COMPSCI 390R

Return Oriented Programming (ROP)

UMASS CYBERSEC CLUB

Topics to Cover:

1. Project 2 due 3/21

UMASS CYBERSEC CLUB

Recap of what we've learned so far:

1. Common C Vulnerabilities
    a) Integer Overflows
    b) Type Conversions

2. Redirecting Execution with Buffer Overflows

3. Creating Shellcode
    a) Spawning a Shell
    b) File IO
    c) Reverse Shells

UMASS CYBERSEC CLUB

Recap of what we've learned so far:

We can now exploit buffer overflows

- Execute shellcode given a pointer to a buffer
- Redirect execution to other locations in the function

13% of all vulnerabilities in the Common Vulnerabilities and Exposures database are buffer overflows

Almost 2000 buffer overflows reported in 2021 alone

There’s some problems though

Today's Program:

```c
#include <stdio.h>

int main() {
        char buf[8];
        fgets(buf, 100, stdin);
}
```

(Note this is statically compiled so fget's code is in the program)

UMASS CYBERSEC CLUB

Corrupting the Stack:

```c
#include <stdio.h>

int main() {
        char buf[8];
        fgets(buf, 100, stdin);
}
```

We have a buffer overflow in this program

Can redirect execution anywhere in memory, but to where?

No pointer to buffer, and it's too short to write shellcode anyways

UMASS CYBERSEC CLUB

Looking Closer:

```
endbr64
push   rbp
mov    rbp,rsp
sub    rsp,0x10
mov    rdx,QWORD PTR [rip+0xc1f30]

lea    rax,[rbp-0x8]
mov    esi,0x64
mov    rdi,rax
call   0x40bee0 <fgets>
mov    eax,0x0
leave
ret
```

1. Setting up the stack frame

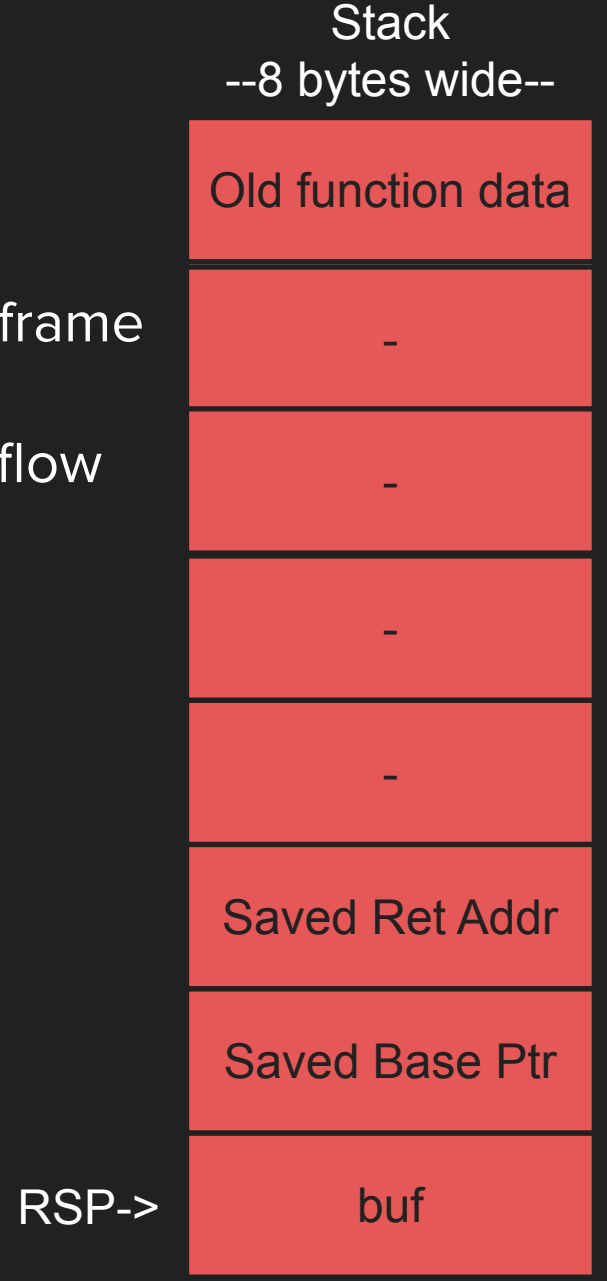
UMASS CYBERSEC CLUB

Looking Closer:

```
endbr64
push   rbp
mov    rbp,rsp
sub    rsp,0x10
mov    rdx,QWORD PTR [rip+0xc1f30]

lea    rax,[rbp-0x8]
mov    esi,0x64
mov    rdi,rax
call   0x40bee0 <fgets>
mov    eax,0x0
leave
ret
```

1. Setting up the stack frame
2. Trigger a buffer overflow



UMASS CYBERSEC CLUB

Looking Closer:

```
endbr64
push   rbp
mov    rbp,rsp
sub    rsp,0x10
mov    rdx,QWORD PTR [rip+0xc1f30]

lea    rax,[rbp-0x8]
mov    esi,0x64
mov    rdi,rax
call   0x40bee0 <fgets>
mov    eax,0x0
leave
ret
```

1. Setting up the stack frame
2. Trigger a buffer overflow
3. Use leave opcode
    a. Add back to rsp
    b. Pop base pointer

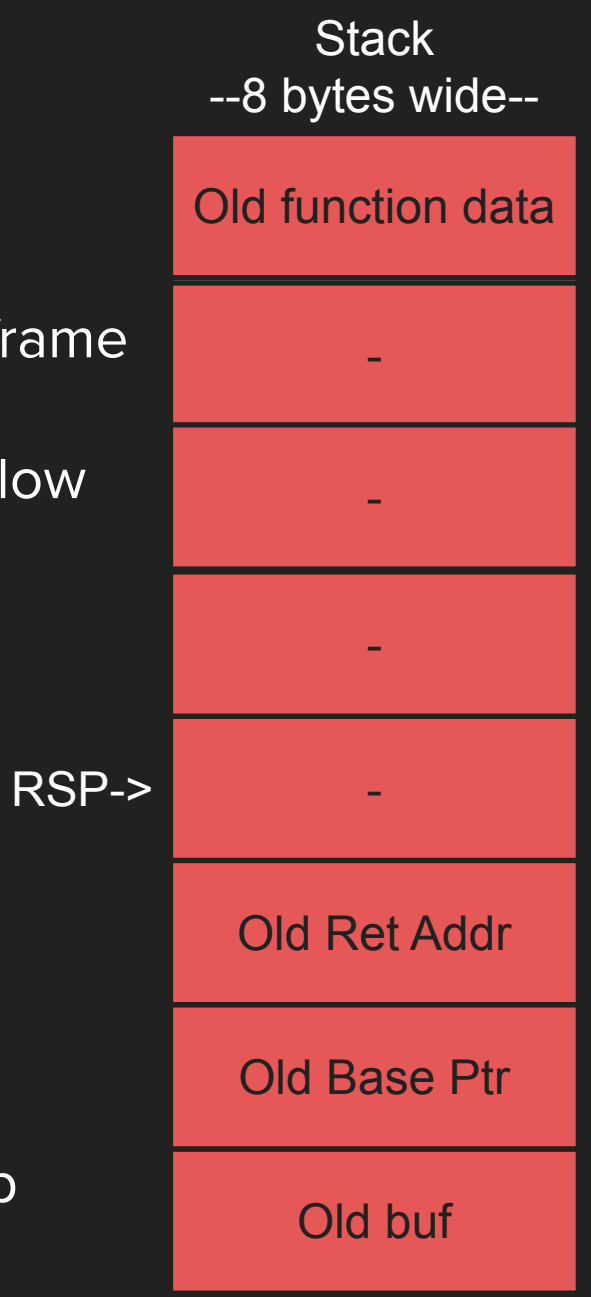
UMASS CYBERSEC CLUB

Looking Closer:

```
endbr64
push   rbp
mov    rbp,rsp
sub    rsp,0x10
mov    rdx,QWORD PTR [rip+0xc1f30]

lea    rax,[rbp-0x8]
mov    esi,0x64
mov    rdi,rax
call   0x40bee0 <fgets>
mov    eax,0x0
leave
ret
```

1. Setting up the stack frame
2. Trigger a buffer overflow
3. Use leave opcode
    a. Add back to rsp
    b. Pop base pointer
4. Use ret opcode
    a. pop ret addr to rip
    b. rsp moves again...

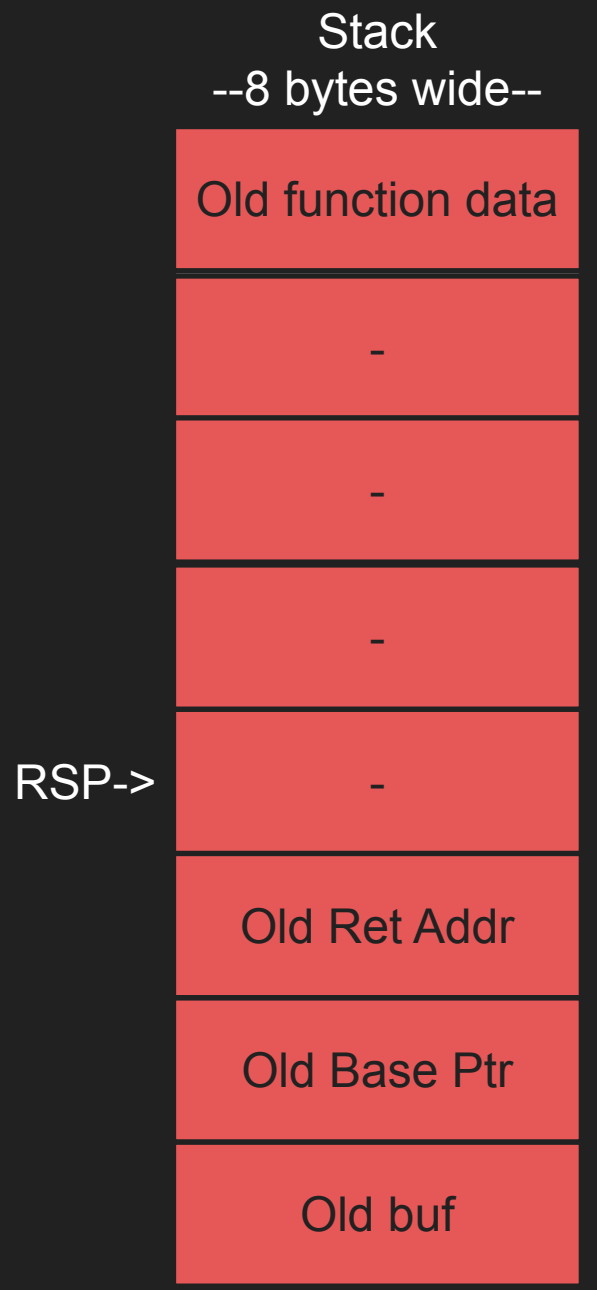


UMASS CYBERSEC CLUB

Where are we now?

- Can control the return address, base pointer and buffer
  - Lets us enter any function that we want elsewhere in the program
  - Lets us write our own code if we have a pointer to the buffer in memory

- Still have control of the stack memory and RSP
  - Can store data there for later use
    - If gets is used we can write unlimited bytes (kinda)
  - We can redirect to opcodes that work with the stack
    - push/pop
    - enter/leave
    - ret



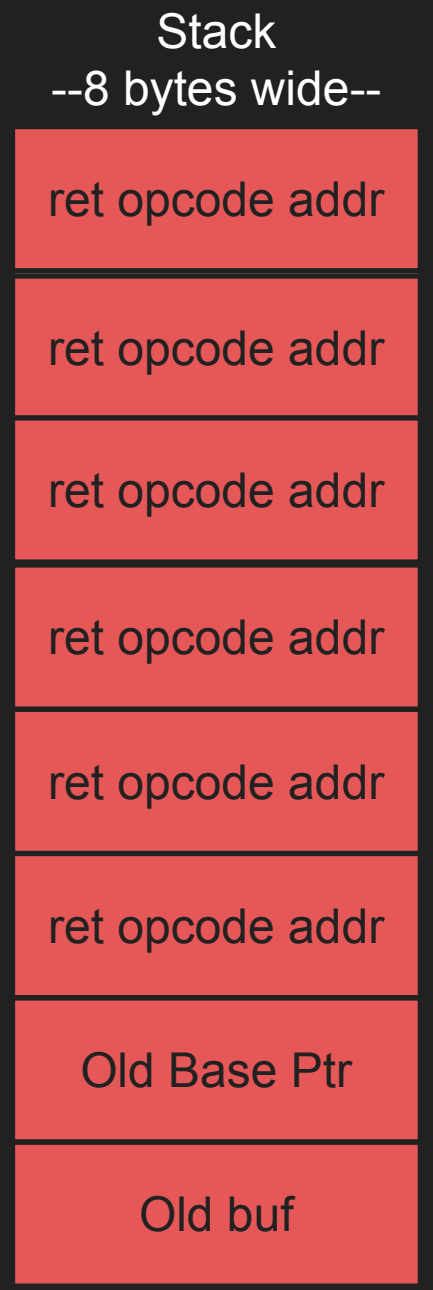
UMASS CYBERSEC CLUB

Any ideas of what to do?

Lets play a game:

1. Overwrite the buffer like last time
2. Overwrite the return address with the address of a ret opcode
3. Write the same return address over and over in the stack
4. Loop calling ret and continuously popping the same address

Since the ret opcode takes the address at the top of the stack, we can keep jumping to the same ret opcode over and over (or different ret opcodes)



UMASS CYBERSEC CLUB

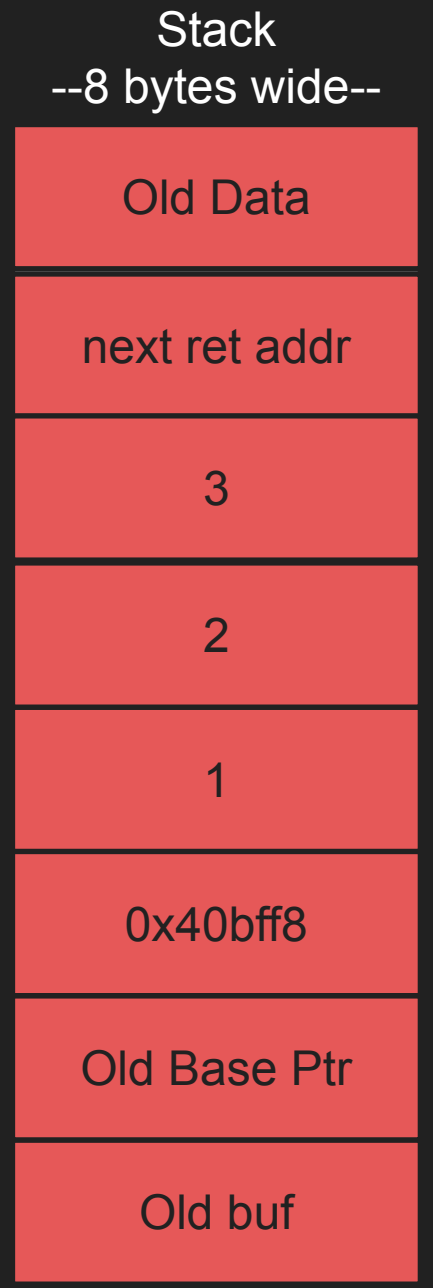
Taking this further:

Why are we just returning to a ret opcode?

Example Code:

```
0x000000000040bff8 <+280>:    pop    r12
0x000000000040bffa <+282>:    pop    r13
0x000000000040bffc <+284>:    pop    r14
0x000000000040bffe <+286>:    ret
```

If we jump to 0x40bff8, we can call the pop opcodes and ret again



UMASS CYBERSEC CLUB

Taking this further:

Write - Where - What gadgets

Example Code:

```
0x44b005 mov qword ptr [rsi], rax ; ret
0x409dee pop rsi ; ret
0x449417 pop rax ; ret
0x43dbd0 xor rax, rax ; ret
```

Using these sets of gadgets, we can control rsi and rax. This allows us

to put any value (rax) into any location (rsi), hence write what where

Return Oriented Programming

1. Find a list of useful “gadgets” (mov rax, 0; ret)
2. Write our shellcode using these gadgets
3. Win???

How to find gadgets though?

UMASS CYBERSEC CLUB

ROPGadget

ROPGadget is a useful tool that lets you automate finding gadgets and writing shellcode.

We will be using this in class!

```
Unique gadgets found: 39976

ROP chain generation
============================================================

- Step 1 -- Write-what-where gadgets

        [+] Gadget found: 0x44b005 mov qword ptr [rsi], rax ; ret
        [+] Gadget found: 0x409dee pop rsi ; ret
        [+] Gadget found: 0x449417 pop rax ; ret
        [+] Gadget found: 0x43dbd0 xor rax, rax ; ret

- Step 2 -- Init syscall number gadgets

        [+] Gadget found: 0x43dbd0 xor rax, rax ; ret
        [+] Gadget found: 0x470bf0 add rax, 1 ; ret
        [+] Gadget found: 0x470bf1 add eax, 1 ; ret

- Step 3 -- Init syscall arguments gadgets

        [+] Gadget found: 0x401894 pop rdi ; ret
        [+] Gadget found: 0x409dee pop rsi ; ret
        [+] Gadget found: 0x47e57b pop rdx ; pop rbx ; ret

- Step 4 -- Syscall gadget

        [+] Gadget found: 0x401fd4 syscall

- Step 5 -- Build the ROP chain

        #!/usr/bin/env python2
        # execve generated by ROPgadget

        from struct import pack

        # Padding goes here
        p = ''
```

UMASS CYBERSEC CLUB

Experiment Time:

Take the binary from moodle, install ropgadget from https://github.com/JonathanSalwan/ROPgadget

Download the C code from moodle and compile with the flags at the top of the file

Try writing your own ropchains, including controlling variables, calling syscalls and writing to memory.

```
Unique gadgets found: 39976

ROP chain generation
===========================================================

- Step 1 -- Write-what-where gadgets

        [+] Gadget found: 0x44b005 mov qword ptr [rsi], rax ; ret
        [+] Gadget found: 0x409dee pop rsi ; ret
        [+] Gadget found: 0x449417 pop rax ; ret
        [+] Gadget found: 0x43dbd0 xor rax, rax ; ret

- Step 2 -- Init syscall number gadgets

        [+] Gadget found: 0x43dbd0 xor rax, rax ; ret
        [+] Gadget found: 0x470bf0 add rax, 1 ; ret
        [+] Gadget found: 0x470bf1 add eax, 1 ; ret

- Step 3 -- Init syscall arguments gadgets

        [+] Gadget found: 0x401894 pop rdi ; ret
        [+] Gadget found: 0x409dee pop rsi ; ret
        [+] Gadget found: 0x47e57b pop rdx ; pop rbx ; ret

- Step 4 -- Syscall gadget

        [+] Gadget found: 0x401fd4 syscall

- Step 5 -- Build the ROP chain

        #!/usr/bin/env python2
        # execve generated by ROPgadget

        from struct import pack

        # Padding goes here
        p = ''
```

UMASS CYBERSEC CLUB