“Advanced C Programming”

Coding Practices

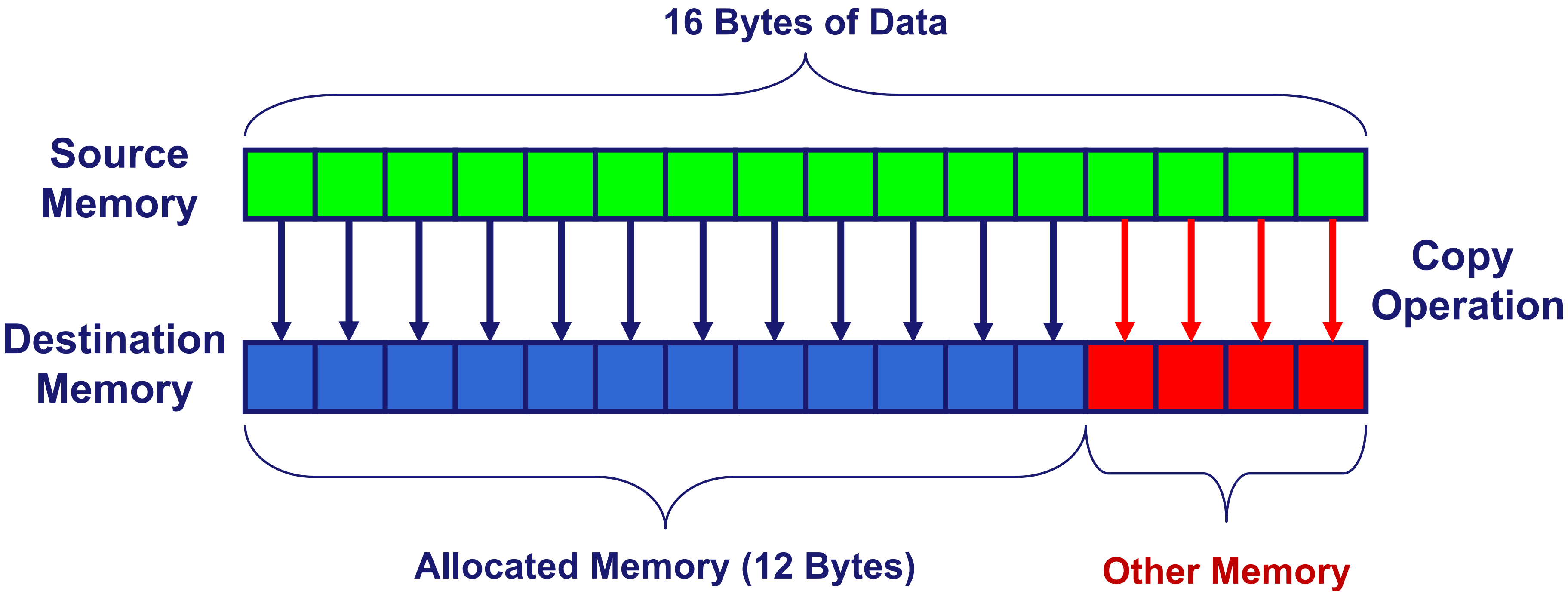
Introduction

- An insecure program may corrupt the data without the intervention of any third person
  - **System Crash, Program Crash, Unexpected results etc..**

Buffer Overflows

Definition: A buffer overflow occurs when a program attempts to write data past the end (or) before the beginning of a buffer.

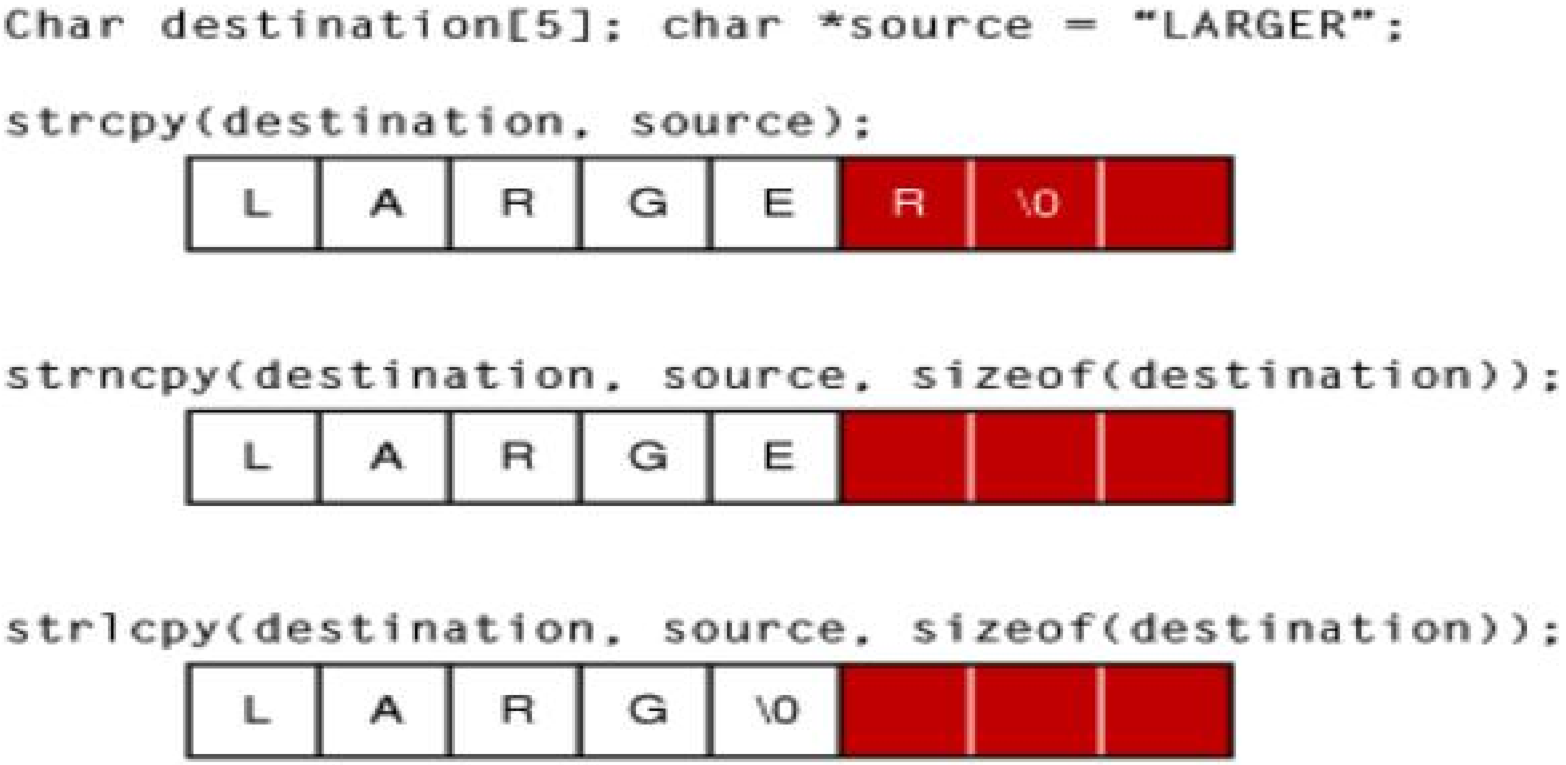
- If the input is **longer than the allocated memory** for it then the data will **"overwrite"** other data in memory.
- **Buffer overflows** – **Stack Overflows, Heap Overflows, String overflows**



String Overflows

- Many string-handling functions have no built-in checks for string length.
- strings are frequently the source of exploitable buffer overflows.

Three string copy functions handle the same over-length string



```
Char destination[5]; char *source = “LARGER”;

strcpy(destination, source);
```

L	A	R	G	E	R	\0	

```
strncpy(destination, source, sizeof(destination));
```

L	A	R	G	E			

```
strlcpy(destination, source, sizeof(destination));
```

L	A	R	G	\0			

String Overflows…

Three string copy functions handle the same over-length string

- **Strcpy** function merely writes the entire string into memory, overwriting whatever came after it – **Not Safe**

- **Strncpy** function truncates the string to the correct length, but without the terminating null character – **Not safe**

- **Strlcpy** function truncating the string to one byte smaller than the buffer size and adding the terminating null character – **Fully safe**

Variable scopes

Do not reuse variable names in a single scope

Example:

```
char msg[100];
void hello_message()
{
        char msg[80] = "Hello";
        strcpy(msg, "Error");
}
```

Msg[100] and msg[80] declarations are in the same scope.

Multi Variable declarations

Take proper care when we declare multiple variables in a single line

Example:

```
char* s1=0, s2=0;
```

It is equal to

```
char *s1=0;
char s2=0;
```

But, our intension may be,

```
char *s1=0;
char *s2=0;
```

Precedence

Use parentheses to define precedence

Example:

x & 1==0

↓

x & (1==0) → x & 0 → Always zero

Solution:

(x & 1) == 0 → Checks the least significant bit of x

sizeof

Operands to the sizeof operator should not contain side effects

Example:

```
int a = 14;
int b = sizeof(a++);
```

The expression a++ will not be evaluated here.

Solution:

```
int a = 14;
int b = sizeof(a);
a++;
```

Enum

Ensure enum constants map to unique values

Example:

```
enum {red=4, orange, yellow, green, blue, indigo=6, violet};
```

Problem:

yellow and indigo have same values

Solution:

Do not do arbitrary assignments in enum.

```
enum {red, orange, yellow, green, blue, indigo, violet};
```

Integer Arithmetic

Ensure that integer arithmetic will not cause overflow

Example:

```
unsigned int a, b, c;
c = a + b;
```

Problem:

a + b may not fit in c.

Solution:

Do error handling to check whether a+b fits in c or not.

Memory deallocation

Set pointers to dynamically allocated memory to NULL after they are released – It avoids double free vulnerability.

Example:

```
if (message_type == value_1) {
/* Process message type 1 */
free(message);
}
/* ...*/
if (message_type == value_2) {
/* Process message type 2 */
free(message);
}
```

Problem:

We are trying to deallocate "message" twice. It causes double free vulnerability.

Memory deallocation…

Set pointers to dynamically allocated memory to NULL after they are released – It avoids double free vulnerability.

Solution:

```
if (message_type == value_1) {
/* Process message type 1 */
free(message);
message = NULL;
}
/* ...*/
if (message_type == value_2) {
/* Process message type 2 */
free(message);
message = NULL;
}
```

Assign NULL to message after free(). We can call free() on a NULL pointer. It will do nothing.

Few more….

- Do not access freed memory
- Free dynamically allocated memory only once
- Detect and handle critical memory allocations

Few more….

- Use logical variable names to avoid any confusion.
- Piling up everything into the main function is absurd. Functions in C helps you to overcome this problem plus it reduces the code redundancy.
- Make use of the switch statement instead of making complications nested if-statements.
- Never leave pointers uninitialized. It may point to some random memory locations and may cause the system to crash.

Thank You