

## C does not Enforce Memory Safety

### Many ways to access memory illegally

- Accessing an array out of bounds
- Bad pointer arithmetic manipulations
- Using a pointer to deallocated memory
  - Returning a pointer to a local
  - Deallocated with free() [More on this next week]
- Other strangeness

### Many such bugs uncovered only in strange situations

- When stressing a bounds of a system
- For example, manipulating large files, large input strings, etc.

### Java does enforce memory safety

- Exceptions triggered for such errors

CSE 240

2

## In-Class Demonstration: C Code

Consider the following code

```
void read_array()
{
    int r5, r1, array[6];
    r1 = 0;
    while ((r5 = get_hex_from_keyboard()) != 0) {
        array[r1] = r5;
        r1 = r1 + 1;
    }
    // More code here to manipulate array (sum it, etc.)
}
```

Code puts hex values from keyboard into the array

- Stops when zero value entered at keyboard
- What happens if the users enters many numbers?

CSE 240

7

## Problems due to Lack of Safety

### Crashes

- Program accesses illegal memory, crashes
- Ok for programs (save often!)
  - Why?
- Not so good for the operating system
  - Blue screen of death (BSOD)

### Non-Crash Errors

- Strange glitches
- Worst-case: payroll system prints a million dollar check

### Intentional Exploits

- These bugs are often repeatable
- Can be exploited for great mischief and harm

CSE 240

6

## In-Class Demonstration: LC-3 Code

```
READ_AR ADD R6, R6, #-10 ; push stack pointer
        STR R7, R6, #9   ; save R7 (link register)
        STR R5, R6, #8   ; save R5
        STR R1, R6, #7   ; save R1
        STR R0, R6, #6   ; save R0
LOOP1  TRAP x26         ; R5 gets 4-digit hex number
        ADD R5, R5, #0   ; check for zero (to stop)
        BRz STOP
        ADD R0, R1, R6   ; calculate array address
        STR R5, R0, #0   ; store number into array
        ADD R1, R1, #1   ; increment counter
        BR LOOP1
STOP   LDR R7, R6, #9   ; restore R7 (link register)
        LDR R5, R6, #8   ; restore R5
        LDR R1, R6, #7   ; restore R1
        LDR R0, R6, #6   ; restore R0
        ADD R6, R6, #10  ; pop stack pointer
        RET
```

CSE 240

8

## Smashing the Stack

### Exploit a buffer or array overrun

- Overwrite return address of function
- Inject arbitrary code (processor specific)
- Take over machine

### Surely this is just an academic exercise? right?

- Ummm... No.

### Attacking...

- Servers: use a fake web browser to send a malicious string
- Clients: server sends malformed HTML, installs spyware
- Most machines are both clients and servers...

## Internet Worms

### A “Worm” is a self-propagating malicious program

- A program that attacks remote machines
- Installs a copy of itself after successful attack
- Attacks other hosts at random
- Continues to propagate, growing exponentially

### First internet worm: Morris Worm

- 1988 a self-propagating worm brought the internet to its knees
- Many since: Code Red, Nimda, Sasser

### Huge Internet Security Issue

- Disrupt internet services (Denial of Service)
- Can attack specific sites (Distributed Denial of Service)
- Other malicious scenarios too nasty to think about...

## Non-Executable Stack - A (Partial) Solution

### Prevent hardware from executing instruction on the stack

- Goal: prevent code-injection attacks on the stack
- A new register like Memory Protection Register (MPR)
  - Except controls instruction fetch (not loads or stores)
- For example, trying to execute the code in our example:
  - illegal execution exception

### Intel recently introduced “NX bit”

- Operating can set pages as “not executable”
- Other chips have had this for years
- Newest version of Windows supports this feature

### Only a partial solution...

## Subverting Non-Executable Stack

### Consider the following code:

```
int valid = get_and_check_password();
if (valid) {
    send_super_secret_datafile();
} else {
    invalid_password_error_message();
}
```

### If password check overflows a fixed-size array

- Smash the stack to “return” to `send_secret_datafile()`

### Other attacks are also possible

- Heap-based attacks, calling library functions, OS calls

### Non-Execute Stack: Helpful, but not a cure-all

## Non-Secure Code Example

**Buffer overflows often happen with C strings:**

```
char str[1024];  
get_string_from_network(str);  
- or -  
scanf("%s", str);
```

### Solutions:

- Always check for bounds
- `scanf("%1023s", str);`
  - Why 1023?
- Create a “string” or “array” struct (in C) or class (in C++)
- Use Java.

## Take-Away Points

**Memory errors in C are a real problem**

- Crash programs
- Security violations

**Lots of C code out in the world**

- Operating systems
- Web browsers
- Web servers
- Database servers
- Embedded systems...

**Programs that interact with the outside world must consider security**

- Almost all programs today
- Code defensively, use good programming practice
- Consider Java (and other memory-safe language)