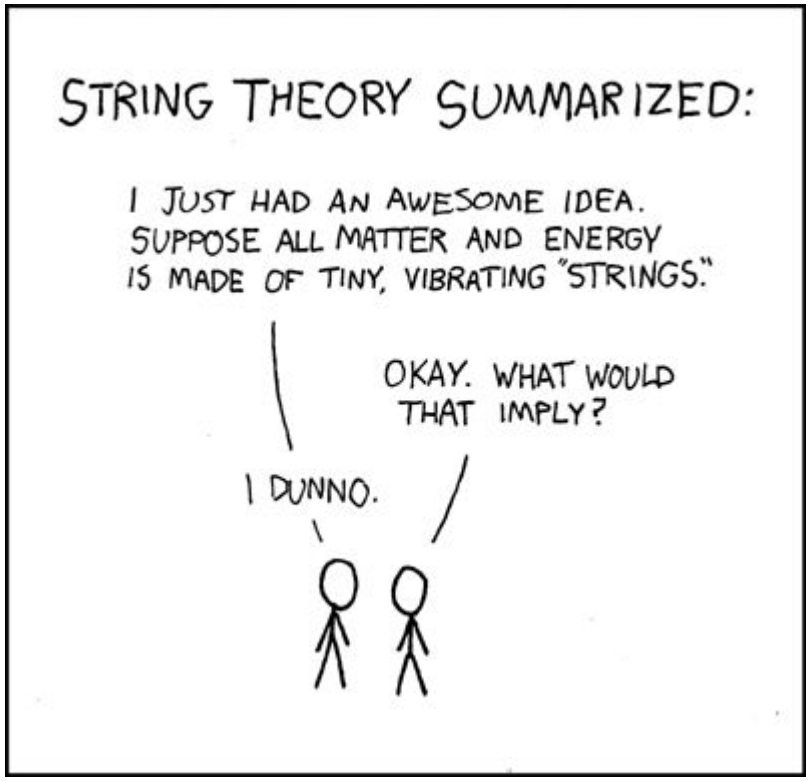
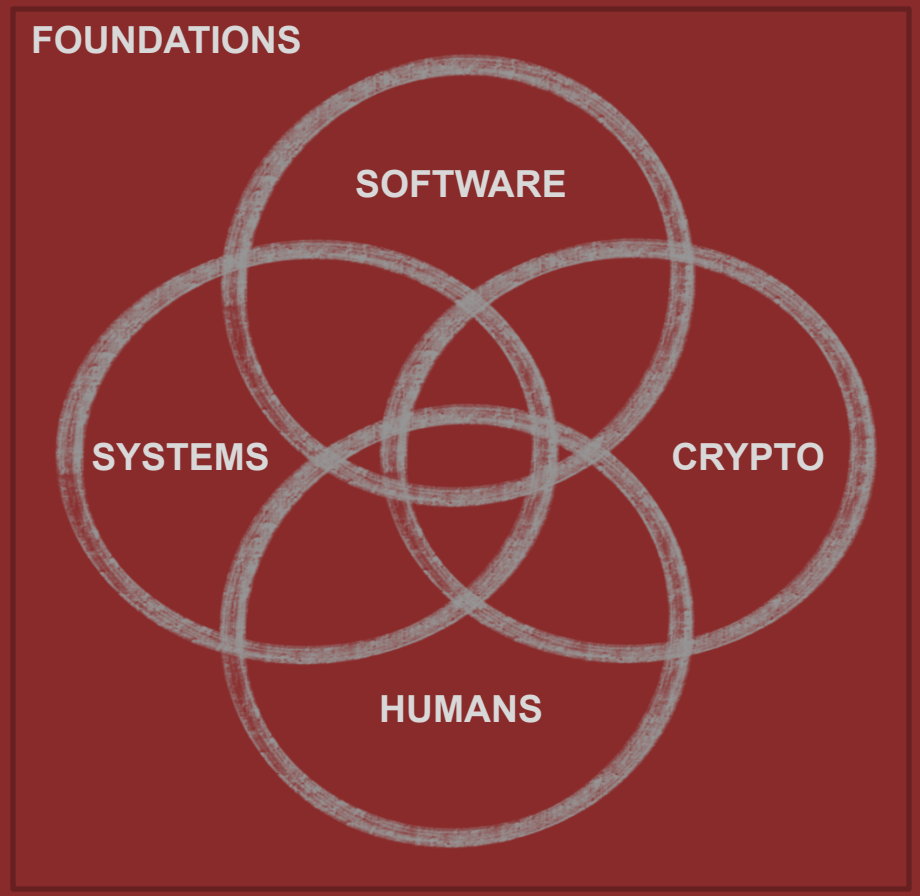
Διάλεξη #5 - Format String Attacks



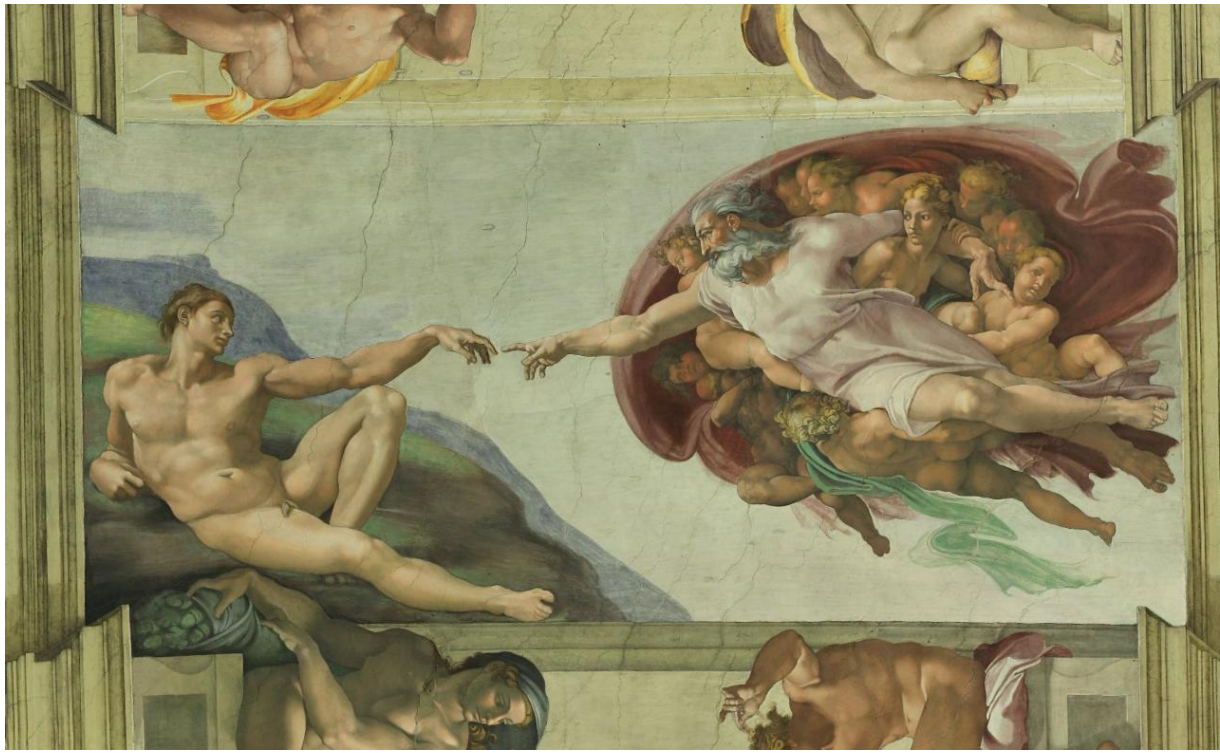


Huge thank you to David Brumley from Carnegie Mellon University for the guidance and content input while developing this class

Ανακοινώσεις / Διευκρινίσεις

Την Προηγούμενη Φορά

- Control Flow Hijack Attacks
- Basics of buffer overflow attacks continued (shellcode + nopsled)



Σήμερα

1. x86 Fundamentals continued
2. Variadic Functions
3. Format String Attacks

Debugging με GDB

1. gcc -g -ggdb -o prog prog.c
2. gdb --args ./program arg1 arg2
3. run, break, step, continue, finish
4. backtrace
5. print / x commands
6. Cheat Sheet

What if I reverse the order in which I write to memory - would I be safe?

Two more x86 Basic Concepts

Memory can be addressed with more than [register]

An ***Addressing Mode*** specifies how to calculate the effective memory address of an operand by using information from registers and constants contained with the instruction or elsewhere.

Motivation:
Common C memory index patterns

```
Type buf[s];
buf[index] = *(<buf addr>+sizeof(Type)*index)
```

Form	Meaning on memory M	Example at&t
imm (r)	M[r + imm]	-8(%rbp)
imm (r_1, r_2)	M[r_1 + r_2 + imm]	-16(%rbx, %rcx)
imm (r_1, r_2, s)	M[r_1 + r_2*s + imm]	-8(%rdx, %r9, 48)
imm	M[imm]	0x12345678

Referencing Memory

Loading a value from memory: `mov`

```
# <rax> = *buf;
mov  -0x38(%rbp),%rax (A)
mov  rax, [rbp-0x38] (I)
```

Loading Effective Address: `lea`

```
# <rax> = buf;
lea  -0x38(%rbp),%rax (A)
lea  rax, [rbp-0x38] (I)
```

Assembly is spaghetti

Abstractions you know and love

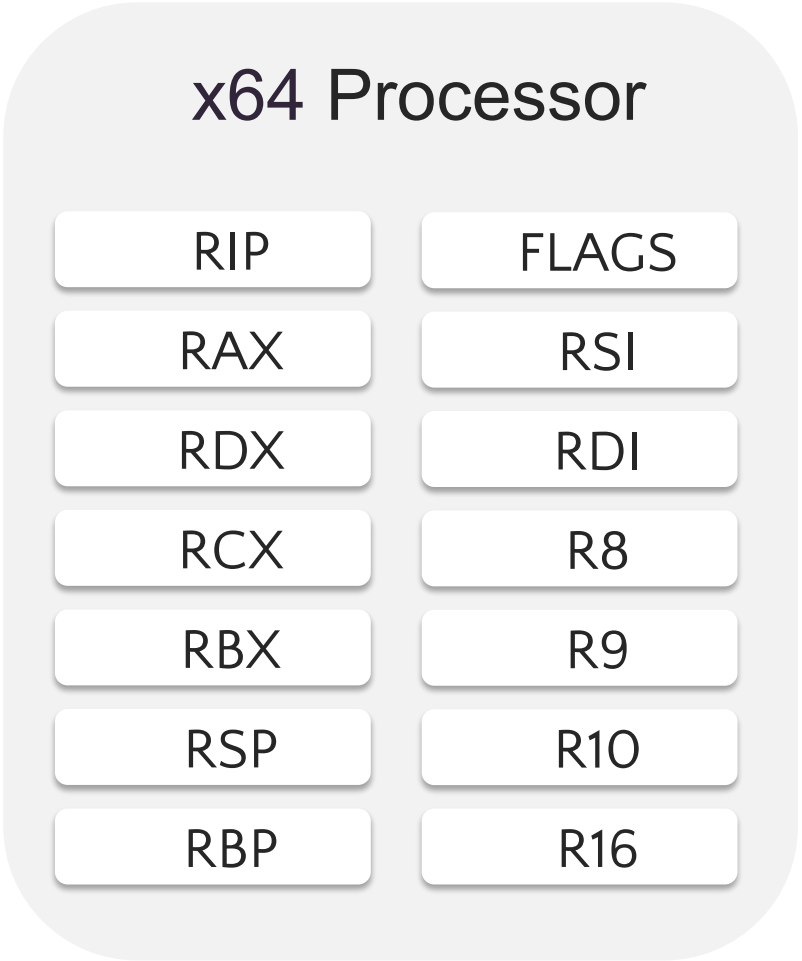
- if-then-else
- functions
- for loops
- while loops

What the machine executes

- Direct jumps: `jmp <addr>`
- Indirect jumps: `jmp <register>`
- Branch: `if <flag> goto line`

Two types of unconditional control flow

- **Direct jump**: `jmp 0x45`
- **Indirect jump**: `jmp *rax`



Note: Typically no direct way to set or get RIP

A very special register: EFLAGS

- EFLAGS are hardware bits used to determine control flow
- Set via instructions implicitly.
- “cmp b,a”: calculate a-b and set flags:
  - Was there a carry? (CF Flag set)
  - Was the result zero? (ZF Flag set)
  - What was the parity of the result? (PF flag)
  - Did overflow occur? (OF Flag)
  - Is the result signed? (SF Flag)

‘if’ implementation pseudocode

C code

```
if (x ≤ y)
  return 1;
else
  return 0;
```

Assembly

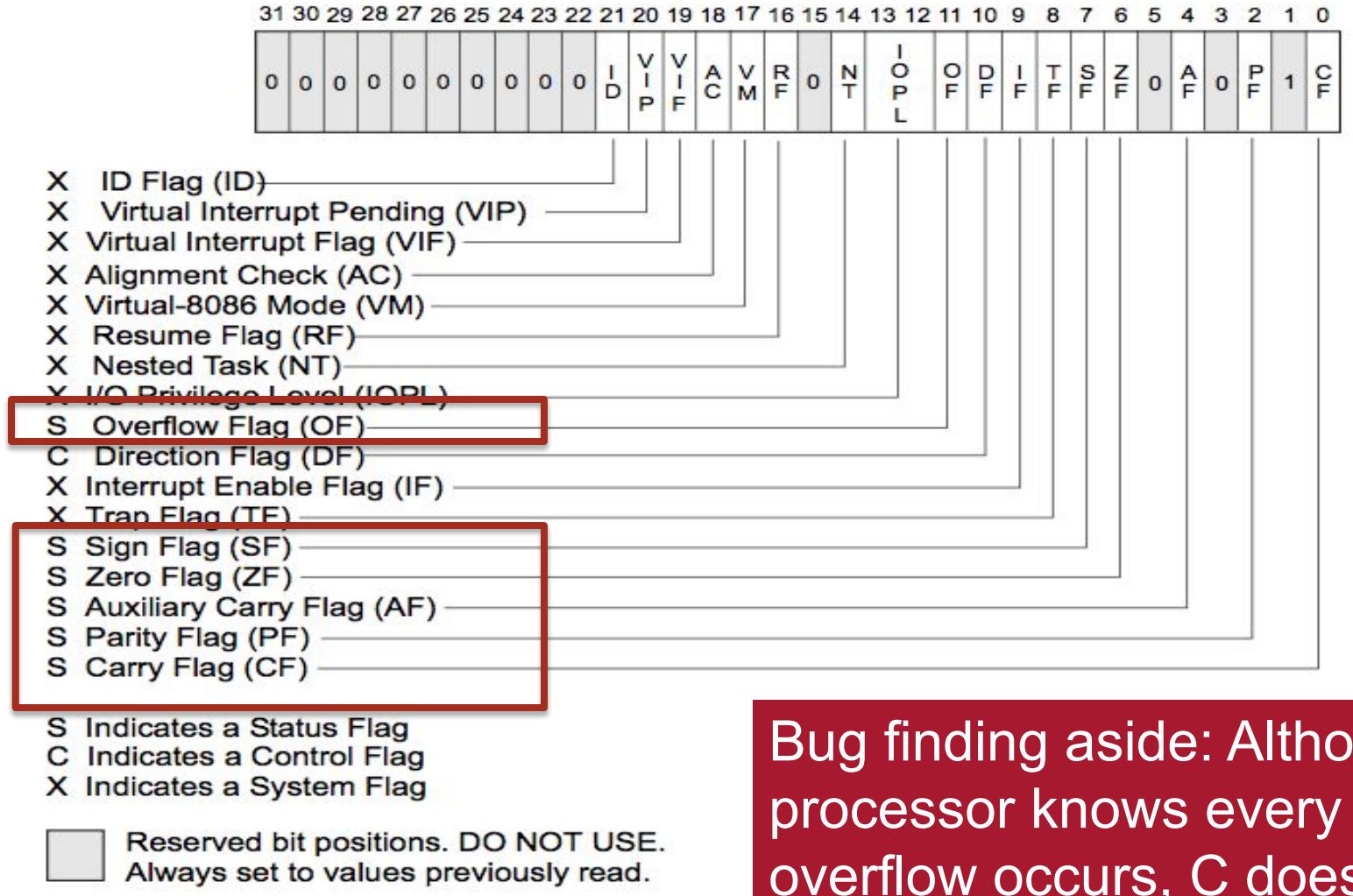
```
d:  cmp   -0x8(%rbp),%eax
10: jg    19 <if_then_else+0x19>
12: mov   $0x1,%eax
17: jmp   1e <if_then_else+0x1e>
19: mov   $0x0,%eax
```

Line d: calculate
%eax - mem[ebp-0x8]

- sets ZF=0 if the result is zero
- sets SF if the result is negative

Line 10: Semantically, jump if eax is greater when

- If ZF = 0 and SF=0, then the result is non-negative so eax was greater
- If SF=1 and OF=1, the result is negative but overflow occurred, which means eax is still greater
- Else eax is smaller



From the Intel x86 manual

Bug finding aside: Although the x86 processor knows every time integer overflow occurs, C does not make this result visible.

See the x86 manuals available on Intel's website for more information

Instr.	Description	Condition
JO	Jump if overflow	OF == 1
JNO	Jump if not overflow	OF == 0
JS	Jump if sign	SF == 1
JZ	Jump if zero	ZF == 1
JE	Jump if equal	ZF == 1
JL	Jump if less than	SF <> OF
JLE	Jump if less than or equal	ZF ==1 or SF <> OF
JB	Jump if below	CF == 1
JP	Jump if parity	PF == 1

Switching Gears: Variadic Functions

Variadic Συναρτήσεις

Συναρτήσεις που έχουν μεταβαλλόμενο αριθμό ορισμάτων (π.χ., printf) λέγονται variadic. Δηλώνουμε τον μεταβλητό αριθμό ορισμάτων χρησιμοποιώντας την έλλειψη (ellipsis): …

Παραδείγματα:

```
int printf(const char * format, ...);
int scanf(const char * format, ...);
```

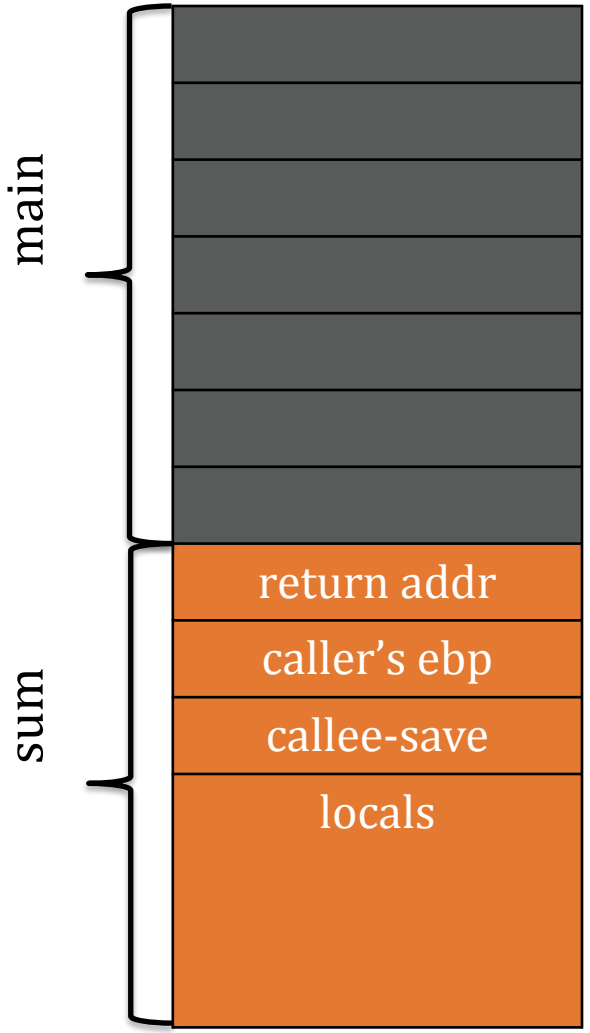
Παράδειγμα με Variadic Function

```c
#include <stdarg.h>

int sum(int count, ...) {
  int result = 0;
  va_list args;
  va_start(args, count);
  for(int i = 0; i < count; i++)
     result += va_arg(args, int);
  va_end(args);
  return result;
}

int main() {
  return sum(6, 1, 2, 3, 4, 5, 6);
}
```

```
$ ./variadic
$ echo $?
21
```

Χρησιμοποιώντας τις "μαγικές" συναρτήσεις va_list, va_start, va_arg, va_end μπορούμε να διατρέξουμε όλα τα ορίσματα (δεν ξέρουμε τι κάνουν; χρησιμοποιούμε man!)

Stack Diagram

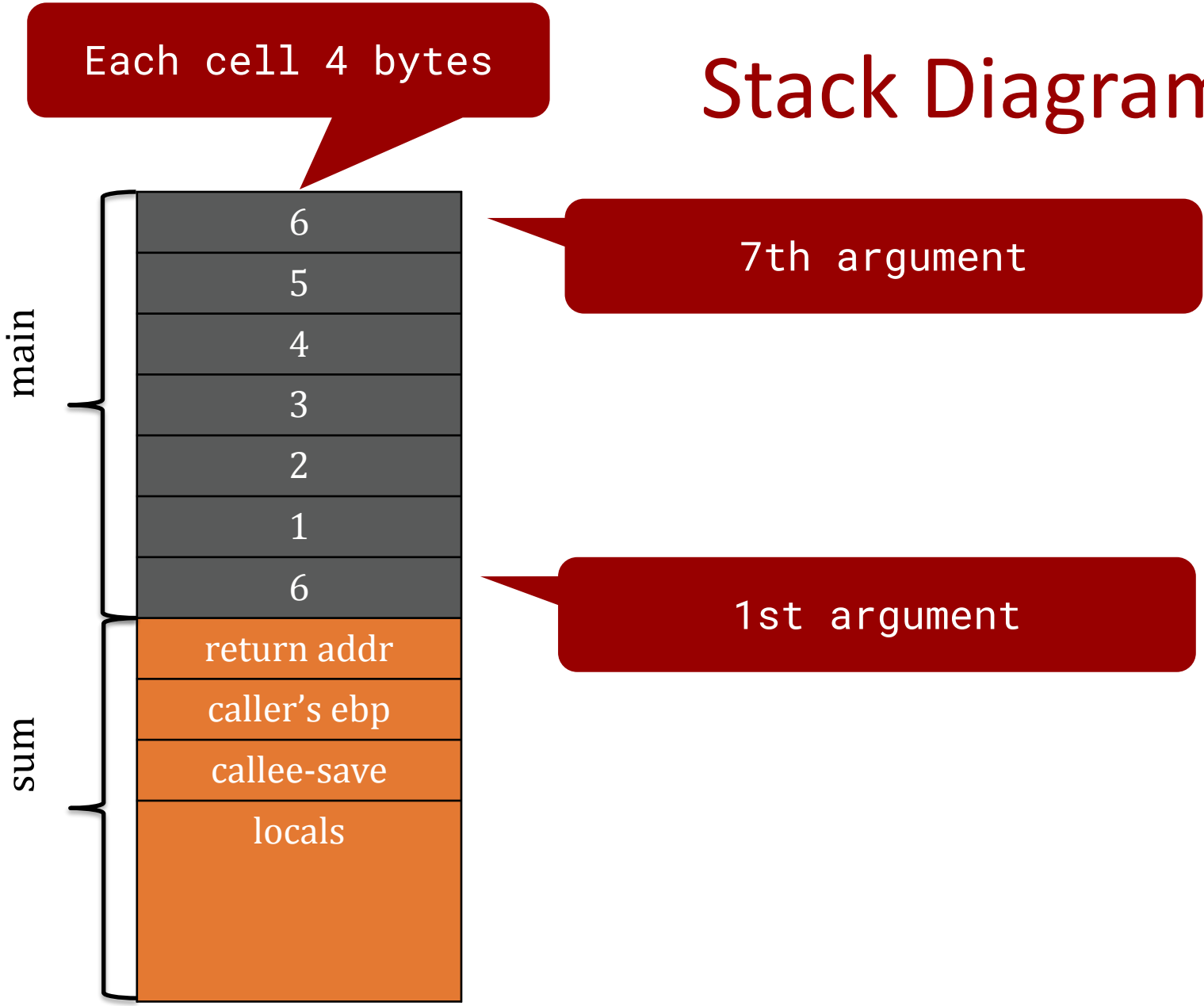


Μόλις καλέσαμε την

```
sum(6, 1, 2, 3, 4, 5, 6)
```

Πως είναι το stack;

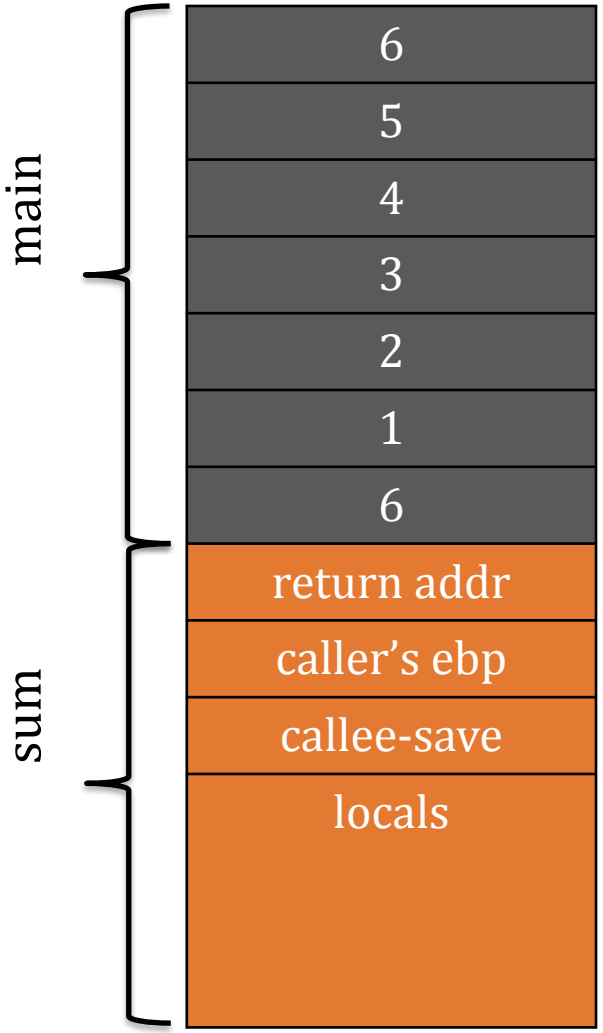
Stack Diagram



Why do you think va_list is initialized with "count"? Careful, this is a trick question!

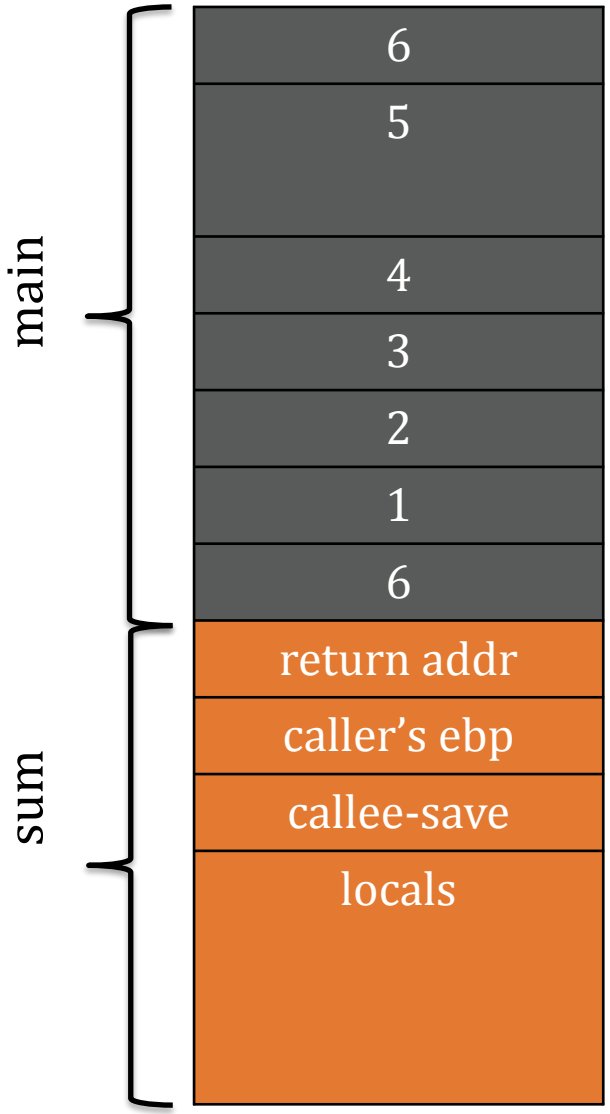
```
va_list args;

va_start(args, count);

for(int i = 0; i < count; i++)
```

Stack Diagram



What would happen if the argument corresponding to **5** was of type int64_t?

Stack Diagram



What would happen if the argument corresponding to **5** was of type int64_t?

What about the va_arg call? Does that need to change?

```
for(int i = 0; i < count; i++)

 result += va_arg(args, int);
```

Variadic Functions

... are functions of ***indefinite arity***

Widely supported in languages:

- C
- C++
- Javascript
- Perl
- PHP
- ...

In cdecl, caller is responsible to clean up the arguments

Why?

Example Format String Functions

Specifies number and types of arguments

Variable number of arguments

```
printf(char *fmt, ...)
```

Function	Purpose
printf	prints to stdout
fprintf	prints to a FILE stream
sprintf	prints to a string
vfprintf	prints to a FILE stream from va_list
syslog	writes a message to the system log
setproctitle	sets argv[0]

Generally useful, but …

Format String Attacks

"If an attacker is able to provide the format string to an ANSI C format function in part or as a whole, a format string vulnerability is present." – scut/team teso

Assembly View

- For **non-variadic** functions, the compiler:
  - knows number and types of arguments
  - emits instructions for caller to put arguments into registers and push extra arguments right to left
  - emits instructions for callee to access arguments in registers or via frame pointer (or stack pointer [advanced])

- For **variadic** functions, the program dynamically determines which registers and stack slots have arguments based upon a format specifier.

Example (1/3)

Suppose we want to implement a `printf`-like function that only prints when a debug key is set:

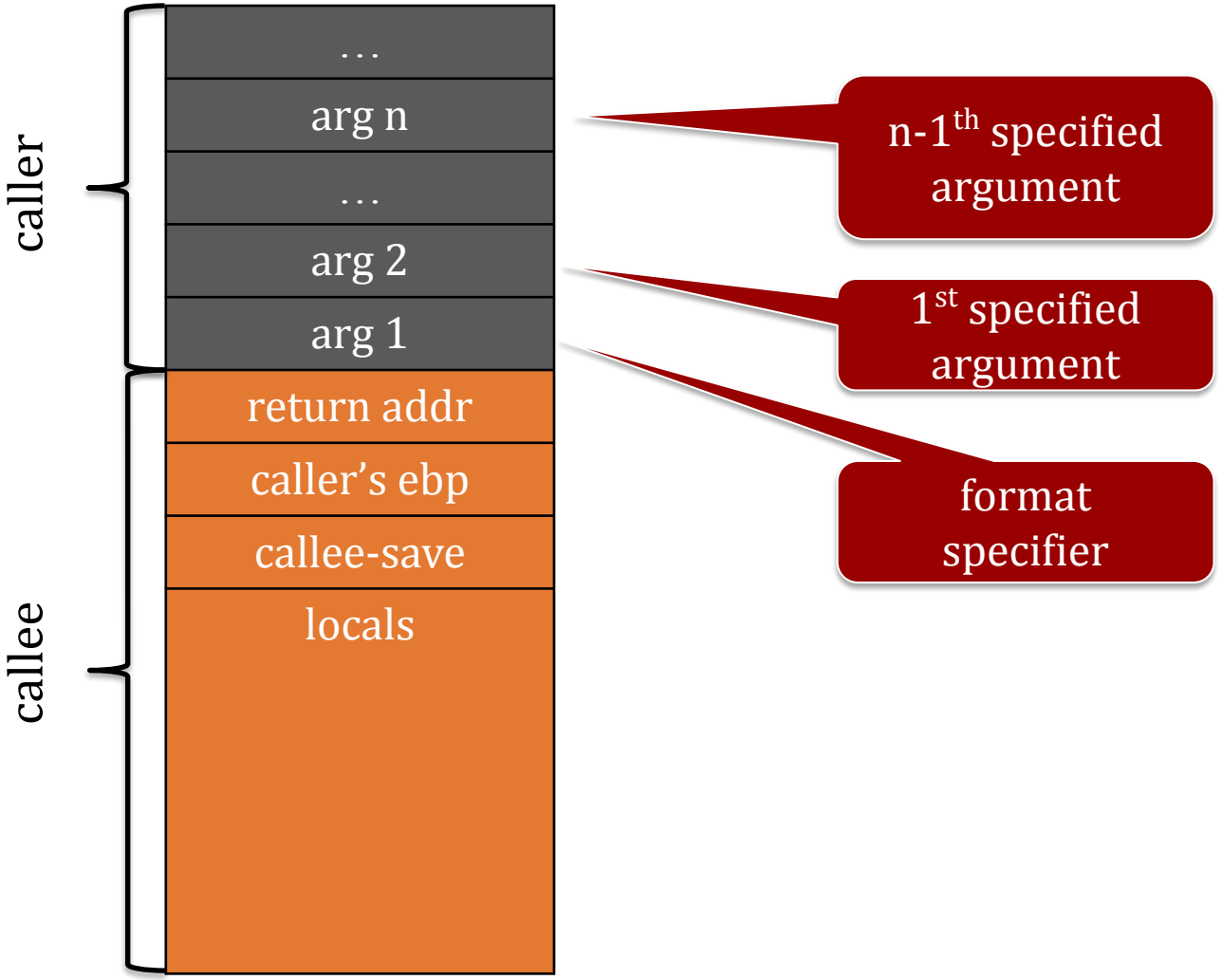
```
void debug(char *key, char *fmt, ...) {
  va_list ap;
  char buf[BUFSIZE];

  if (!KeyInList(key)) return;

  va_start(ap, fmt);
  vsprintf(buf, fmt, ap);
  va_end(ap);
  printf("%s", buf);
}
```

Set up ap to point to stack using last fixed argument

Call vsprintf with args

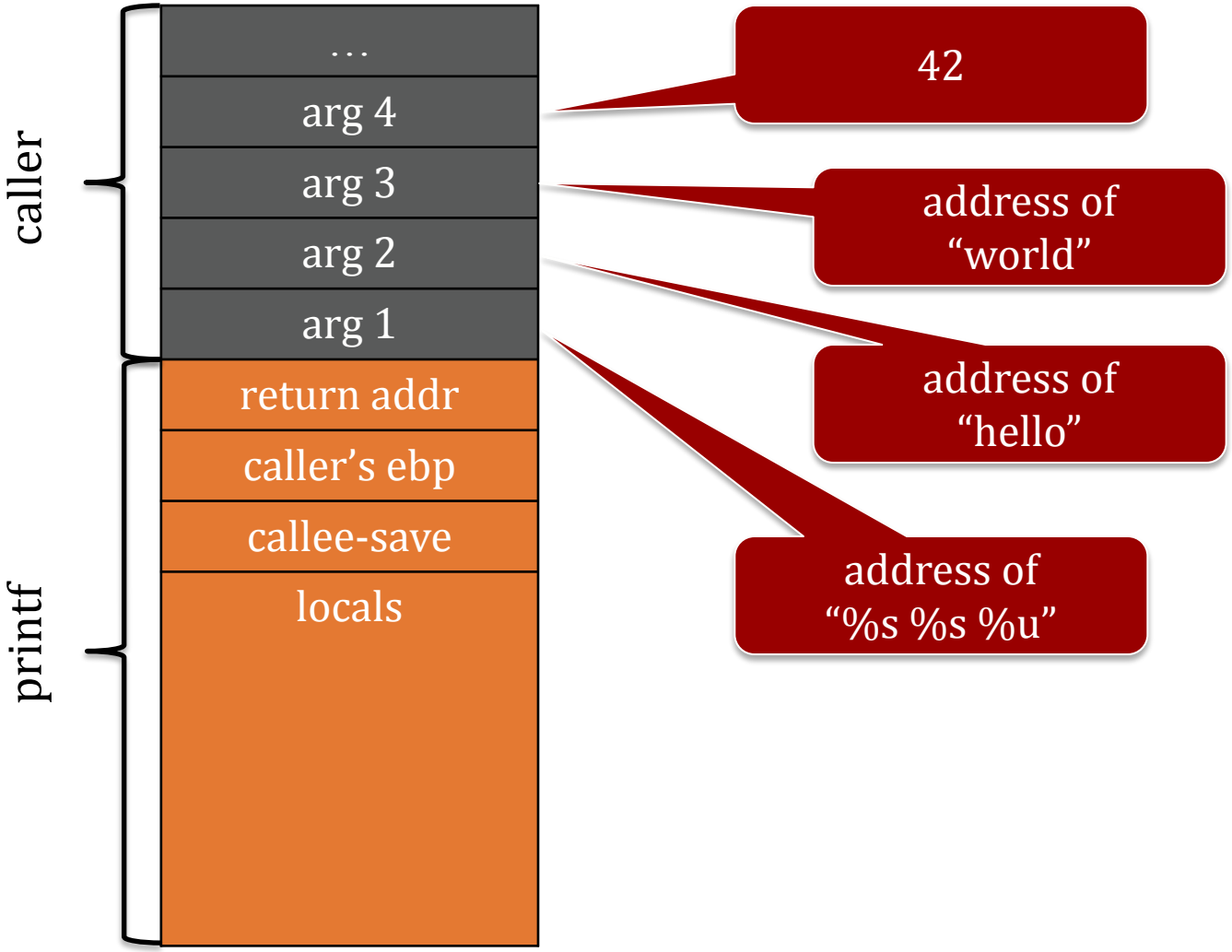
Cleanup

Stack Diagram



- Think of `va_list` as a pointer to the second argument (first after format)
- Each format specifier indicates ***type*** of current arg
  - Know how far to increment pointer for next arg

Example (2/3)




```
char s1[] = "hello";
char s2[] = "world";
printf("%s %s %u",
        s1, s2, 42);
```

Example (3/3)

```
#include <stdio.h>
#include <stdarg.h>
void foo(char *fmt, ...) {
        va_list ap;
        int d;
        char c, *p, *s;

        va_start(ap, fmt);
        while (*fmt)
                switch(*fmt++) {
                case 's':                       /* string */
                        s = va_arg(ap, char *);
                        printf("string %s\n", s);
                        break;
                case 'd':                       /* int */
                        d = va_arg(ap, int);
                        printf("int %d\n", d);
                        break;
                case 'c':                       /* char */
                        /* need a cast here since va_arg only
                           takes fully promoted types */
                        c = (char) va_arg(ap, int);
                        printf("char %c\n", c);
                        break;
                }
        va_end(ap);
}
```

```
foo(“sdc”, “Hello”, 42, ‘A’);
  =>
string Hello
int 42
char A
```

Conversion Specifications

%[flag][width][.precision][length]specifier

Specifier	Output	Passed as
%d	decimal (int)	value
%u	unsigned decimal (unsigned int)	value
%x	hexadecimal (unsigned int)	value
%s	string (const unsigned char *)	reference
%n	# of bytes written so far (int *)	reference

0 flag: zero-pad

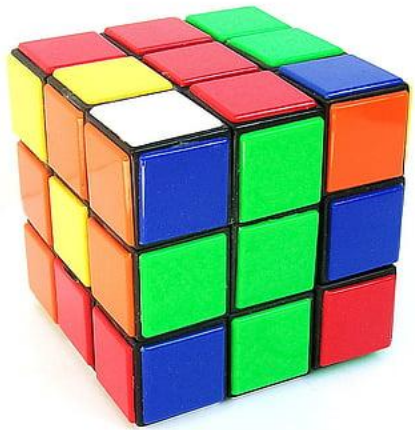
- `%08x`
  zero-padded 8-digit hexadecimal number

Minimum Width

- `%3s`
  pad with up to 3 spaces
- printf("S:%3s", "1");
  `S:  1`
- printf("S:%3s", "12");
  `S: 12`
- printf("S:%3s", "123");
  `S:123`
- printf("S:%3s", "1234");
  `S:1234`

```
man -s 3 printf
```

Ένας Γρίφος



I need 2-3 volunteers

```
ssh ubuntu@44.203.64.255
ilovedi@uoa
```

Μπορούμε να μαντεύουμε σωστά κάθε φορά;

```
int main(int argc, char ** argv) {
  int number, guess;
  srand(time(0));
  number = rand(); // Generates a random number between 0 and RAND_MAX
  if (argc > 1) printf(argv[1]);
  printf("\nGuess the number: "); fflush(stdout);
  scanf("%d", &guess);
  if (guess == number) {
    printf("Congratulations! You guessed the number in one shot. HOW?\n");
    return 0;
  }
```

Snippet from https://github.com/ethan42/fmt0/blob/master/guessme.c

Ευχαριστώ και καλή μέρα εύχομαι!

Keep hacking!