

# Lecture 9: Buffer Overflows

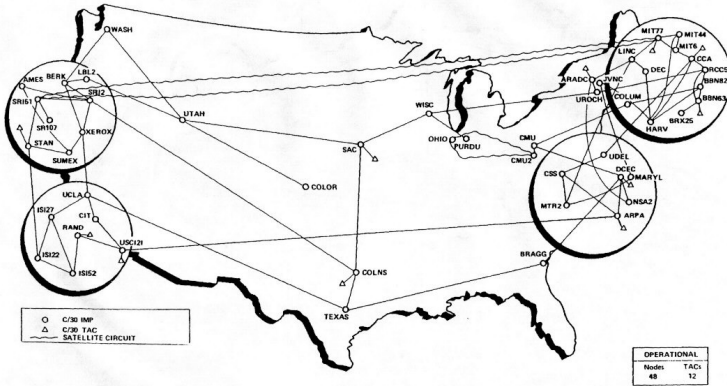
---

CS 105

Spring 2024

# Buffer Overflow Examples

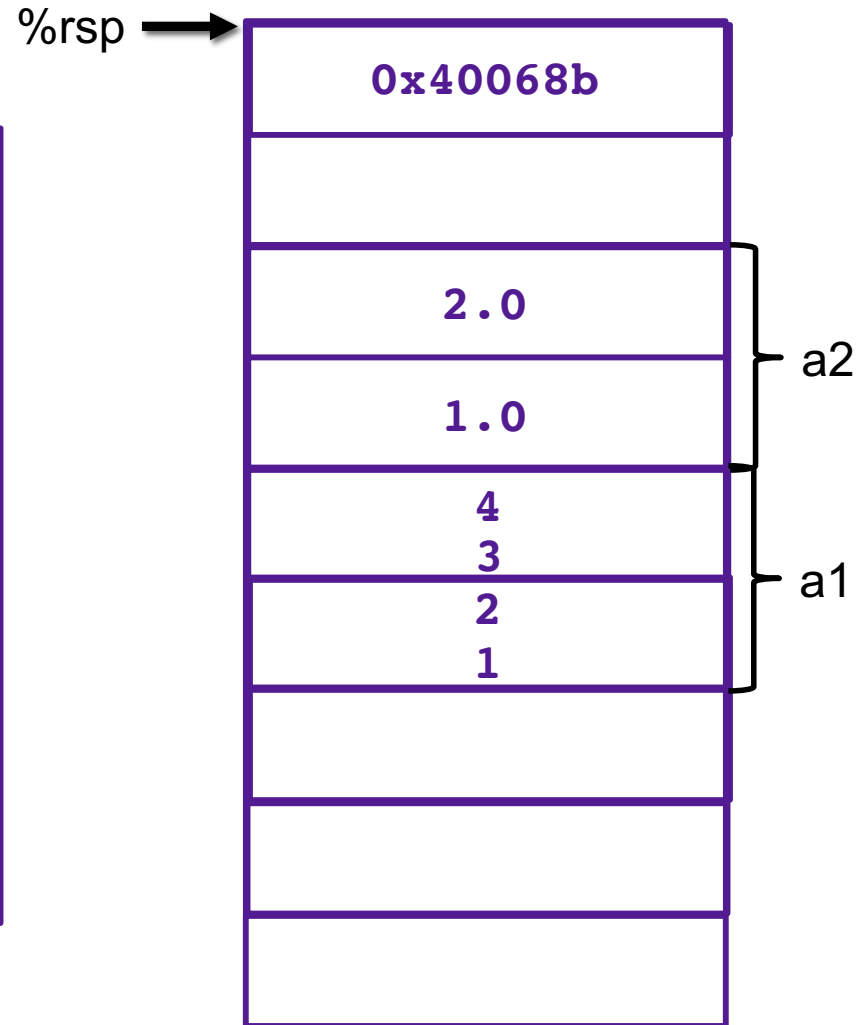
ARPANET Geographic Map, 31 October 1988



# Review: Function Calls in Assembly

```
void f1() {  
    double a2[2] = {1.0, 2.0};  
    int a1[4] = {1, 2, 3, 4};  
}
```

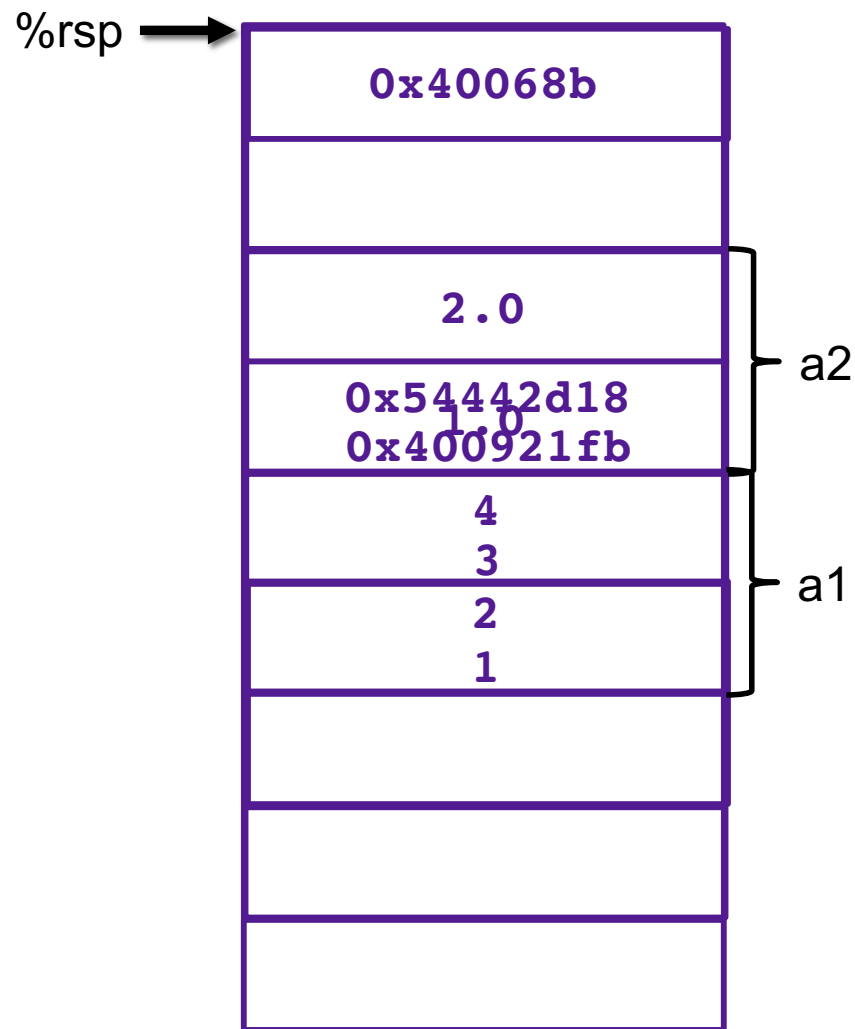
```
f1:  
    sub    $0x28,%rsp  
    movsd  0x216(%rip),%xmm0  
    movsd  %xmm0,0x10(%rsp)  
    movsd  0x210(%rip),%xmm0  
    movsd  %xmm0,0x18(%rsp)  
    movl   $0x1,(%rsp)  
    movl   $0x2,0x4(%rsp)  
    movl   $0x3,0x8(%rsp)  
    movl   $0x4,0xc(%rsp)  
    add    $0x28,%rsp  
    retq  
main:  
    call  f1  
    retq
```



# Memory Referencing Bug Example

```
void f1() {  
    double a2[2] = {1.0, 2.0};  
    int a1[4] = {1, 2, 3, 4};  
    a1[4] = 1413754136;  
    a1[5] = 1074340347;
```

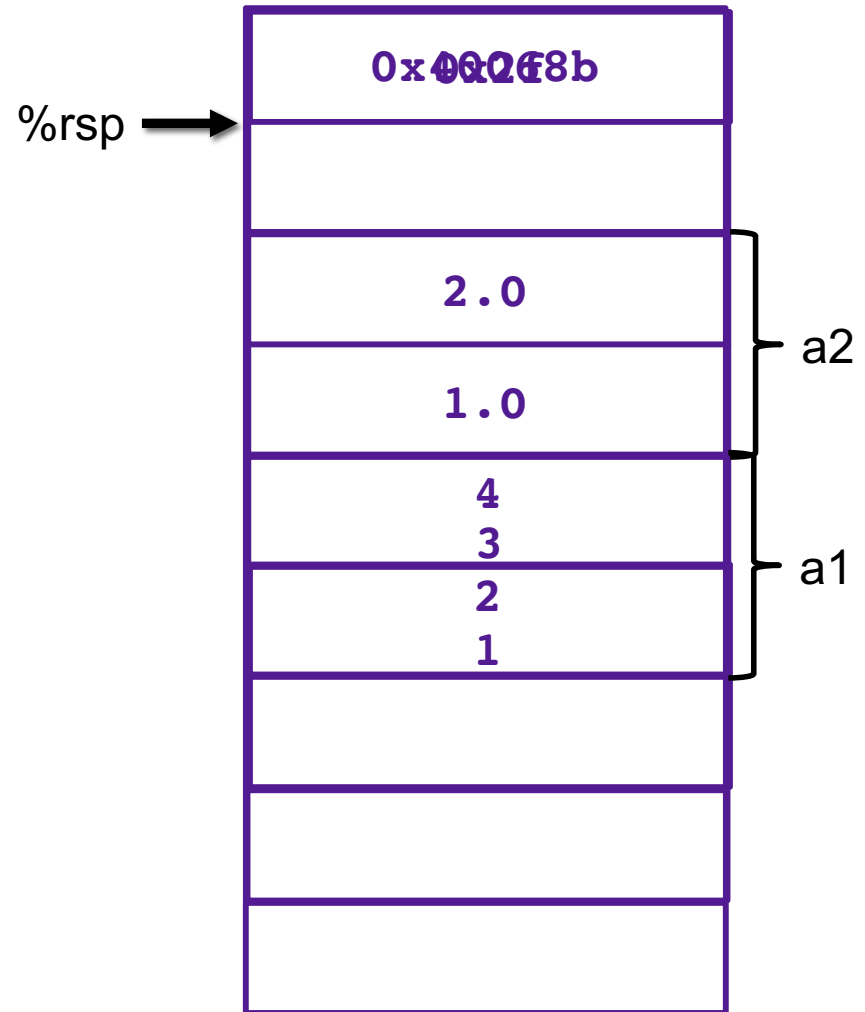
```
f1:  
    sub    $0x28,%rsp  
    movsd  0x216(%rip),%xmm0  
    movsd  %xmm0,0x10(%rsp)  
    movsd  0x210(%rip),%xmm0  
    movsd  %xmm0,0x18(%rsp)  
    movl   $0x1,(%rsp)  
    movl   $0x2,0x4(%rsp)  
    movl   $0x3,0x8(%rsp)  
    movl   $0x4,0xc(%rsp)  
    movl   $0x54442d18,0x10(%rsp)  
    movl   $0x400921fb,0x14(%rsp)  
    add    $0x28,%rsp  
    retq
```



# Memory Referencing Bug Example

```
void f1() {  
    double a2[2] = {1.0, 2.0};  
    int a1[4] = {1, 2, 3, 4};  
    a1[10] = 47;  
}
```

```
f1:  
    sub    $0x28, %rsp  
    movsd  0x216(%rip), %xmm0  
    movsd  %xmm0, 0x10(%rsp)  
    movsd  0x210(%rip), %xmm0  
    movsd  %xmm0, 0x18(%rsp)  
    movl   $0x1, (%rsp)  
    movl   $0x2, 0x4(%rsp)  
    movl   $0x3, 0x8(%rsp)  
    movl   $0x4, 0xc(%rsp)  
    movl   $0x2f, 0x28(%rsp)  
    add    $0x28, %rsp  
    retq
```



# Exercise 1: Memory Bugs

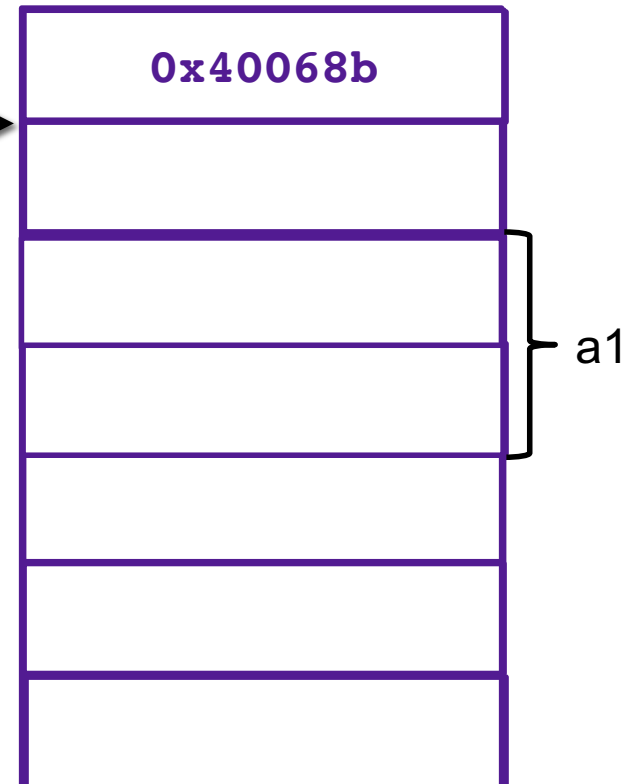
- What is the state of the stack immediately before the program returns from f2?
- What will happen immediately after f2 returns?

```
int f2(){  
    int a1[4] = {1,2,3,4};  
    a1[6] = 0x40247a;  
}
```

f2:

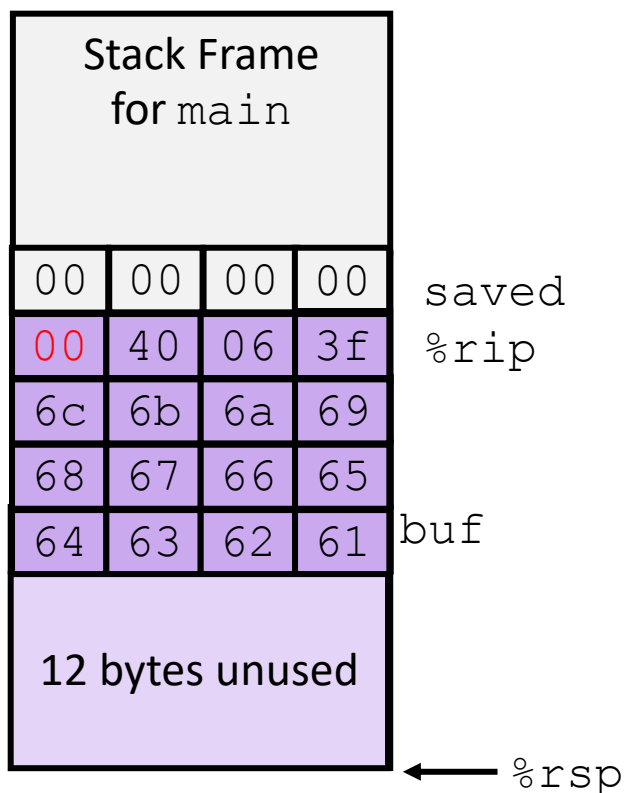
```
sub    $0x18,%rsp  
movl   $0x1, (%rsp)  
movl   $0x2, 0x4(%rsp)  
movl   $0x3, 0x8(%rsp)  
movl   $0x4, 0xc(%rsp)  
movl   $0x40247a, 0x18(%rsp)  
add    $0x18,%rsp  
retq
```

%rsp →



# Buffer Overflows

- Most common form of memory reference bug
  - Unchecked lengths on string inputs
  - Particularly for bounded character arrays on the stack



```
/* Echo Line */  
void echo() {  
    char buf[4];  
    gets(buf);  
    puts(buf);  
}
```

```
echo:  
    subq    $0x18, %rsp  
    lea    0xc(%rsp), %rdi  
    call   gets  
    lea    0xc(%rsp), %rdi  
    call   puts  
    addq   $0x18, %rsp  
    ret
```

# Exercise 2: Buffer Overflow

Construct an exploit string that will successfully cause the program to print "You are now logged in" without knowing the correct password

1. How many bytes of padding are in this exploit string?
2. What value will you overwrite the return address with?

```
int authenticate(char *password){
    char buf[4];
    gets(buf);
    int correct = !strcmp(password, buf);
    return correct;
}

int main(int argc, char ** argv){
    char * pw = "123456";
    printf("Enter your password: ");
    while(!authenticate(pw)){
        printf("Incorrect. Try again: ");
    }
    printf("You are now logged in\n");
    return 0;
}
```



# Exercise 2: Buffer Overflow

Construct an exploit string that will cause the program to print "You are now logged in" when the correct password is entered.

1. How many bytes of padding are needed to overflow the buffer?
2. What value will you overwrite with the password?

```
int authenticate(char *password){
    char buf[4];
    gets(buf);
    int correct = !strcmp(password, buf);
    return correct;
}

int main(int argc, char ** argv){
    char * pw = "123456";
    printf("Enter your password: ");
    while(!authenticate(pw)){
        printf("Incorrect. Try again: ");
    }
    printf("You are now logged in\n");
    return 0;
}
```

```
authenticate:
0x4005f6 <+0>: sub    $0x28,%rsp
0x4005fa <+4>: mov    %rdi,0x8(%rsp)
0x4005ff <+9>: lea   0x18(%rsp),%rax
0x400604 <+14>: mov   %rax,%rdi
0x400607 <+17>: mov   $0x0,%eax
0x40060c <+22>: callq 0x4004e0 <gets@plt>
0x400611 <+27>: lea   0x18(%rsp),%rdx
0x400616 <+32>: mov   0x8(%rsp),%rax
0x40061b <+37>: mov   %rdx,%rsi
0x40061e <+40>: mov   %rax,%rdi
0x400621 <+43>: callq 0x4004d0 <strcmp@plt>
0x400626 <+48>: test  %eax,%eax
0x400628 <+50>: sete  %al
0x40062b <+53>: movzbl %al,%eax
0x400636 <+64>: add   $0x28,%rsp
0x40063a <+68>: retq

main:
0x40063b <+0>: sub    $0x28,%rsp
0x40063f <+4>: mov    %edi,0xc(%rsp)
0x400643 <+8>: mov    %rsi,(%rsp)
0x400647 <+12>: movq   $0x400728,0x18(%rsp)
0x400650 <+21>: mov    $0x40072f,%edi
0x400655 <+26>: mov    $0x0,%eax
0x40065a <+31>: callq 0x4004b0 <printf@plt>
0x40065f <+36>: jmp    0x400670 <main+53>
0x400661 <+38>: mov    $0x400748,%edi
0x400666 <+43>: mov    $0x0,%eax
0x40066b <+48>: callq 0x4004b0 <printf@plt>
0x400670 <+53>: mov    0x18(%rsp),%rax
0x400675 <+58>: mov    %rax,%rdi
0x400678 <+61>: callq 0x4005f6 <authenticate>
0x40067d <+66>: test  %eax,%eax
0x40067f <+68>: je     0x400661 <main+38>
0x400681 <+70>: mov    $0x400768,%edi
0x400686 <+75>: callq 0x4004a0 <puts@plt>
0x40068b <+80>: mov    $0x0,%eax
0x400690 <+85>: add   $0x28,%rsp
0x400694 <+89>: retq
```

# Exercise 2: Buffer O

authenticate:

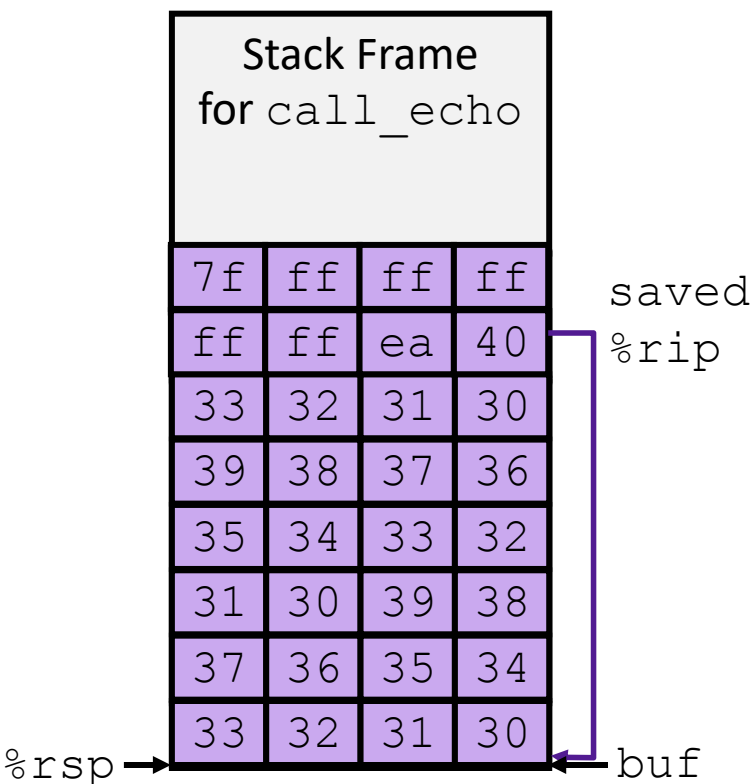
```
0x4005f6 <+0>: sub    $0x28,%rsp
0x4005fa <+4>: mov    %rdi,0x8(%rsp)
0x4005ff <+9>: lea   0x18(%rsp),%rax
0x400604 <+14>: mov   %rax,%rdi
0x400607 <+17>: mov   $0x0,%eax
0x40060c <+22>: callq 0x4004e0 <gets@plt>
0x400611 <+27>: lea   0x18(%rsp),%rdx
0x400616 <+32>: mov   0x8(%rsp),%rax
0x40061b <+37>: mov   %rdx,%rsi
0x40061e <+40>: mov   %rax,%rdi
0x400621 <+43>: callq 0x4004d0 <strcmp@plt>
0x400626 <+48>: test  %eax,%eax
0x400628 <+50>: sete  %al
0x40062b <+53>: movzbl %al,%eax
0x400636 <+64>: add   $0x28,%rsp
0x40063a <+68>: retq
```

main:

```
0x40063b <+0>: sub    $0x28,%rsp
0x40063f <+4>: mov    %edi,0xc(%rsp)
0x400643 <+8>: mov    %rsi,(%rsp)
0x400647 <+12>: movq   $0x400728,0x18(%rsp)
0x400650 <+21>: mov    $0x40072f,%edi
0x400655 <+26>: mov    $0x0,%eax
0x40065a <+31>: callq 0x4004b0 <printf@plt>
0x40065f <+36>: jmp    0x400670 <main+53>
0x400661 <+38>: mov    $0x400748,%edi
0x400666 <+43>: mov    $0x0,%eax
0x40066b <+48>: callq 0x4004b0 <printf@plt>
0x400670 <+53>: mov    0x18(%rsp),%rax
0x400675 <+58>: mov    %rax,%rdi
0x400678 <+61>: callq 0x4005f6 <authenticate>
0x40067d <+66>: test  %eax,%eax
0x40067f <+68>: je     0x400661 <main+38>
0x400681 <+70>: mov    $0x400768,%edi
0x400686 <+75>: callq 0x4004a0 <puts@plt>
0x40068b <+80>: mov    $0x0,%eax
0x400690 <+85>: add   $0x28,%rsp
0x400694 <+89>: retq
```

# Stack Smashing

- Idea: fill the buffer with bytes that will be interpreted as code
- Overwrite the return address with address of the beginning of the buffer



```
/* Echo Line */  
void echo()  
{  
    char buf[4];  
    gets(buf);  
    puts(buf);  
}
```

```
echo:  
    subq $18, %rsp  
    movq %rsp, %rdi  
    call gets  
    call puts  
    addq $18, %rsp  
    ret
```