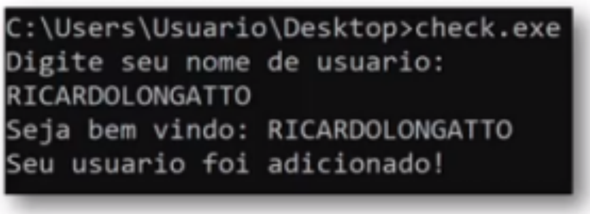
Buffer Overflow

~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~

Exemplo Didático

~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~

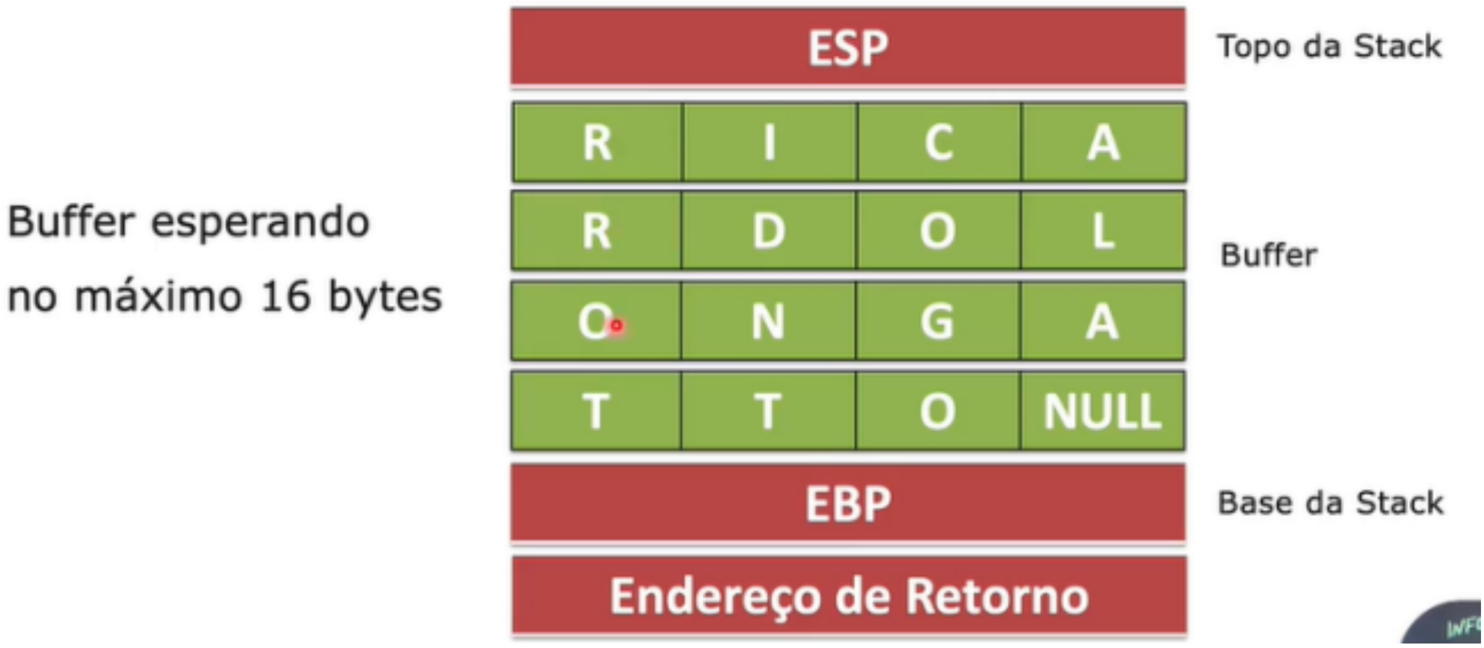
O desenvolverdor precisa armazenar o nome do usuário, então ele define uma variável nome com um buffer de 16 bytes. (pois ele sabe que os nomes de usuário do seu sistema não passam de 16)



Nome = um buffer de 16 bytes

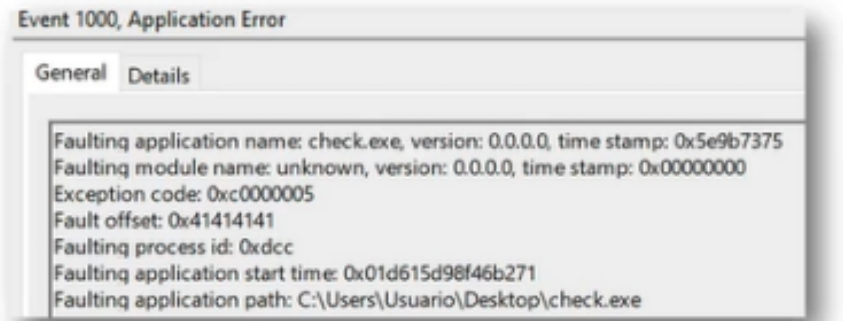
Alguns exemplos de funções em linguagem C
gets - Pega o valor inserido pelo usuário e coloca no buffer (nome)
strcpy - Copia o valor inserido pelo usuário via argumento no buffer (nome)
scanf - Recebe um valor inserido pelo usuário e coloca no buffer (nome)

Nossa stack ficaria similar ao exemplo abaixo:



E se o programa recerber mais dados do que ele espera e não tratar corretamente a entrada de dados?

```
C:\Users\Usuario\Desktop>check.exe
Digite seu nome de usuario:
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
Seja bem vindo: AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
Seu usuario foi adicionado!
```



→ Agora acontece a condição do Buffer Overflow

~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~

Buffer Overflow

~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~

E se o usuário enviar mais dados do que o buffer espera?

EIP - 41414141
Próximo endereço a ser chamado 41414141 = erro
O buffer está em verde e o overflow em vermelho
→ Pra que isso não ocorra, ou o desenvolvedor deve usar funções mais seguras, ou deve fazer uma validação da informação enviada pelo usuário

+ Se conseguirmos controlar o **EIP** conseguimos controlar o fluxo de execução do programa, desta forma, podemos inserir um ***shellcode*** e direcionar o fluxo de execução do programa para o nosso shellcode.

+ Etapas até o shellcode:

→ Descobrir se o programa é vulnerável a buffer overflow
→ Encontrar a quantidade correta de bytes para atingir o EIP
→ Testar o controle do EIP
→ Testar bad characters (bad chars)
→ Encontrar um bom endereço de retorno
→ Gerar um shellcode
→ Ganhar Shell