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, 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**.
Não esquecer que estes trabalhos experimentais deverão ser realizados no servidor Unix de SC, à semelhança dos trabalhos anteriores.

Responda manualmente às questões na folha fornecida para o efeito e entregue a resolução até à hora de início da sessão PL seguinte, com a presença do estudante durante a sessão PL. Não serão aceites trabalhos entregues depois deste prazo.

Buffer overflow

1. **(A)** 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 um novo local de memória, e devolve um apontador para o resultado.

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

2. **(A)** **Construa** um `main` simples que invoque a função `getline` e compile-o sem qualquer otimização, i.e., com `-O0`; confirme que o código executável "desmontado" (*disassembled*) da função `getline` 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   0xfffffff8(%ebp),%eax ; nota: 0xf...f8 = -8
6   8048480 <getline+12>:    push  %eax
7   8048481 <getline+13>:    call  8048360 <gets@plt>    ; Invoca gets
```

3. **(A)** Execute o programa introduzindo uma *string* suficientemente longa (por exemplo, `123456789012`) 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.

4. (A/R) Analise a execução do código desmontado no exercício **2** (função `getline`) até à linha 5. Ao longo da execução, vários valores em registos e no quadro desta função (*stack frame*) serão alterados.
   Observando esses valores e seguindo as instruções em *assembly,* permite-nos deduzir os valores que definem e constituem a *stack frame*.
   Preencha o diagrama da *stack frame* que é gerado até este ponto da execução, com a estimativa desses valores e endereços.

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

6. (R) **Preencha** o diagrama relativo à *stack frame* de `getline`, após a execução da função `gets`, usando a *string* de 12 carateres sugerida no exercício **3**.

7. (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.

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

9. (R) **Identifique e caracterize** os problemas associados à utilização da função `gets`.

10. (B) Considere uma implementação (ainda pior!) da função `getline`.
    Indique que tipo de erros adicionais esta função poderá originar.

```
1 char *getline()
2 {
3     char buf[8];
4     gets(buf);
10    return(buf);
11 }
```

11. (B) **Desenvolva** uma versão que implemente a função de forma segura.
    Sugestão: use uma função de *input* que limite o número de caracteres que podem ser lidos, tal como `fgets()` ou `getline()`.

Sugestão para mais informação sobre *buffer overflows*:
https://en.wikipedia.org/wiki/Buffer_overflow

Nº	Nome:	Turma:

Resolução dos exercícios

Nota: as questões 1 e 2 são para ser resolvidas como TPC, as restantes na aula

1. Código C de um `main` simples que invoque a função `getline`

Copie para aqui o código C de um `main` simples que colocou no servidor remoto (para invocar a função `getline`).

2. Análise do código desmontado

Compile o código C sem qualquer otimização (com `-O0`) e **copie** para aqui o código executável "desmontado" (*disassembled*) da função `getline` até à chamada da função `gets`, mostrando (com um *print screen* ou foto do monitor) todos os comandos que usou para compilar e para ter o código desmontado da função.
Anote detalhadamente o código desmontado, ignorando as fases de arranque e término da função.

3. Execução do código

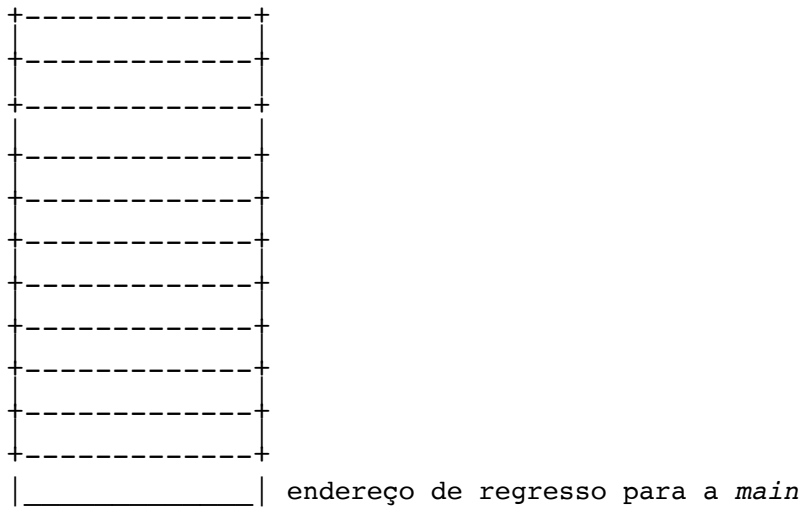
Replique aqui tudo que apareceu no monitor assim que mandou executar o código (incluindo os caracteres que tiver introduzido e o resultado da execução do código).

4. Estimando o quadro da função `getline` na *stack*

A anomalia que constatou poderá ser devida a um erro na função `getline` que provocou a alteração indevida de informação armazenada na *stack*. Isso pode conduzir a que a execução do código tenha tentado aceder a uma zona de memória que não faz parte da área de memória que estava alocada a este programa. Tal pode acontecer no regresso de uma função, se o valor do endereço de regresso (que está na *stack*) tiver sido indevidamente modificado.
Para verificar se foi isto que aconteceu, temos de analisar o quadro da função `getline`.

Preencha o diagrama do quadro da função `getline` (a sua *stack frame*) que é gerado até este ponto da execução (até à linha 5), com a estimativa dos seus valores e endereços, procedendo assim:

- **a. Indique** à esquerda das caixas (cada caixa representa 4 células de memória) a posição apontada pelo registo `%esp` e pelo registo `%ebp`, bem como outras posições relevantes (nesse caso, utilize posições relativas ao registo `%ebp`, e.g., `%ebp+4`, etc.);
- **b. coloque** à direita de cada caixa uma etiqueta que descreva o que representa a caixa (nota: algumas caixas podem conter valores irrelevantes);
- **c. coloque** dentro da caixa correspondente o endereço de regresso para a `main` (nota: consulte o código desmontado da função `main`)



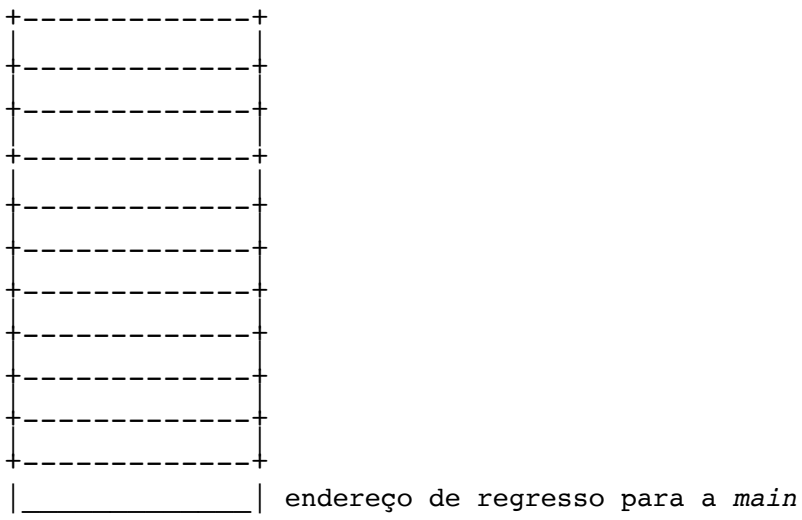
5. Confirmação de valores do quadro da função `getline` na *stack*

Vamos confirmar a *stack frame* que construiu, colocando um *breakpoint* na linha 5 de `getline` e executando o programa.
Quando este parar no *breakpoint*, veja os valores dos registos relevantes para a *stack frame* (`%ebp` e `%esp`), bem como o conteúdo da posição da *stack* onde está armazenado o endereço de regresso.

Coloque os comandos/resultados que utilizou para obter esses 3 valores (apresente da mesma maneira que apareceu no seu monitor).

6. Nova análise do quadro da função `getline` na *stack*

Preencha o diagrama seguinte relativo à *stack frame* de `getline`, estimando os valores dos conteúdos das caixas, após a execução da função `gets`, usando a *string* de 12 carateres sugerida no exercício **3**.



7. (e 8.) Explicação da alteração do quadro da função `getline` na *stack*

Identifique no diagrama em cima 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.
Identifique o(s) registo(s) que foi(oram) corrompido(s) no regresso da função `getline` e mostre como foram modificados.
Identifique e caracterize os problemas associados à utilização da função `gets`.