Class Exercise—ROP and CFI
ENEE 457
9/15/21

1. In this exercise, we will take a closer look at how to launch a Return Oriented Programming (ROP) attack that starts a shell. To do this, we need to implement the assembly instructions corresponding to the shellcode. Actually, we will only have time to implement a few of these instructions. Specifically, the first steps of launching the shell are to (1) zero out the %eax register (2) load string "/bin" (0x6e69622f in hex) somewhere in memory---say into memory address 0x404 (3) load string "//sh" (0x68732f2f in hex) to the consecutive location in memory.

We are given the following gadgets.

```
0x9b4550: mov [%ebx] %ecx
          ret
```

```
0x9c27e0: pop %ecx
          ret
```

```
0x9b8d20: xor %eax %eax
          ret
```

```
0x9c61b0: pop %ebx
          ret
```

Assume that we are currently executing a function func(). What should the stack look like right before "ret" is called at the end of the execution of func() in order for steps (1) (2) and (3) to be completed upon return from func()?

Class Exercise—ROP and CFI
ENEE 457
9/15/21

2. Consider the following functions, which we divide into blocks as in the picture below. Each block can be identified by the number to its left.

```
int e() {
1   return 3;
}
```

```
void f(int i) {
2   e();
3   g(--i);
4   h(--i);
}
```

```
void g(int i) {
5   f(--i);
6   h(--i);
}
```

```
void h(int i) {
7   f(--i);
8   g(--i);
}
```

Draw the corresponding Control Flow Graph (CFG) for these functions. Each node of the CFG will correspond to one of the blocks above. Number the nodes of the CFG according to the number assigned to the corresponding block.

Assume we would like to implement inline monitoring in order to enforce control flow integrity. Assume the possible labels are {A,B,C,D,E,F,G,H}. How should each node in the CFG be labeled in a "detailed" labeling of the graph?