CSCI 3330
Cybersecurity

Word of the day

SHOTGUN DEBUGGING n.

the making of relatively undirected changes to software in the hope that a bug will be perturbed out of existence. This almost never works, and usually introduces more bugs.

Agenda for today

- Dynamic linking
- Format strings
- Format string vulnerabilities?

Steps to compile a C program

- Preprocessing
- Compiling
- Assembling
- Linking

Libraries

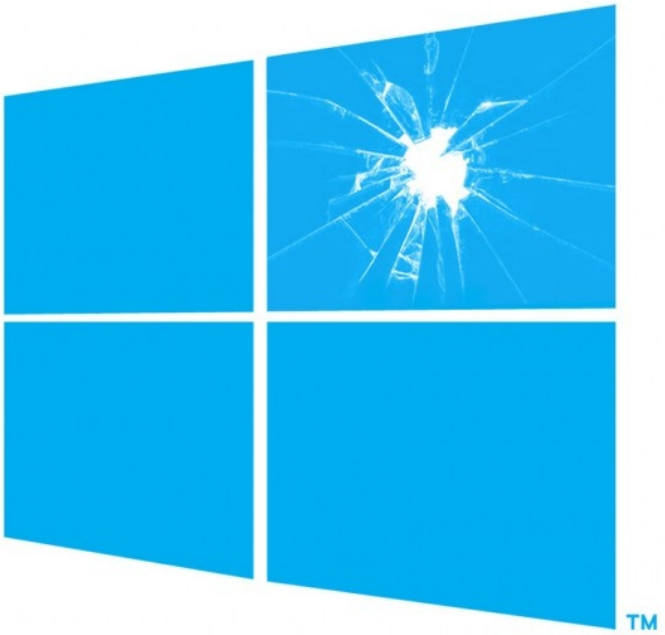
- Collection of functions (and data)
- Static linking: linked at compile time
  - Advantage: don’t need additional libraries at runtime
- Dynamic linking: linked at runtime
  - Advantage: can update libraries independently
  - Saves space and memory

Dynamic libraries

- *.so (Linux)
- *.dylib (macOS)
- *.dll (Windows)

Dynamic libraries

- *.so (Linux)
- ~~*.dylib (macOS)~~
- ~~*.dll (Windows)~~



Dynamic linking

```
$ ldd /usr/bin/whoami
linux-vdso.so.1 (0x00007ffed67fa000)
libc.so.6 => /lib/x86_64-linux-gnu/libc.so.6 (0x000072fc4a000000)
/lib64/ld-linux-x86-64.so.2 (0x000072fc4a2ce000)
```

Global offset table (GOT)

- Entry in table for every dynamically linked function
- Each entry initially they point to code in the PLT

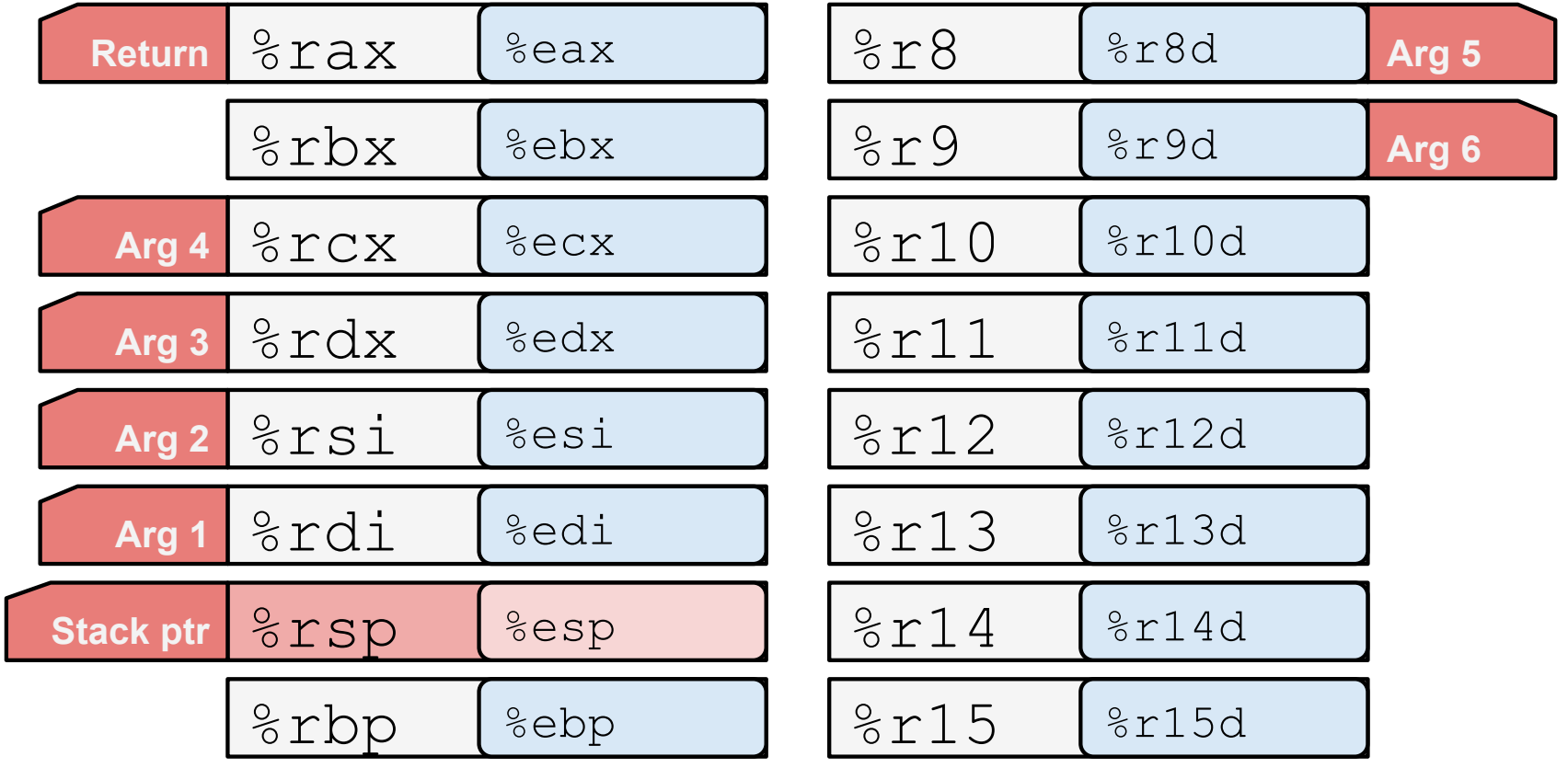
Procedure Linkage Table (PLT)

- The entry for each dynamically linked function contains executable code to jump into resolver
- Resolver (in ld.so) finds the address of the function
- Updates GOT to point away from the PLT to the actual function (so future calls call it directly)
- Finally, jumps to actual function

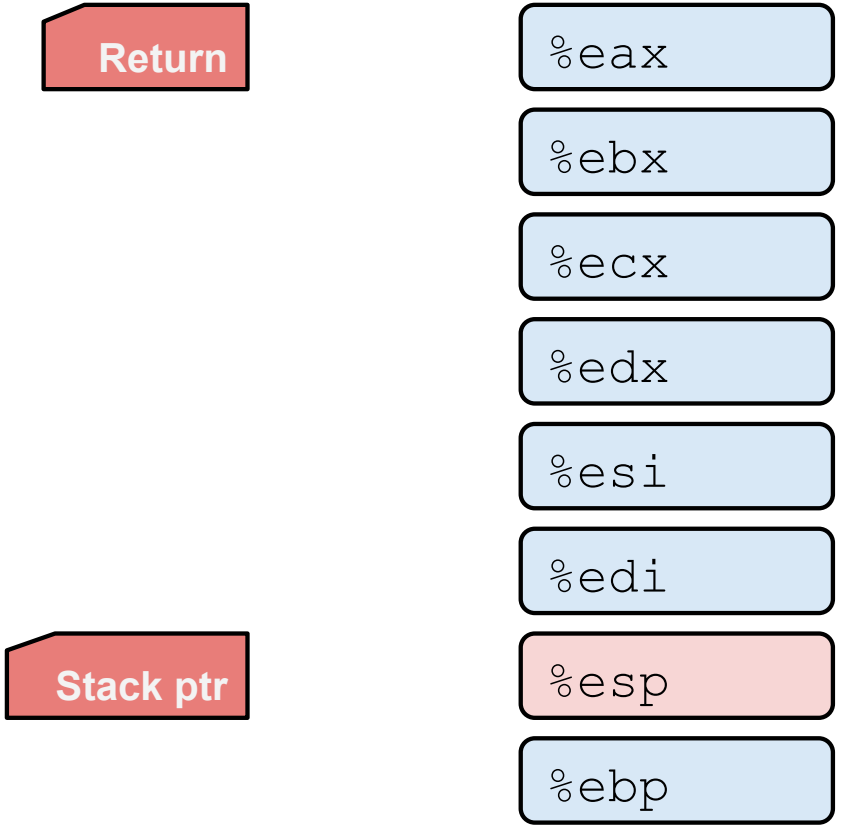
Calling conventions

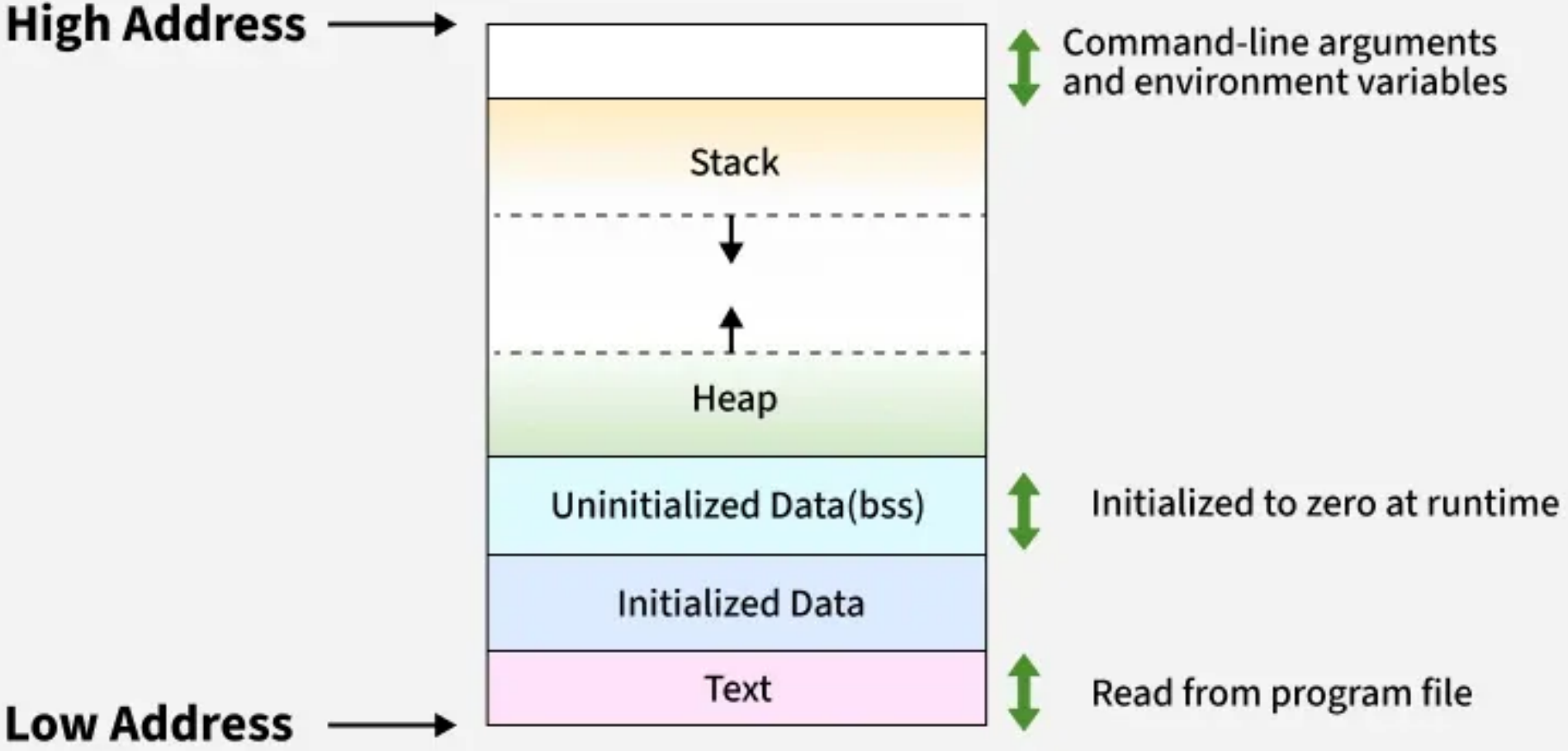
- The ISA-level “contract” for calling functions
- If I call a function in a library, how does it “know”
  - Where the arguments are?
  - Where the return value goes?
  - Who preserves which registers?

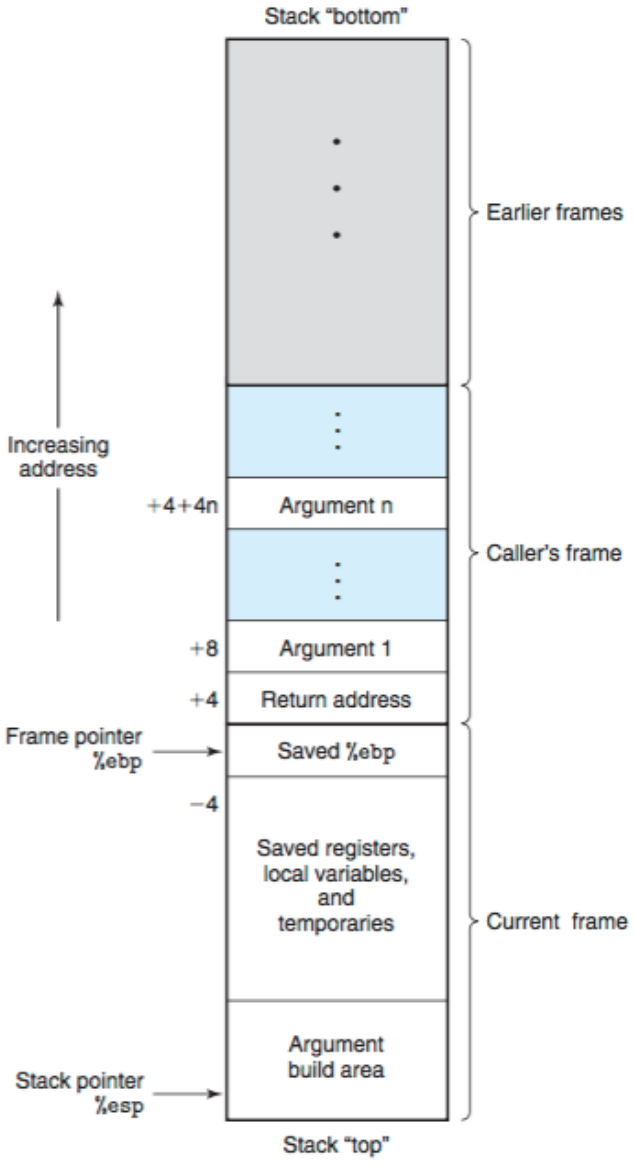
x86 (64-bit) calling convention



x86 (32-bit) calling convention




High Address
Command-line arguments
and environment variables
Stack
Heap
Uninitialized Data(bss)
Initialized to zero at runtime
Initialized Data
Text
Read from program file
Low Address


Stack "bottom"
Earlier frames
Increasing address
+4+4n
Argument n
Caller's frame
+8
Argument 1
+4
Return address
Frame pointer %ebp
Saved %ebp
−4
Saved registers, local variables, and temporaries
Current frame
Argument build area
Stack pointer %esp
Stack "top"

Format string vulnerabilities

- *Yet another* vulnerability arising from an in-band signal

Format strings

- *Yet another* vulnerability arising from an in-band signal
- ```
  const char *user_name = ...;
  int uid = ...;
  double grade = ...;
  printf("Username: %s, uid: %d, grade: %f\n",
         user_name, uid, grade);
  ```
  ```

Format strings

- Python support!
- `>>> '%s' % 'hello, world'`
  `'hello, world'`
- `>>> '%s %d %f' % ('hello', 1234, 4.2)`
  `'hello 1234 4.200000'`

Format of a format string

- `%[$][flags][width][.precision][length modifier]`**`conversion`**
- %s – string
- %d – decimal integer
- %x – hexadecimal integer
- %p – pointer
- %f – double-precision float
- %n – output # of bytes written so far to int pointer

What does this print?

```
unsigned int a = 0xdeadbeef;
printf("0x%x\n", a);
```

What does this print?

```
    unsigned int a = 0xdeadbeef;
    printf("0x%x\n", a);
```

```
0xdeadbeef
```

What does this print?

```
int a;
printf("hello%n\n", &a);
printf("%d\n", a);
```

What does this print?

```
    int a;
    printf("hello%n\n", &a);
    printf("%d\n", a);
```

```
hello
5
```

Format of a format string

- `%[$][flags][width][.precision][length modifier]conversion`
- `>>> '%x' % 0xdeadbeef`

Format of a format string

- `%[$][flags][width][.precision][length modifier]conversion`
- ```
  >>> '%x' % 0xdeadbeef
  'deadbeef'
  ```
  ```

Format of a format string

- `%[$][flags][width][.precision][length modifier]conversion`
- `>>> '%x' % 0xdeadbeef`
  `'deadbeef'`
- `>>> '%.16x' % 0xdeadbeef`

Format of a format string

- `%[$][flags][width][.precision][length modifier]conversion`
- `>>> '%x' % 0xdeadbeef`
  `'deadbeef'`
- `>>> '%.16x' % 0xdeadbeef`
  `'00000000deadbeef'`

Format of a format string

- `%[$][flags][width][.precision][length modifier]conversion`
- ```
  >>> '%x' % 0xdeadbeef
  'deadbeef'
  ```
- ```
  >>> '%.16x' % 0xdeadbeef
  '00000000deadbeef'
  ```
- `>>> '%.1000x' % 0xdeadbeef`

Format of a format string

- `%[$][flags][width][.precision][length modifier]conversion`
- ```
  >>> '%x' % 0xdeadbeef
  'deadbeef'
  ```
- ```
  >>> '%.16x' % 0xdeadbeef
  '00000000deadbeef'
  ```
- ```
  >>> '%.1000x' % 0xdeadbeef
  '0' * (1000 - 8) + 'deadbeef'
  ```
  ```

Format of a format string

- `%[$][flags][width][.precision][length modifier]conversion`
- h – half length
- hh – half half (quarter) length
- l – long
- ll – long long

What does this print?

```
unsigned int a = 0xdeadbeef;
printf("0x%x\n", a);
```

What does this print?

```
    unsigned int a = 0xdeadbeef;
    printf("0x%x\n", a);
```

```
0xdeadbeef
```

What does this print?

```
unsigned short a = 0xbeef;
printf("0x%hx\n", a);
```

What does this print?

```
    unsigned short a = 0xbeef;
    printf("0x%hx\n", a);
0xbeef
```

What does this print?

```
unsigned char a = 0xbe;
printf("0x%hhx\n", a);
```

What does this print?

```
unsigned char a = 0xbe;
printf("0x%hhx\n", a);
```

```
0xbe
```

What does this print?

```
int a;
printf("hello%n\n", &a);
printf("%d\n", a);
```

What does this print?

```
    int a;
    printf("hello%n\n", &a);
    printf("%d\n", a);
```

```
hello
5
```

What does this print?

```
short a;
printf("hello%hn\n", &a);
printf("%hd\n", a);
```

What does this print?

```
    short a;
    printf("hello%hn\n", &a);
    printf("%hd\n", a);
```

```
hello
5
```

What does this print?

```
char a;
printf("hello%hhn\n", &a);
printf("%hhd\n", a);
```

What does this print?

```
    char a;
    printf("hello%hhn\n", &a);
    printf("%hhd\n", a);
```

```
hello
5
```

Assignment 7

- Deadline: November 18, 2025, 10:00pm
- Textbook RSA attacks
  https://tildesites.bowdoin.edu/~j.knockel/csci3330f25/assignment7.pdf