initialized at run-time.
  - was grouped into Data in CS61C.

- **Data section**
  - is initialized at compile-time.

- **Text section**
  - holds the program instructions (read-only).
IA-32 Notation

• Format : inst src dst
• registers prefixed with a ‘%’, constants with ‘$’
• (%ebx) means accessing the memory address stored in register %ebx
• l suffix on instruction indicates a “long-word” (32-bit) instruction

Important Registers

• %eax, %ebx, %ecx, %edx, %edi, %esi – general purpose registers (%eax used to store return value)
• %ebp – base pointer. Indicates start of stack frame
• %esp – stack pointer. Indicates bottom on stack
• %eip – instruction pointer. Indicates instruction to run
Common Instructions

- `mov a, b` – copy value of `a` into `b`
- `push a` – push `a` onto the stack (decrement stack, copy value over)
- `pop a` – pop data from stack into `a` (copy value over, increment stack)
- `call func` – push address of next instruction onto stack & transfer control to `func`
- `ret` – pop return address off stack and jump to it
- `leave` – syntactic sugar for `mov %ebp, %esp` followed by `pop %ebp` (restores prev stack frame)
Function Structure

foo:

push %ebp
mov %esp, %ebp
sub $???, %esp
...
...
leave
ret

| Function prologue (creates a new stack frame)
| Function body
| Function epilogue (restore old stack frame & jump to return address)
Calling Convention

foo:

...  
...  
...  
push $5  
push $1  
call bar
add $8, %esp
...  
...
...

Example of calling bar(1,5)
Arguments pushed on stack in reverse order (last argument pushed first)
Stack restored after function call end
Extra: Caller also sometimes need to save registers, but don’t worry about this too much
Let’s walk through an example!

• Terminology:
  – SFP : saved %ebp on the stack
  – OFP : old %ebp from the previous stack frame
  – RIP : return address on the stack
Function Calls

void foo(int a, int b, int c)
{
    int bar[2];
    char qux[3];
    bar[0] = 'A';
    qux[0] = 0x42;
}

int main(void)
{
    int i = 1;
    foo(1, 2, 3);
    return 0;
}
Function Calls in Assembler

```c
int main(void)
{
    int i = 1;
    foo(1, 2, 3);
    return 0;
}
```

```
main:
pushl %ebp
movl %esp,%ebp
subl $4,%esp
movl $1,-4(%ebp)
pushl $3
pushl $2
pushl $1
call foo
addl $12,%esp
xorl %eax,%eax
leave
ret
```
Function Calls in Assembler

```c
int main(void)
{
    int i = 1;
    foo(1, 2, 3);
    return 0;
}
```

main:

- `pushl %ebp`
- `movl %esp,%ebp`
- `subl $4,%esp`
- `movl $1,-4(%ebp)`
- `pushl $3`
- `pushl $2`
- `pushl $1`
- `call foo`
- `addl $12,%esp`
- `xorl %eax,%eax`
- `leave`
- `ret`
Function Calls in Assembler

```c
int main(void)
{
    int i = 1;
    foo(1, 2, 3);
    return 0;
}

main:
    pushl %ebp
    movl %esp,%ebp
    subl $4,%esp
    movl $1,-4(%ebp)
    pushl $3
    pushl $2
    pushl $1
    call foo
    addl $12,%esp
    xorl %eax,%eax
    leave
    ret
```
Function Calls in Assembler

```c
int main(void)
{
    int i = 1;
    foo(1, 2, 3);
    return 0;
}
```

**main:**
- pushl %ebp
- movl %esp,%ebp
- subl $4,%esp
- movl $1,-4(%ebp)
- pushl $3
- pushl $2
- pushl $1
- call foo
- addl $12,%esp
- xorl %eax,%eax
- leave
- ret

---

Function Calls
int main(void)
{
    int i = 1;
    foo(1, 2, 3);
    return 0;
}

main:
pushl %ebp
movl %esp,%ebp
subl $4,%esp
movl $1,-4(%ebp)
pushl $3
pushl $2
pushl $1
call foo
addl $12,%esp
xorl %eax,%eax
leave
ret
Function Calls in Assembler

```c
int main(void)
{
    int i = 1;
    foo(1, 2, 3);
    return 0;
}
```

```
main:
pushl %ebp
movl %esp,%ebp
subl $4,%esp
movl $1,-4(%ebp)
pushl $3
pushl $2
pushl $1
call foo
addl $12,%esp
xorl %eax,%eax
leave
ret
```
Function Calls in Assembler

```c
int main(void)
{
    int i = 1;
    foo(1, 2, 3);
    return 0;
}

main:
pushl %ebp
movl %esp,%ebp
subl $4,%esp
movl $1,-4(%ebp)
pushl $3
pushl $2
pushl $1
call foo
addl $12,%esp
xorl %eax,%eax
leave
ret
```
Function Calls in Assembler

```c
void foo(int a, int b, int c) {
    int bar[2];
    char qux[3];
    bar[0] = 'A';
    qux[0] = 0x42;
}
```

```
foo:
pushl %ebp
movl %esp,%ebp
subl $12,%esp
movl $65,-8(%ebp)
movb $66,-12(%ebp)
leave
ret
```
Function Calls in Assembler

```c
void foo(int a, int b, int c)
{
    int bar[2];
    char qux[3];
    bar[0] = 'A';
    qux[0] = 0x42;
}

foo:
    pushl %ebp
    movl %esp,%ebp
    subl $12,%esp
    movl $65,-8(%ebp)
    movb $66,-12(%ebp)
    leave
    ret
```
Function Calls in Assembler

```c
void foo(int a, int b, int c)
{
    int bar[2];
    char qux[3];
    bar[0] = 'A';
    qux[0] = 0x42;
}

foo:
    pushl %ebp
    movl %esp,%ebp
    subl $12,%esp
    movl $65,-8(%ebp)
    movb $66,-12(%ebp)
    leave
    ret
```
void foo(int a, int b, int c)
{
    int bar[2];
    char qux[3];
    bar[0] = 'A';
    qux[0] = 0x42;
}

foo:
    pushl %ebp
    movl %esp,%ebp
    subl $12,%esp
    movl $65,-8(%ebp)
    movb $66,-12(%ebp)
    leave
    ret
Function Calls in Assembler

```c
void foo(int a, int b, int c)
{
    int bar[2];
    char qux[3];
    bar[0] = 'A';
    qux[0] = 0x42;
}
```

```assembly
foo:
    pushl %ebp
    movl %esp,%ebp
    subl $12,%esp
    movl $65,-8(%ebp)
    movb $66,-12(%ebp)
    leave
    ret
```
Function Calls in Assembler

```c
void foo(int a, int b, int c)
{
    int bar[2];
    char qux[3];
    bar[0] = 'A';
    qux[0] = 0x42;
}
```

```assembly
foo:
pushl  %ebp
movl  %esp,%ebp
subl  $12,%esp
movl  $65,-8(%ebp)
movb  $66,-12(%ebp)
leave
ret
```
Function Calls in Assembler

```c
void foo(int a, int b, int c)
{
    int bar[2];
    char qux[3];
    bar[0] = 'A';
    qux[0] = 0x42;
}

foo:
    pushl %ebp
    movl %esp,%ebp
    subl $12,%esp
    movl $65,-8(%ebp)
    movb $66,-12(%ebp)
    leave
    ret
```

![Stack Frame Diagram]

---

Function Calls
Function Calls in Assembler

```c
void foo(int a, int b, int c)
{
    int bar[2];
    char qux[3];
    bar[0] = 'A';
    qux[0] = 0x42;
}

foo:
    pushl %ebp
    movl %esp,%ebp
    subl $12,%esp
    movl $65,-8(%ebp)
    movb $66,-12(%ebp)
    leave
    ret
```

Diagram showing the stack layout with labels for `ebp`, `esp`, `sfp`, `rip`, `ofp`, and the `leave` instructions.
Function Calls in Assembler

```c
void foo(int a, int b, int c)
{
    int bar[2];
    char qux[3];
    bar[0] = 'A';
    qux[0] = 0x42;
}
```

---

```c
foo:
pushl  %ebp
movl  %esp,%ebp
subl  $12,%esp
movl  $65,-8(%ebp)
movb  $66,-12(%ebp)
leave
ret
ret
```

---

Function Calls
Function Calls in Assembler

```c
int main(void)
{
    int i = 1;
    foo(1, 2, 3);
    return 0;
}

main:
    pushl  %ebp
    movl  %esp,%ebp
    subl  $4,%esp
    movl  $1,-4(%ebp)
    pushl $3
    pushl $2
    pushl $1
    call  foo
    addl  $12,%esp
    xorl  %eax,%eax
    leave
    ret
```
Function Calls in Assembler

```c
int main(void)
{
    int i = 1;
    foo(1, 2, 3);
    return 0;
}
```

```
main:
pushl %ebp
movl %esp,%ebp
subl $4,%esp
movl $1,-4(%ebp)
pushl $3
pushl $2
pushl $1
call foo
addl $12,%esp
xorl %eax,%eax
leave
ret
```
Function Calls in Assembler

```c
int main(void)
{
    int i = 1;
    foo(1, 2, 3);
    return 0;
}
```

```
main:
pushl  %ebp
movl  %esp,%ebp
subl  $4,%esp
movl  $1,-4(%ebp)
pushl $3
pushl $2
pushl $1
call  foo
addl  $12,%esp
xorl  %eax,%eax
leave
ret
```
Function Calls in Assembler

```c
int main(void)
{
    int i = 1;
    foo(1, 2, 3);
    return 0;
}
```

```
main:
pushl %ebp
movl %esp,%ebp
subl $4,%esp
movl $1,-4(%ebp)
pushl $3
pushl $2
pushl $1
call foo
addl $12,%esp
xorl %eax,%eax
leave
ret
```
Function Calls in Assembler

```plaintext
int main(void)
{
    int i = 1;
    foo(1, 2, 3);
    return 0;
}

main:
    pushl %ebp
    movl %esp,%ebp
    subl $4,%esp
    movl $1,-4(%ebp)
    pushl $3
    pushl $2
    pushl $1
    call foo
    addl $12,%esp
    xorl %eax,%eax
    leave
    ret
```
```c
int main(void)
{
    int i = 1;
    foo(1, 2, 3);
    return 0;
}

main:
pushl  %ebp
movl  %esp,%ebp
subl  $4,%esp
movl  $1,-4(%ebp)
pushl $3
pushl $2
pushl $1
call  foo
addl  $12,%esp
xorl  %eax,%eax
leave
ret
```
Buffer Overflow

- C is not memory safe, many functions can write past buffers
  - Ex: strcpy(), gets(), etc

<table>
<thead>
<tr>
<th>Return Address</th>
</tr>
</thead>
<tbody>
<tr>
<td>Previous Base Pointer</td>
</tr>
<tr>
<td>local char[] buffer</td>
</tr>
</tbody>
</table>

Writing past length of char[] buffer will begin to overwrite things on the stack like previous base pointer and return address.

Overwriting return address can allow program to jump to wherever attacker wants.