Assembly do IA-32 em ambiente Linux

TPC9 e Guião laboratorial

Alberto José Proença & Luís Paulo Santos

Objetivo

A lista de exercícios/tarefas propostos no TPC9 / Guião laboratorial, para execução no servidor, reforça a análise laboratorial (e a ferramenta associada, o depurador `gdb`) referente ao conjunto de **instruções e técnicas para suporte à invocação e execução de funções em C**.
Os exercícios para serem resolvidos antes da aula TP estão assinalados com uma caixa cinza.

Buffer overflow

1. O seguinte código C mostra uma implementação (de baixa qualidade) de uma função que lê uma linha da *standard input*, copia a *string* lida para uma novo local de memória, e devolve um apontador para o resultado.

```
1 /*  Isto e' codigo de qualidade questionavel.
2     Tem como objetivo ilustrar tecnicas deficientes de programacao. */
3 char *getline()
4 {
5     char buf[8];
6     char *result;
7     gets(buf);
8     result = malloc(strlen(buf));
9     strcpy(result, buf);
10    return(result);
11 }
```

a) (A) **Construa** um `main` simples que invoque a função `getline` e compile-o sem qualquer otimização, i.e., com `-O0`; confirme que o programa executável "desmontado" (*disassembled*) até à chamada da função `gets` é semelhante a:

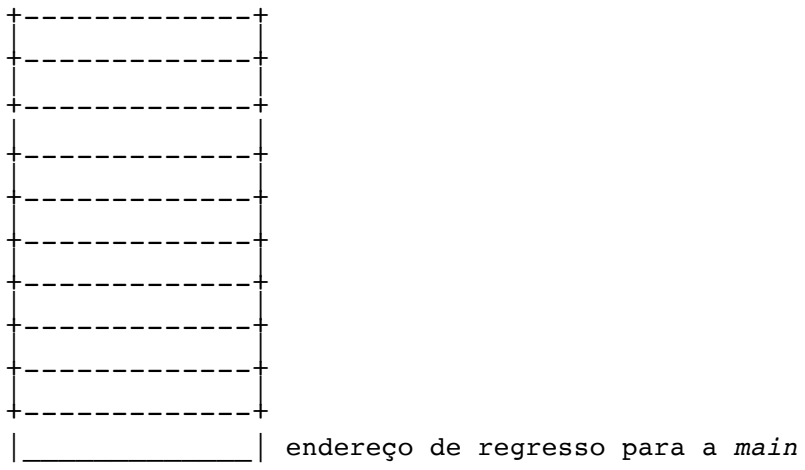
```
1    8048474 <getline+0>:          push  %ebp
2    8048475 <getline+1>:          mov   %esp,%ebp
3    8048477 <getline+3>:          sub   $0x18,%esp
4    804847a <getline+6>:          sub   $0xc,%esp
5    804847d <getline+9>:          lea   -0x8(%ebp),%eax
6    8048480 <getline+12>:         push  %eax
7    8048481 <getline+13>:         call  8048360 <gets@plt>   Invoca gets
```

b) (A) Execute o programa, introduza uma *string* suficientemente longa (por exemplo, `1, 2, 3, 4, 5, 6, 7, 8, 9, 0, 1, 2`) e confirme que o programa termina anormalmente.

Pretende-se ao longo deste Guião laboratorial detetar o local onde ocorreu a anomalia na execução do programa, com o auxílio de um depurador.

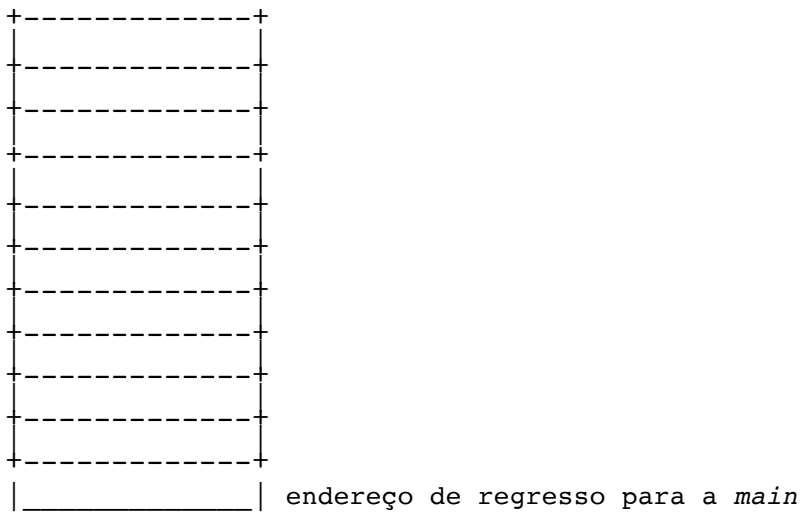
Dica: deverá concluir que tal aconteceu na execução da instrução `ret` da função `getline`.

c) **(A/R)** Analise a execução do código desmontado na alínea **a)** (função `getline`) até à linha 5. Ao longo da execução, vários valores e registos que definem a *stack frame* associada a esta função serão alterados. Observando esses valores e registos e seguindo as instruções *assembly* permite-nos deduzir os valores que definem e constituem a *stack frame*. **Preencha** o seguinte diagrama da *stack frame* que é gerado até este ponto da execução, com a estimativa desses valores e endereços. **Coloque** em cada caixa (que representa 4 *bytes*) o respetivo valor em hexadecimal, à esquerda da caixa o endereço mais baixo das 4 células que estão representadas em cada caixa, e à direita uma etiqueta que descreva o que representa a caixa.
Nota: algumas das caixas (células de memória) podem conter valores arbitrários.



Confirme agora a *stack frame* que construiu, colocando um *breakpoint* na linha 5 de `getline` e executando o programa. Indique a posição de `%ebp`. **Confirme** que o endereço de regresso está correto, examinando o código da função `main()`.

d) **(R)** **Preencha** o diagrama seguinte relativo à *stack frame* de `getline`, após a execução da função `gets`, usando a *string* de 12 carateres sugerida na alínea **b)**.



e) **(R)** **Identifique** as células de memória da *stack frame* que foram alteradas após executar a função `gets`. **Descreva** detalhadamente o impacto destas alterações na restante execução do programa.

f) **(R)** **Identifique** o(s) registo(s) que foi(oram) corrompido(s) no regresso da função `getline` e **mostre** como foram modificados.

g) **(B)** Para além do problema de *buffer overflow*, que duas outras coisas estão erradas no código de `getline`?