Buffer Overflow

CWE-120: Buffer Copy without Checking Size of Input ('Classic Buffer Overflow')

FURB
UNIVERSIDADE DE BLUMENAU

Definição

- Algumas linguagens de programação esperam que o programador seja responsável pela integridade dos dados na memória.
- Nestas linguagens, não existe verificação na atribuição de valores às variáveis. Isto é, o programa não confere se o valor armazenado na variável cabe no espaço de memória alocado.
- Isso torna os programas mais rápidos de serem executados, já que o código compilado não precisa conferir se a manipulação é integra a cada acesso às variáveis.

Exemplo

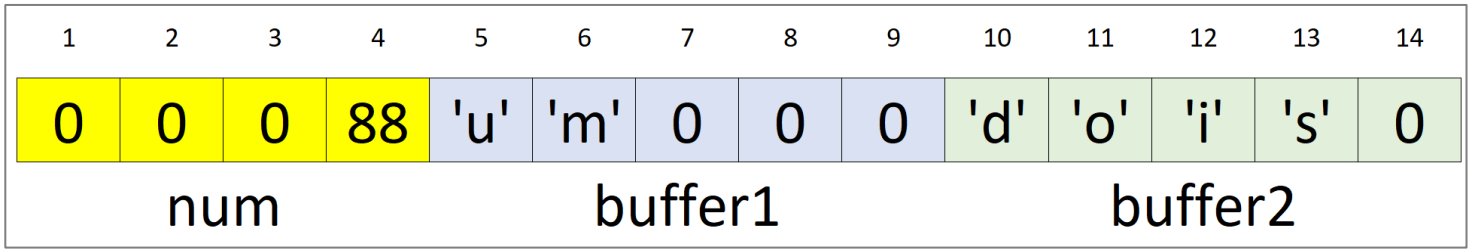
```c
#include <stdio.h>
#include <string.h>
#include <conio.h>

int main (int argc, char*argv[]) {
    int numero = 88;
    char buffer1[5];
    char buffer2[5];

    strcpy(buffer1, "um");
    strcpy(buffer2, "dois");

    printf("[ANTES ] buffer2 contém '%s'\n", buffer2);
    printf("[ANTES ] buffer1 contém '%s'\n", buffer1);
    printf("[ANTES ] numero  contém '%d'\n", numero);

    strcpy(buffer1, argv[1]);

    printf("[DEPOIS] buffer2 contém '%s'\n", buffer2);
    printf("[DEPOIS] buffer1 contém '%s'\n", buffer1);
    printf("[DEPOIS] numero  contém '%d'\n", numero);
}
```

```
[ANTES ] buffer2 contém 'dois'
[ANTES ] buffer1 contém 'um'
[ANTES ] num     contém 88
[DEPOIS] buffer2 contém 'dois'
[DEPOIS] buffer1 contém 'ABC'
[DEPOIS] num     contém 88
```

Saída

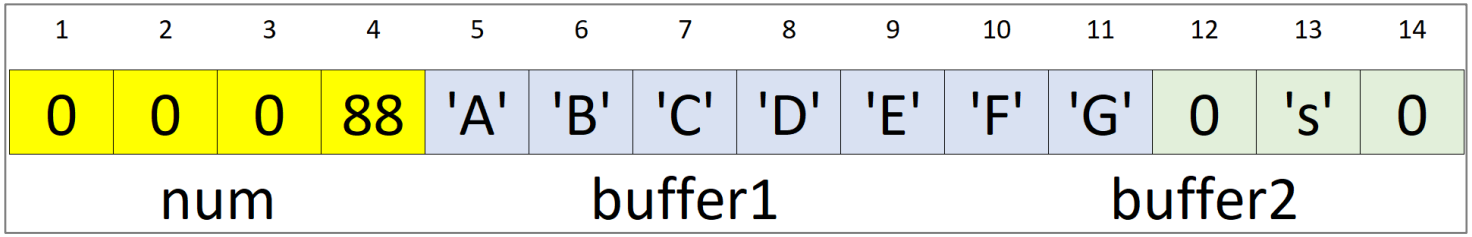
Exemplo

Supor que o programa tenha sido executado com o argumento: ABCDEFG




```
[ANTES ] buffer2 contém 'dois'
[ANTES ] buffer1 contém 'um'
[ANTES ] num     contém 88
```

```
strcpy(buffer1, argv[1]);
```




```
[DEPOIS] buffer2 contém 'FG'
[DEPOIS] buffer1 contém 'ABCDEFG'
[DEPOIS] num     contém 88
```

Buffer overflow

- Quando o valor atribuído a uma variável contém um tamanho superior à área reservada para a variável, o valor irá ocupar o espaço consecutivo. Este problema é conhecido como *buffer overflow.*
  - Trata-se de uma situação onde um programa em execução tenta escrever dados na memória além da área previamente alocada.
- Consequências:
  - Mudança de comportamento das aplicações
  - Se o tamanho do valor for superior à área de dados do programa, o programa irá interromper

Buffer overflow

- O programa deveria fazer uma verificação de tamanho e restringir a entrada de usuário
- *Buffer overflow* ocorre em linguagens de programação em que o programador precisa explicitamente alocar memória para o programa.
- As linguagens mais sujeitas a este problema são C e C++.

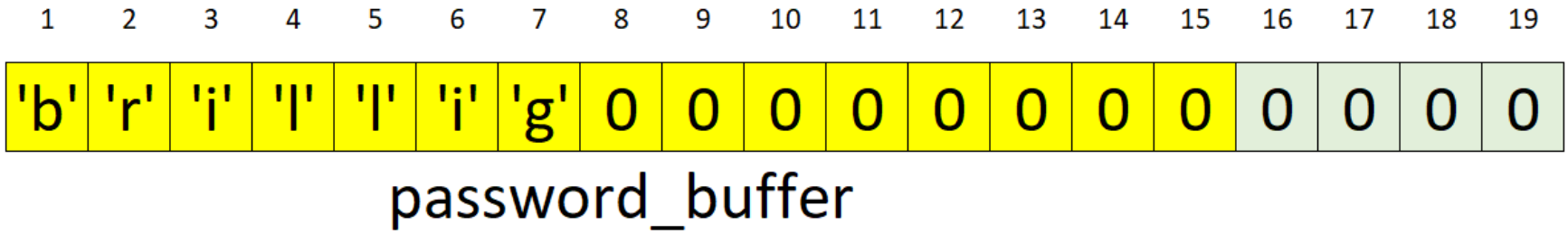
Exemplo 2

```
int main(int argc, char *argv[]) {
    if (argc < 2) {
        printf("Usage: %s <password>\n", argv[0]);
        exit(0);
    }

    if (check_authentication(argv[1])) {
        printf("\n-=-=-=-=-=-=-=-=-=-=-=-=-\n");
        printf(" Access Granted.\n");
        printf("-=-=-=-=-=-=-=-=-=-=-=-=-\n");
    } else {
        printf("\nAccess Denied.\n");
        exit(1);
    }
    ...
}
```

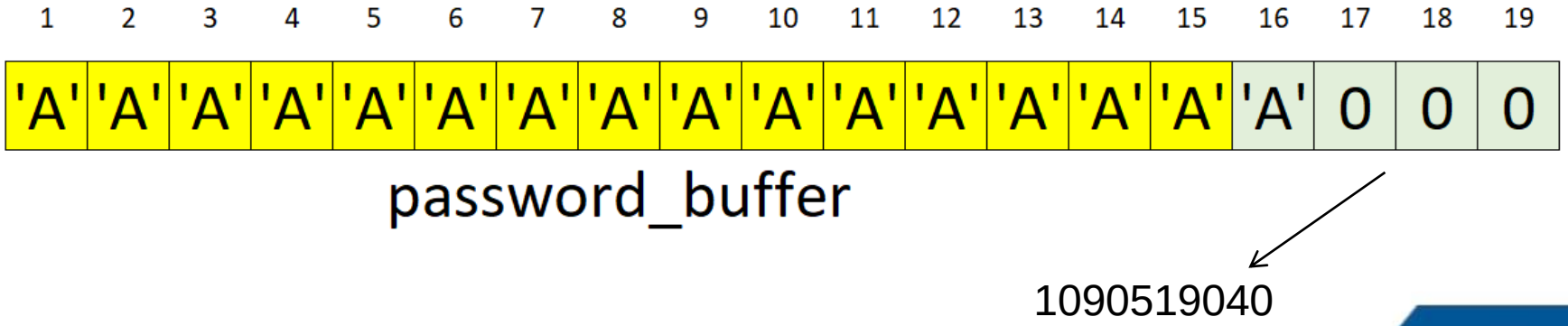
```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>

int check_authentication(char *password) {
    char password_buffer[16];
    int auth_flag = 0;
    strcpy(password_buffer, password);
    if (strcmp(password_buffer, "brillig") == 0)
        auth_flag = 1;
    if (strcmp(password_buffer, "outgrabe") == 0)
        auth_flag = 1;
    return auth_flag;
}
```

Exemplo 2

Ao submeter a entrada “brillig”:



Ao submeter a entrada “AAAAAAAAAAAAAAAA” (16 ‘A’):



Exemplo 2

- Execução:

```
auth_overflow AAAAAAAAAAAAAAAA
```

- Saída:

```
-=-=-=-=-=-=-=-=-=-=-=-=-=-=-
 Access Granted.
-=-=-=-=-=-=-=-=-=-=-=-=-=-=-
```

- Entrada:

```
auth_overflow AAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
```

- Saída:

```
Segment fault in address 0x0800412a
```

Como mitigar

- Algumas funções de bibliotecas reutilizáveis não são intrinsecamente seguras, tal como ocorre com a função `strcpy, scanf, etc.`

- A prática segura é, utilizar sempre funções intrinsecamente seguras.