Assembly do IA-32 em ambiente Linux

TPC8 e Guião laboratorial

Alberto José Proença & Luís Paulo Santos

Objetivo

A lista de exercícios/tarefas propostos no TPC8 / Guião laboratorial, <u>para execução no servidor</u>, 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.

```
Isto e' codigo de qualidade questionavel.
      Tem como objetivo ilustrar tecnicas deficientes de programação. */
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 -00; 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>:
                                       %esp, %ebp
                                 mov
3
    8048477 <getline+3>:
                                       $0x18,%esp
                                 sub
    804847a <getline+6>:
                                sub
                                       $0xc, %esp
5
    804847d <getline+9>:
                                 lea
                                       -0x8(%ebp), %eax
6
    8048480 <getline+12>:
                                push %eax
    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.
```

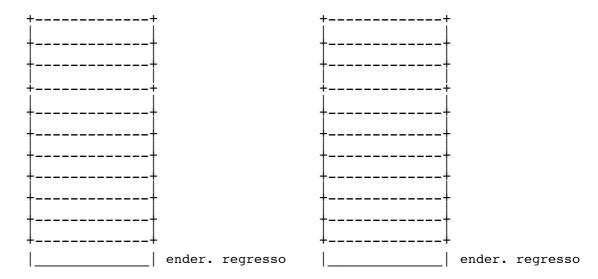
Pretende-se ao longo deste Guião laboratorial detetar o local onde ocorreu a anomalia na execução do programa, com o auxílio de um depurador.

<u>Dica</u>: deverá chegar à conclusão que tal aconteceu na execução da instrução ret da função getline.

c) (A/R) Considerando que a *stack* "cresce para cima", **preencha o diagrama da** *stack frame* com o máximo de indicações, logo após execução da instrução da linha 5 (no código desmontado em cima).

Coloque, no diagrama da **esquerda** <u>em cada caixa</u> (que representa 4 *bytes*) o respectivo valor em hexadecimal (se conhecido), <u>à esquerda</u> o endereço mais baixo das 4 células que estão representadas em cada caixa, e <u>à direita</u> uma etiqueta que ajude a esclarecer o conteúdo da *stack* (por ex., "ender. regresso").

Confirme agora a stack frame que construiu, colocando breakpoints em locais apropriados (antes de gets) 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 da direita para mostrar os valores expectáveis após a invocação da função gets (linha 8), e depois confirme esses valores (correndo o programa).
- e) (R) Para que endereço acha que o programa está a tentar regressar?

 Resp.: _____
 Confirme a sua previsão.
- f) (R) Que registo(s) acha que foi(oram) corrompido(s) no regresso da função getline e como? Confirme a sua previsão.
- g) ^(B) Para além do problema de *buffer overflow*, que duas outras coisas estão erradas no código de getline?