CS 354 - Machine Organization & Programming
Tuesday, November 19, 2019

Project p5 (4.5%): DUE at 10 pm on Monday, December 2nd

Project p6 (4.5%): Assigned on Tuesday, November 26th

Homework hw6 (1.5%): DUE TOMORROW at 10 pm on Wednesday, November 20th

Today is last chance to pick up exams from me at lecture.

Last Time

Stack Allocated Arrays in Assembly
Stack Allocated Multidimensional Arrays
Stack Allocated Structs
Alignment
Alignment Practice
Unions

Today

Unions (from last time)
Pointers
Function Pointers
Buffer Overflow & Stack Smashing

Flow of Execution
Exceptional Events
Kinds of Exceptions

Next Time

More Exceptions, Processes, Context Switches
Read: B&O 8.1 - 8.3, 8.4 through p. 719

Pointers

Recall Pointer Basics in C

```
int i = 11;
int *iptr = &i;
*iptr = 22;
```

pointer type	`int *`
pointer value	`0x2A300F87`, 0x00000000 (NULL)
address of	`&i`
dereferencing	`*iptr`

Recall Casting in C

```
int *p = malloc(sizeof(int) * 11);

... (char *)p + 2
```

Function Pointers

What? A *function pointer*

◆

◆

Why?

◆

◆

How?

```
int func(int x) { ...}      //1.

int (*fptr)(int);           //2.

fptr = func;                //3.

int x = fptr(11);           //4.
```

Example

```
#include <stdio.h>

void add     (int x, int y) { printf("%d + %d = %d\n", x, y, x+y); }
void subtract(int x, int y) { printf("%d - %d = %d\n", x, y, x-y); }
void multiply(int x, int y) { printf("%d * %d = %d\n", x, y, x*y); }

int main() {
    void (*fptr_arr[])(int, int) = {add, subtract, multiply};
    unsigned int choice;
    int i = 22, j = 11; //user should input

    printf("Enter: [0-add, 1-subtract, 2-multiply]\n");
    scanf("%d", &choice);
    if (choice > 2) return -1;
    fptr_arr[choice](i, j);
    return 0;
}
```

Buffer Overflow & Stack Smashing

Bounds Checking

```
int a[5] = {1,2,3,4,5};
printf("%d", a[11]);
```

→ What happens when you execute the code?

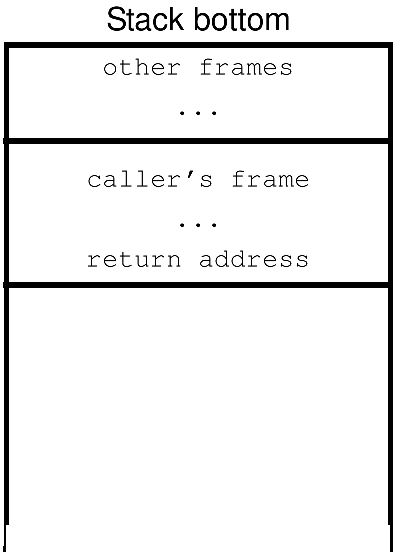
❋ *The lack of bounds checking array accesses*

Buffer Overflow

-

-

```
void echo() {
   char bufr[8];
   gets(bufr);
   puts(bufr);
}
```



Stack bottom

other frames
...

caller's frame
...
return address

Stack Smashing

1. Get "exploit code" in

2. Get "exploit code" to run

3. Cover your tracks

Flow of Execution

What?

<u>control transfer</u>

<u>control flow</u>

- ➢ What control structure results in a smooth flow of execution?

- ➢ What control structures result in abrupt changes in the flow of execution?

Exceptional Control Flow

<u>logical control flow</u>

<u>exceptional control flow</u>

<u>event</u>

<u>processor state</u>

Some Uses of Exceptions

process

OS

hardware

Exceptional Events

What? An *exception*

◆

◆

◆

→ What’s the difference between an asychronous vs. a synchronous exception?

asynchronous

synchronous

General Exceptional Control Flow

Application Exception Handler

0. normal flow I_0

I_1

1.

2.

3.

4.

Kinds of Exceptions

→ Which describes a ***Trap***? ***Abort***? ***Interrupt***? ***Fault***?

1.

signal from external device
asynchronous
returns to Inext

How? Generally:

1.

2.

3. transfer control to appropriate exception handler

4. transfer control back to interrupted process's next instruction

vs. polling

2.

intentional exception
synchronous
returns to Inext

How? Generally:

1.

   `int`

2. transfer control to the OS system call handler

3. transfer control back to process's next instruction

3.

potentially recoverable error
synchronous
might return to Icurr and re-execute it

4.

nonrecoverable fatal errors
synchronous
doesn't return