x86-64 Procedures
Topics

- ABIs, the runtime stack, and control transfer
- Data transfer and local storage
- Security issues
• A **procedure** is a portion of code packaged for re-use
  - Key abstraction in software development
  - Provide **modularity** and **encapsulation**
  - Many alternative names: functions, methods, subroutines

• Well-designed procedures have:
  - Well-documented, typed arguments and return value(s)
  - Clear impact on program state (or no impact)
    • Also known as “side effects”
Problem

- Impossible to implement procedures in assembly with branches or jumps alone
  - Once you’ve jumped, how do you return?
  - Can hard-code for one call site, but not for 2+
- Need a mechanism for “remembering” where we came from
  - And any machine state important for getting back
  - Don’t want to use registers because there are so few
  - Solution: use memory! (but how and where?)
• Application Binary Interface (ABI)
  – Interface between program & system at the binary level
  – Includes rules about how procedure calls are implemented
  – These rules are referred to as calling conventions
  – We will study the standard x86-64 calling conventions

• Calling conventions specify:
  – Control transfer
  – Data transfer
  – Local storage
Runtime stack

- Basic idea: maintain a system stack frame for each procedure call
  - All active procedure have a frame
  - Each frame stores information about a single active call
    - Arguments, local variables, return address
  - GDB's "backtrace" command follows the chain up
  - Recursion just works!

Here function P has called function Q
Control transfer

- Use stack to store return addresses
  - **Return address**: the instruction AFTER the call
  - `call / callq` pushes 64-bit return address onto stack and sets `%rip`
  - `ret / retq` pops the return address and sets `%rip`

```
400550 <main>:
  ...  
400563  callq 400540 <foo>  
400568  movq 0x8(%rsp), %rdx  
  ...  
```

```
400540 <foo>:
  ...  
400540  xorq %rax, %rax  
40054d  retq  
```

(a) Executing call  
(b) After call  
(c) After ret
Data transfer

• In x86-64, up to six integral (integer or pointer) arguments are passed to a procedure via these registers (in order):
  – %rdi, %rsi, %rdx, %rcx, %r8, %r9
  – Other arguments are passed on the stack (and pushed in reverse order)

• A single return value is passed back via %rax
  – Large structs often “returned” using a pointer
Local storage (registers)

- Some registers are designated **callee-saved**
  - In x86-64: `%rbx, %rbp, %r12, %r13, %r14, %r15`
  - A procedure must save/restore these registers (often using push/pop) if they are used during the procedure
  - When possible, avoid using these registers inside procedures (lower overhead)

- Other registers (except `%rsp`) are **caller-saved**
  - Caller must save them if they need to be preserved
  - The stack pointer is a special case (used for communication)
Local storage (memory)

- Procedures can allocate space on the stack for **local variables**
  - Subtract # of bytes needed from %rsp
  - Deallocate by restoring old %rsp value

- Variable-sized allocations require special handling
  - Use **base / frame pointer** (%rbp) to track “anchor” for current frame
  - Save previous base pointer on stack at beginning of function
  - Section 3.10.5 in textbook
Base pointers

• Use base pointer (%rbp) to track the beginning of current frame
  - Parameters at positive offsets
  - Local values at negative offsets
  - Chain of base pointers up the stack
  - Push/pop BP like return address

CALLER

Pre-call:
pushq <param2>
pushq <param1>
callq <func>

Prologue:
pushq %rbp
movq %rsp, %rbp
subq $n, %rsp

Epilogue:
movq %rbp, %rsp
popq %rbp
retq

Post-return:
addq $16, %rsp

CALLEE

void foo()
{
    int a, b;
    bar(a)
    return;
}

void bar(x)
{
    int c;
    baz(x, c);
    return;
}

void baz(x, y)
{
    int d;
    return;
}

void foo()
{}
void bar(x)
{}
void baz(x, y)
{}

CALLER CALLEE

CS 430/432 preview
What is the security problem with the following C function?

```c
void echo () {
    char buf[8];
    gets(buf);
    printf(buf);
}
```

- A) It reads from an unspecified file stream
- B) It writes to standard output
- C) It can write to memory in echo’s stack frame
- D) It can write to memory in the caller’s stack frame
- E) It stores a character array on the stack
Buffer overflows

• Major x86-64 security issue
  - C and assembly do not check for out-of-bounds array accesses
  - x86-64 stores return addresses and data on the same stack
  - Out-of-bound writes to local variables may overwrite other stack frames
  - Allows attackers to change control flow just by providing the right "data"
  - Many historical exploits (including Morris worm)

```c
void echo ()
{
    char buf[8];
    gets(buf);
    printf(buf);
}
```

DO NOT WRITE CODE LIKE THIS!
Buffer overflows

- **Shellcode (exploit code)**
  - Pre-compiled snippets of code that exploit a buffer overflow

```c
char shellcode[] =
    "\xeb\x1f\x5e\x89\x76\x08\x31\xc0\x88\x46\x07\x89\x46\x0c\xb0\x0b"
    "\x89\xf3\x8d\x4e\x08\x8d\x56\x0c\xcd\x80\x31\xdb\x89\xd8\x40\xcd"
    "\x80\xe8\xdc\xff\xff\xff/bin/sh";
```

Complication: Must pad the shellcode with address of the buffer (guess and/or use a NOP-sled)
Mitigating buffer overflows

- **Stack randomization**
  - Randomize starting location of stack
  - Makes it more difficult to guess buffer address
  - In Linux: address-space layout randomization

- **Corruption detection**
  - Insert a *canary* (guard value) on stack after each array
  - Check canary before returning from function

- **Read-only code regions**
  - Mark stack memory as "no-execute"
  - Hinders just-in-time compilation and instrumentation
Exercise

• Trace the following code--what is the value of \%rax at the end?
  – Initial values: \%rsp = 0xcf8d7fffffff, \%rip = 0x4004e8

4004d6 <leaf>:
  4004d6: 48 8d 7f 0f
  4004da: c3

        leaq  0xf(%rdi),%rdi
          retq

4004db <top>:
  4004db: 48 83 ef 05
  4004df: e8 f2 ff ff ff
  4004e4: 48 01 ff
  4004e7: c3

        subq  $0x5,%rdi
          callq  4004d6
          addq  %rdi,%rdi
          retq

4004e8 <main>:
  4004e8: 48 c7 c7 64 00 00 00
  4004ef: e8 e7 ff ff ff
  4004f4: 48 89 f8
  4004f7: c3

        movq  $100,%rdi
          callq  4004db
          movq  %rdi,%rax
          retq
Exercise

• Trace the following code--what is the value of %rax at the end?

4004d6 <leaf>:
  4004d6:  leaq  0xf(%rdi),%rdi
  4004da:  retq

4004db <top>:
  4004db:  subq  $0x5,%rdi
  4004df:  callq  4004d6
  4004e4:  addq  %rdi,%rdi
  4004e7:  retq

4004e8 <main>:
  4004e8:  movq  $100,%rdi
  4004ef:  callq  4004db
  4004f4:  movq  %rdi,%rax
  4004f7:  retq
Exercise

- Trace the following code—what is the value of %rax at the end?

```
4004d6 <leaf>:
  4004d6:  leaq  0xf(%rdi),%rdi
  4004da:  retq

4004db <top>:
  4004db:  subq $0x5,%rdi
  4004df:  callq 4004d6
  4004e4:  addq %rdi,%rdi
  4004e7:  retq

4004e8 <main>:
  4004e8:  movq $100,%rdi
  4004ef:  callq 4004db
  4004f4:  movq %rdi,%rax
  4004f7:  retq
```

```text
%rip  0x4004ef
%rsp  0x7fffffffde488
%rdi  100
%rax ???
%rip  0x7fffffffde488
%rsp  ???
%rdi  0x7fffffffde480
%rax ???
%rip  0x7fffffffde478
%rsp  ???
%rdi  0x7fffffffde470
%rax ???
```

...
Exercise

• Trace the following code--what is the value of %rax at the end?

```
4004d6 <leaf>:
    4004d6: leaq 0xf(%rdi),%rdi
    4004da: retq

4004db <top>:
    4004db: subq $0x5,%rdi
    4004df: callq 4004d6
    4004e4: addq %rdi,%rdi
    4004e7: retq

4004e8 <main>:
    4004e8: movq $100,%rdi
    4004ef: callq 4004db
    4004f4: movq %rdi,%rax
    4004f7: retq
```
Exercise

- Trace the following code--what is the value of %rax at the end?

```assembly
4004d6 <leaf>:
4004d6:   leaq  0xf(%rdi),%rdi
4004da:   retq

4004db <top>:
4004db:   subq  $0x5,%rdi
4004df:   callq 4004d6
4004e4:   addq  %rdi,%rdi
4004e7:   retq

4004e8 <main>:
4004e8:   movq  $100,%rdi
4004ef:   callq 4004db
4004f4:   movq  %rdi,%rax
4004f7:   retq
```

<table>
<thead>
<tr>
<th>%rip</th>
<th>0x4004df</th>
</tr>
</thead>
<tbody>
<tr>
<td>%rsp</td>
<td>0x7fffffff480</td>
</tr>
<tr>
<td>%rdi</td>
<td>95</td>
</tr>
<tr>
<td>%rax</td>
<td>???</td>
</tr>
</tbody>
</table>

...
Exercise

- Trace the following code--what is the value of %rax at the end?

```
4004d6 <leaf>:
    4004d6: leaq 0xf(%rdi),%rdi
    4004da: retq

4004db <top>:
    4004db: subq $0x5,%rdi
    4004df: callq 4004d6
    4004e4: addq %rdi,%rdi
    4004e7: retq

4004e8 <main>:
    4004e8: movq $100,%rdi
    4004ef: callq 4004db
    4004f4: movq %rdi,%rax
    4004f7: retq
```
Trace the following code—what is the value of %rax at the end?

4004d6 <leaf>:
   4004d6: leaq 0xf(%rdi),%rdi
   4004da: retq

4004db <top>:
   4004db: subq $0x5,%rdi
   4004df: callq 4004d6
   4004e4: addq %rdi,%rdi
   4004e7: retq

4004e8 <main>:
   4004e8: movq $100,%rdi
   4004ef: callq 4004db
   4004f4: movq %rdi,%rax
   4004f7: retq
Exercise

• Trace the following code--what is the value of %rax at the end?

```
4004d6 <leaf>:
    4004d6:  leaq  0xf(%rdi),%rdi
    4004da:  retq

4004db <top>:
    4004db:  subq  $0x5,%rdi
    4004df:  callq  4004d6
    4004e4:  addq  %rdi,%rdi
    4004e7:  retq

4004e8 <main>:
    4004e8:  movq  $100,%rdi
    4004ef:  callq  4004db
    4004f4:  movq  %rdi,%rax
    4004f7:  retq
```

<table>
<thead>
<tr>
<th>%rip</th>
<th>0x4004e4</th>
</tr>
</thead>
<tbody>
<tr>
<td>%rsp</td>
<td>0x7fffffffffe480</td>
</tr>
<tr>
<td>%rdi</td>
<td>110</td>
</tr>
<tr>
<td>%rax</td>
<td>???</td>
</tr>
</tbody>
</table>

```
Exercise

• Trace the following code--what is the value of %rax at the end?

```
4004d6 <leaf>:
  4004d6: leaq 0xf(%rdi),%rdi
  4004da: retq

4004db <top>:
  4004db: subq $0x5,%rdi
  4004df: callq 4004d6
  4004e4: addq %rdi,%rdi
  4004e7: retq

4004e8 <main>:
  4004e8: movq $100,%rdi
  4004ef: callq 4004db
  4004f4: movq %rdi,%rax
  4004f7: retq
```

<table>
<thead>
<tr>
<th>%rip</th>
<th>0x4004e7</th>
</tr>
</thead>
<tbody>
<tr>
<td>%rsp</td>
<td>0x7fffffffffe480</td>
</tr>
<tr>
<td>%rdi</td>
<td>220</td>
</tr>
<tr>
<td>%rax</td>
<td>???</td>
</tr>
</tbody>
</table>

...
Exercise

• Trace the following code--what is the value of %rax at the end?

4004d6 <leaf>:
  4004d6: leaq 0xf(%rdi),%rdi
  4004da: retq

4004db <top>:
  4004db: subq $0x5,%rdi
  4004df: callq 4004d6
  4004e4: addq %rdi,%rdi
  4004e7: retq

4004e8 <main>:
  4004e8: movq $100,%rdi
  4004ef: callq 4004db
  4004f4: movq %rdi,%rax
  4004f7: retq
Exercise

- Trace the following code--what is the value of %rax at the end?

```
4004d6 <leaf>:
    4004d6: leaq 0xf(%rdi),%rdi
    4004da: retq

4004db <top>:
    4004db: subq $0x5,%rdi
    4004df: callq 4004d6
    4004e4: addq %rdi,%rdi
    4004e7: retq

4004e8 <main>:
    4004e8: movq $100,%rdi
    4004ef: callq 4004db
    4004f4: movq %rdi,%rax
    4004f7: retq
```
Exercise

- Trace the following code—what is the value of %rax at the end?

<table>
<thead>
<tr>
<th>Address</th>
<th>Instruction</th>
<th>%rip</th>
<th>%rsp</th>
<th>%rdi</th>
<th>%rax</th>
</tr>
</thead>
<tbody>
<tr>
<td>4004d6</td>
<td><code>leaq 0xf(%rdi),%rdi</code></td>
<td>???</td>
<td>0xfffffffffffffffe490</td>
<td>220</td>
<td>220</td>
</tr>
<tr>
<td>4004da</td>
<td><code>retq</code></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>4004db</td>
<td><code>subq $0x5,%rdi</code></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>4004df</td>
<td><code>callq 4004d6</code></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>4004e4</td>
<td><code>addq %rdi,%rdi</code></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>4004e7</td>
<td><code>retq</code></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>4004f4</td>
<td><code>movq %rdi,%rax</code></td>
<td>0xfffffffffffffffe488</td>
<td>0xfffffffffffffffe480</td>
<td>0x4004f4</td>
<td>0x4004e4</td>
</tr>
<tr>
<td>4004f7</td>
<td><code>retq</code></td>
<td>0xfffffffffffffffe470</td>
<td>0xfffffffffffffffe478</td>
<td>0x4004e4</td>
<td>???</td>
</tr>
</tbody>
</table>

...
Exercise

- Trace the following code--what is the value of %rax at the end?

\[
\begin{array}{ll}
4004d6 \ <leaf> : & \\
\quad 4004d6 : \text{lea}q \quad 0xf(%rdi),%rdi \\
\quad 4004da : \text{ret}q \\
4004db \ <top> : & \\
\quad 4004db : \text{sub}q \quad $0x5,%rdi \\
\quad 4004df : \text{call}q \quad 4004d6 \\
\quad 4004e4 : \text{add}q \quad %rdi,%rdi \\
\quad 4004e7 : \text{ret}q \\
4004e8 \ <main> : & \\
\quad 4004e8 : \text{mov}q \quad $100,%rdi \\
\quad 4004ef : \text{call}q \quad 4004db \\
\quad 4004f4 : \text{mov}q \quad %rdi,%rax \\
\quad 4004f7 : \text{ret}q \\
\end{array}
\]

\[
\begin{array}{|c|c|}
\hline
\%rip & ??? \\
\hline
\%rsp & 0x7fffffffffe490 \\
\hline
\%rdi & 220 \\
\hline
\%rax & 220 \\
\hline
\end{array}
\]

\[
\begin{array}{|c|c|}
\hline
0x7fffffffffe488 & ??? \\
\hline
0x7fffffffffe480 & 0x4004f4 \\
\hline
0x7fffffffffe478 & 0x4004e4 \\
\hline
0x7fffffffffe470 & ??? \\
\hline
\end{array}
\]

...