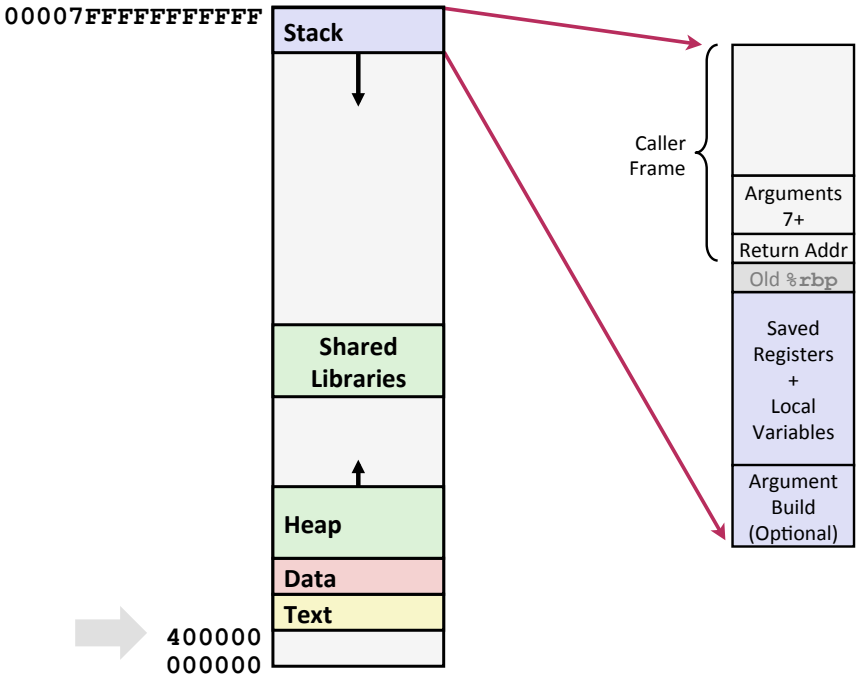
x86-64 Linux Memory Layout



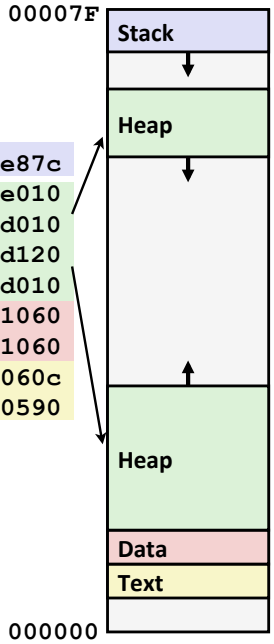
Memory Layout Example

```
char big_array[1L<<24];
char huge_array[1L<<31];

int foo() { return 0; }

int main() {
    void *p1, *p2, *p3, *p4;
    int local = 0;
    p1 = malloc(1L << 28);
    p2 = malloc(1L << 8);
    p3 = malloc(1L << 32);
    p4 = malloc(1L << 8);
 /* Some print statements ... */
}
```

```
      local 0x00007ffe4d3be87c
         p1 0x00007f7262a1e010
         p3 0x00007f7162a1d010
         p4 0x000000008359d120
         p2 0x000000008359d010
  big_array 0x0000000080601060
 huge_array 0x0000000000601060
      main() 0x000000000040060c
       foo() 0x0000000000400590
```



String Library Code

```
/* Get string from stdin */
char* gets(char* dest) {
    int c = getchar();
    char* p = dest;
    while (c != EOF && c != '\n') {
        *p++ = c;
        c = getchar();
    }
    *p = '\0';
    return dest;
}
```

```
strcpy(char* dest, char* src) // copy
strcat(char* dest, char* src) // concatenate
// see also: scanf, fscanf, sscanf, ...
```


Vulnerable Buffer Code

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

```
void call_echo() {
    echo();
}
```

```
unix>./buftest
Type a string:0123456789012345678901234
Segmentation Fault
```

Buffer Overflow Assembly

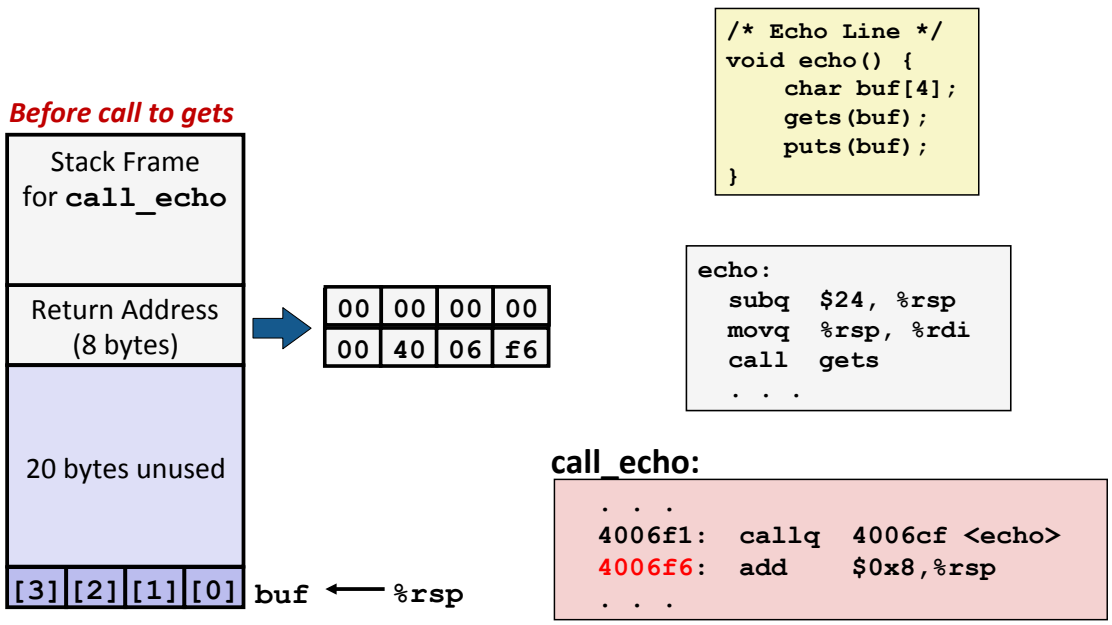
echo:

```
00000000004006cf <echo>:
 4006cf:  48 83 ec 18           sub    $0x18,%rsp
 4006d3:  48 89 e7              mov    %rsp,%rdi
 4006d6:  e8 a5 ff ff ff        callq  400680 <gets>
 4006db:  48 89 e7              mov    %rsp,%rdi
 4006de:  e8 3d fe ff ff        callq  400520 <puts@plt>
 4006e3:  48 83 c4 18           add    $0x18,%rsp
 4006e7:  c3                    retq
```

call_echo:

```
4006e8:   48 83 ec 08           sub    $0x8,%rsp
 4006ec:  b8 00 00 00 00        mov    $0x0,%eax
 4006f1:  e8 d9 ff ff ff        callq  4006cf <echo>
 4006f6:  48 83 c4 08           add    $0x8,%rsp
 4006fa:  c3                    retq
```


Buffer Overflow Stack



Buffer Overflow Examples

After call to gets

Stack Frame for `call_echo`

00	00	00	00
00	40	06	f6
00	32	31	30
39	38	37	36
35	34	33	32
31	30	39	38
37	36	35	34
33	32	31	30

`buf` ← `%rsp`

```
unix>./buftest
Type a string:0123456789012345678901
0123456789012345678901
```

Overflowed, but did not corrupt state

After call to gets

Stack Frame for `call_echo`

00	00	00	00
00	40	00	34
33	32	31	30
39	38	37	36
35	34	33	32
31	30	39	38
37	36	35	34
33	32	31	30

`buf` ← `%rsp`

```
unix>./buftest
Type a string:012345678901234567890123
Segmentation Fault
```

Overflowed and corrupted return pointer

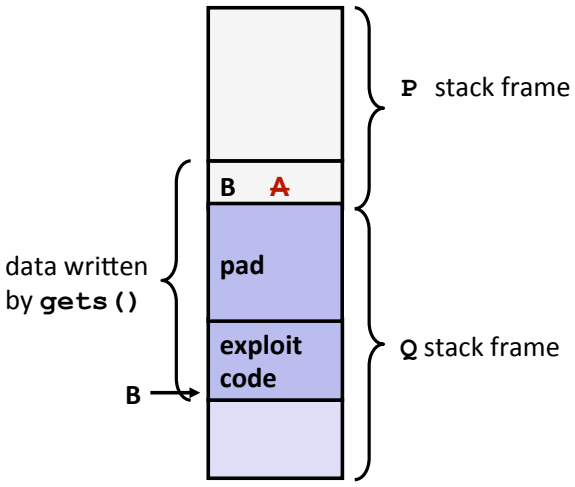
Code Injection Attacks

```
void P(){
  Q();
  ...
}
```

return address A

```
int Q() {
  char buf[64];
  gets(buf);
  ...
  return ...;
}
```

Stack after call to `gets()`

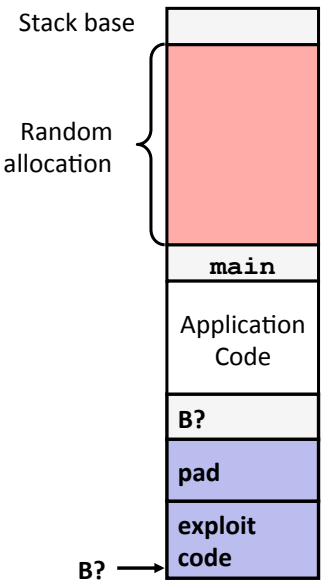


Write Secure Code (!)

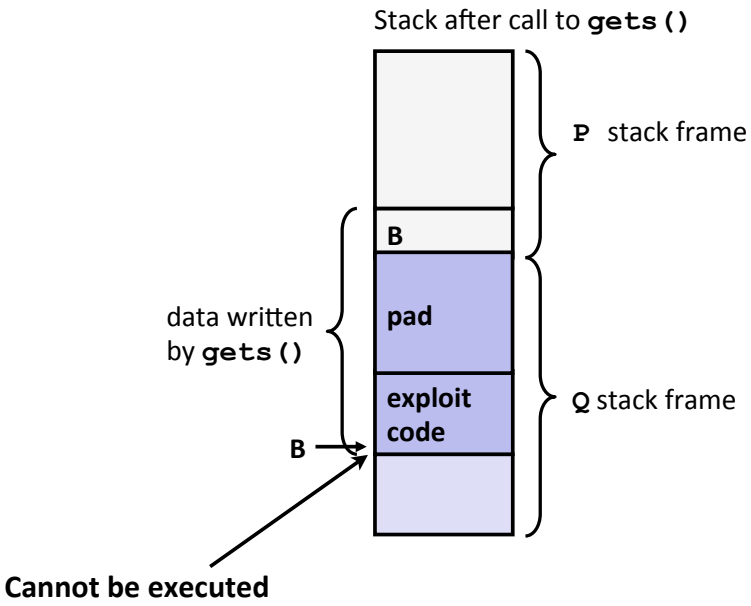
- Example: length-limited string routines
  - `fgets`, `strncpy`, ...
- No %s in `scanf`

```
/* Echo Line */
void echo() {
    char buf[4];  /* Way too small! */
    fgets(buf, 4, stdin);
    puts(buf);
}
```


Stack Randomization (ASLR)

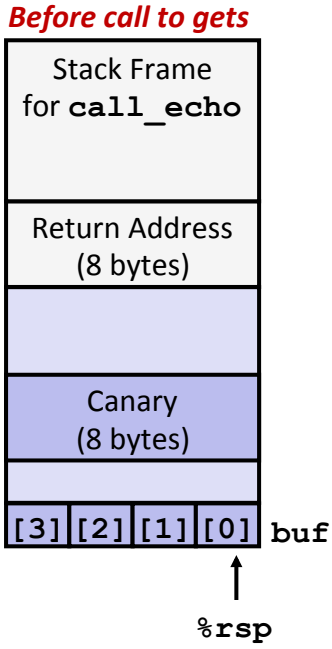


Nonexecutable Memory Segments



Stack Canaries

Stack Canary Example

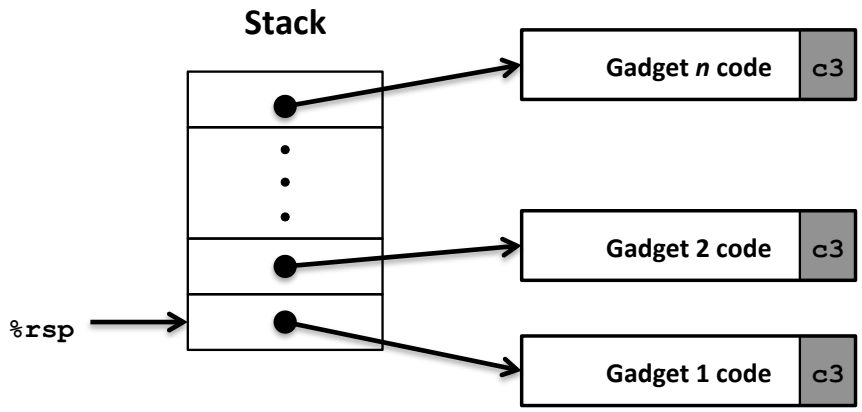



```
/* Echo Line */
void echo() {
    char buf[4];  /* Way too small! */
    gets(buf);
    puts(buf);
}
```

echo:

```
40072f:  sub    $0x18,%rsp
400733:  mov    %fs:0x28,%rax   # Get canary
40073c:  mov    %rax,0x8(%rsp)  # Put on stack
400741:  xor    %eax,%eax
400743:  mov    %rsp,%rdi
400746:  callq  4006e0 <gets>
40074b:  mov    %rsp,%rdi
40074e:  callq  400570 <puts@plt>
400753:  mov    0x8(%rsp),%rax  # Retrieve canary
400758:  xor    %fs:0x28,%rax   # Check canary
400761:  je     400768 <echo+0x39>
400763:  callq  400580 <__stack_chk_fail@plt>
400768:  add    $0x18,%rsp
40076c:  retq
```


Return-Oriented Programming



Gadget Example: Function Ends

```
long ab_plus_c(long a, long b, long c) {
   return a * b + c;
}
```

```
00000000004004d0 <ab_plus_c>:
  4004d0:  48 0f af fe  imul %rsi,%rdi
  4004d4:  48 8d 04 17  lea (%rdi,%rdx,1),%rax
  4004d8:  c3           retq
```

rax ← rdi + rdx

Gadget address = 0x4004d4

Gadget Example 2: Repurposing

```
void setval(unsigned *p) {
    *p = 3347663060u;
}
```

Encodes `movq %rax, %rdi`

```
<setval>:
  4004d9:  c7 07 d4 48 89 c7  movl  $0xc78948d4,(%rdi)
  4004df:  c3                 retq
```

rdi ← rax

Gadget address = 0x4004dc