# CSC209: Software Tools and Systems Programming

## Week 5: Strings [1]

Kianoosh Abbasi

[1] Slides are mostly taken from Andi Bergen's in summer 2021.

# PCRS: Strings Summary

- In C, a string is an array of `chars` terminated by `\0` (`NULL` byte)
- C standard library offers string manipulation functions, defined in `string.h`

# String Manipulation and Memory Safety

- String manipulation is a major cause of memory errors (e.g., *buffer overflow*)
- The C standard library includes both *safe* and *unsafe* string functions
  - **Some** unsafe functions can be used safely if the string is **guaranteed** to be NULL-terminated: `strlen(argv[0]);`
  - But even so-called "safe" functions can cause memory errors if used improperly: `char x[2]; strncpy(x, "blabla", 7);`

# Unsafe String Functions: Example

From `man gets` (Linux):

> *Never use gets(). Because it is impossible to tell without knowing the data in advance how many characters gets() will read, and because gets() will continue to store characters past the end of the buffer, it is extremely dangerous to use. It has been used to break computer security. Use fgets() instead.*

Also from `man gets` (Mac):

> *The gets() function cannot be used securely. Because of its lack of bounds checking, and the inability for the calling program to reliably determine the length of the next incoming line, the use of this function enables malicious users to arbitrarily change a running program's functionality through a buffer overflow attack. It is strongly suggested that the fgets() function be used in all cases.*

# Unsafe String Functions & Security

- *Question*: How can an attacker exploit a buffer overflow to break a system's security?
- *Answer*: Check out this step by step tutorial
- *Real example*: Recent WhatsApp vulnerability
- *Lessons*:
  - Only use C when necessary, and be mindful of safe programming practices
  - Otherwise, be responsible and use the right language for your task

```
strcpy()
strcat()
strlen()
sprintf()
gets()
```

```
strncpy()
strncat()
strnlen()
snprintf()
fgets()
```

How can we safely copy a string?

Option 1:

```
strcpy(dest, src);
```

Option 2:

```
#define MAXS 100
char dest[100];
strncpy(dest, src, MAXS);
```

Option 3:

```
#define MAXS 100
char dest[100];
strncpy(dest, src, MAXS - 1);
```

Option 4:

```
#define MAXS 100
char dest[100]; dest[MAXS-1] = '\0';
strncat(dest, src, MAXS - 1);
```

On the following 3 slides, which code snippets can be considered unsafe.

# Is this a *memory-safe* code snippet (1/3).

```
char str1[20] = "BeginnersBook";
printf("Length of string str1 %d\n", strnlen(str1, 20));
printf("Length of string str1 %d\n", strnlen(str1, 10));
```

# Answer: Is this a *memory-safe* code snippet (1/3).

```
char str1[20] = "BeginnersBook";
printf("Length of string str1 %d\n", strnlen(str1, 20));
printf("Length of string str1 %d\n", strnlen(str1, 10));
```

Yes

# Is this a *memory-safe* code snippet (2/3).

```c
char str1[6] = "csc209";
// some other code here.
int b;
scanf("%d", &b);
printf("Length of string str1 %d\n", strnlen(str1, b));
```

# Answer: Is this a *memory-safe* code snippet (2/3).

```c
char str1[6] = "csc209"; // not enough space allocated
// some other code here.
int b;
scanf("%d", &b); // never trust external input
printf("Length of string str1 %d\n", strnlen(str1, b));
```

# Is this a *memory-safe* code snippet (3/3).

```
char buf[209] = {'\0'};
printf("Enter your name and press <Enter>\n");
gets(buf);
```

# Answer: Is this a *memory-safe* code snippet (3/3).

```
char buf[209] = {'\0'};
printf("Enter your name and press <Enter>\n");
gets(buf); // Accepts infinite stream
```

# Extras

# Address Sanitizer Uses

- Use after free (dangling pointer dereference)
- Heap buffer overflow
- Stack buffer overflow
- Global buffer overflow
- Use after return
- Use after scope
- Initialization order bugs
- Memory leaks

Detailed list of memory errors that Address Sanitizer can detect

# Challenge (New tool, for those interested)

Fuzzing

- Random, unexpected inputs
- Try testing with American Fuzzy Lop

Don't use Address Sanitizer and AFL together (too much RAM)