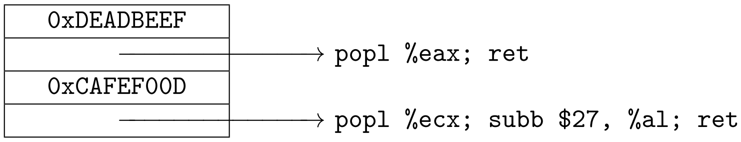
CS 487: Secure Computer Systems **Fall 2017**

Return-oriented programming worksheet

Recall that a (traditional) return-oriented program is structured as a sequence of addresses of code and data on the stack. Each address (represented here by an arrow to code) points to a sequence of code ending in `ret`. For example, the following fragment of a return-oriented program loads `0xCAFEF00D` into register `ecx`, subtracts 27 from register `al`, and then loads `0xDEADBEEF` into register `eax`.



```
0xDEADBEEF
     ──────────→ popl %eax; ret
0xCAFEF00D
     ──────────→ popl %ecx; subb $27, %al; ret
```

Note that the ordering was important due to the unwanted subtraction.

Useful instruction sequences

We're going to use these instruction sequences (and only these) to construct the gadgets on the next sheet.

①
```
popl %esi
ret
```

②
```
popl %ebx
popl %ebp
ret
```

③
```
addl %ecx, %eax
ret
```

④
```
sub %ebx, %eax
ret
```

⑤
```
imul %eax, %ebx
ret
```

⑥
```
xorl %eax, %eax
ret
```

⑦
```
andl -16(%ebp), %ebx
ret
```

⑧
```
orl %esi, %eax
ret
```

⑨
```
movl %ebx, %ecx
ret
```

⑩
```
movl %ecx, 32(%eax)
ret
```

⑪
```
movl (%eax), %ecx
ret
```

Gadgets

Let X, Y, and Z be constant addresses each pointing at 4 bytes of memory. We're going to treat X, Y, and Z like the addresses of global variables `x`, `y`, and `z` in C. Construct the following gadgets by filling in the empty stack diagrams with circled numbers representing the addresses of the corresponding useful instruction sequences and data like 42, X or $Y - 32$. Start at the bottom of the stack diagram and move up. There are larger diagrams on the back of both pages.

1. Load immediate gadget. Set X to be the four byte constant c. (In C, `x = c;`)
2. Move gadget. Copy four bytes from X to Y. (In C, `y = x;`)
3. Load gadget. Treat the four bytes at X as a pointer; load four bytes from it and store in Y. (In C, `y = *x;`)
4. Add gadget (tricky!). Load ints from X and Y, add them, and store in Z. (In C, `z = x + y;`)
5. Store gadget (tricky!). Treat the four bytes at Y as a pointer; load four bytes from X and store at the address pointed to by the pointer. (In C, `*y = x;`)