TPC9 e Guião Laboratorial

Resolução

Pretende-se com esta sessão laboratorial recuperar a informação perdida num ficheiro que continha o código *assembly* da função `int soma_grandes (int n, int *a)` escrevendo, em *assembly*, a parte que foi "danificada".

O código "danificado" desta função percorria os `n` primeiros elementos do vetor de inteiros `a` e adicionava todos os maiores que 1000; terminava devolvendo o valor dessa adição.

Para recuperar este código em *assembly* sugeriu-se a seguinte metodologia:

- escrever um possível algoritmo da função numa notação/linguagem de alto nível;
- preencher a tabela de alocação de registos a argumentos, variáveis locais e eventuais variáveis de caráter mais temporário, que seja necessário armazenar;
- desenhar o quadro de ativação (*stack frame*) da função;
- escrever o código que substitui a parte "danificada";
- validar o código criado comparando-o com a compilação para *assembly* do algoritmo em cima.

1. Um possível algoritmo

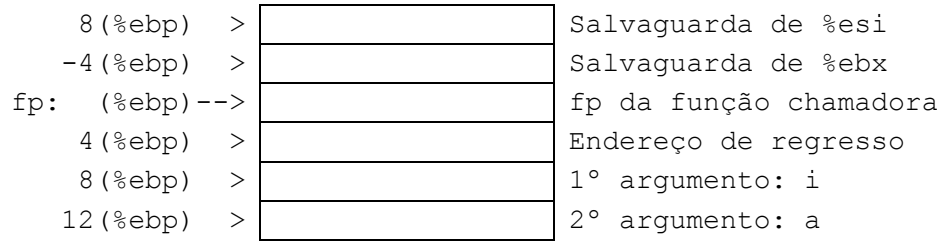
```
int soma_grandes (int n, int *a) {
  int i, r=0;

  for (i=0 ; i<n ; i++) {
    if (a[i] > 1000) r += a[i];
  }
  return r;
}
```

2.a) Alocação de registos

Variável	Reg	Obs.
`r`	`%eax`	Como se trata do valor a devolver pela função, e como o mecanismo utilizado para o fazer é o registo %eax, usa-se desde logo este registo
`i`	`%ecx`	Escolha arbitrária
`a`	`%ebx`	Escolha arbitrária (NOTA: registo *callee saved*)
`n`	`%esi`	Escolha arbitrária (NOTA: registo *callee saved*)
`a[i]`	`%edx`	Escolha arbitrária

2.b) Quadro de ativação da função (*stack frame*)



Algumas considerações:

- eventuais valores associados a registos *caller saved* não são apresentados porque não há hipótese de saber quais os registos que a função que chama soma_grandes() eventualmente salvaguarda; quanto aos *callee saved* a ordem vai depender do código...
- as variáveis locais serão alocadas a registos e não a posições na *stack*;
- os conteúdos das células poderão ser obtidos com o gdb.

2.c) Um código provável da função

```
soma_grandes:
  pushl  %ebp
  movl   %esp, %ebp
  pushl  %ebx                   # salvaguarda os registos ebx
  pushl  %esi                   #      e esi
  movl   8(%ebp), %esi          # %esi = n (1º arg)
  movl   12(%ebp), %ebx         # %ebx = *a (2ª arg)
  xorl   %eax, %eax             # r = 0
  xorl   %ecx, %ecx             # i = 0
TESTE:
  cmpl   %esi, %ecx
  jge    FIM_CICLO              # se i >= n sai do ciclo for
  movl   (%ebx, %ecx, 4), %edx  # %edx = a[i]
  cmpl   $1000, %edx
  jle    FIM_IF                 # se a[i] <= 1000 salta por cima da adição
  addl   %edx, %eax               # r += a[i]
FIM_IF:
  incl   %ecx                   # i++
  jmp    TESTE                  # repete o ciclo for
FIM_CICLO:
  popl   %esi                   # recupera os registos esi
  popl   %ebx                   #                       e ebx
  leave                         #   e frame pointer da função chamadora
  ret                           # recupera ender regresso e volta à func cham
```

5.
Pretende-se construir um ficheiro executável que devolva a *password* que tem guardada internamente, através da manipulação de uma função *assembly* específica do programa.
Note que o enunciado indica claramente que o uso do `gdb` deverá ser evitado para descobrir a *password* guardada internamente no executável.

Seguindo a sugestão de resolução:

i. **Testar o funcionamento do executável** `hackme`.
Nota: a *password* tem de ser um número inteiro.

```
./hackme
Password: 1234

Log: Authentication output: 0
Log: ...
Log: Incorrect password.
```

Pela análise do *output* do programa, após a sua execução com a *password* 1234 como *input*, é possível identificar já uma pista na mensagem *"Authentication output: 0*": o valor devolvido pela função de autenticação (o *output*) está a ser enviado para o monitor e, neste caso, é "*0"*.
Esta vulnerabilidade do programa será explorada nos pontos seguintes de modo a obter a *password* do programa.

ii. Analisar a estrutura da função de autenticação, no ficheiro `autentica.s`.

```
autentica:
    pushl   %ebp              # fase de arranque da função
    movl    %esp, %ebp #   com criação do novo frame pointer
    movl    8(%ebp), %eax     # leitura do 1º arg (apontador) para %eax
    movl    (%eax), %eax      # valor apontado pelo 1º arg para %eax
    cmpl    %eax, 12(%ebp)    # comparação do 2ºarg com %eax
    sete    %al               # %al= resultado lógico (0/1) da comparação
    movzbl  %al, %eax         # restantes 24 bits de %eax a 0₂ (valor a
                                devolver pela função)
    leave                     # recupera o frame pointer da fun chamad
    ret                       # regressa à função chamadora
```

Pela análise deste código pode-se inferir que este foi gerado por uma estrutura condicional em C, semelhante a:

```
if (*arg1 == arg2)
    return 1;
else
    return 0;
```

iii. Modificar a função de autenticação para que devolva a *password* interna de modo que a função chamadora a possa enviar para o monitor.

Uma possível resolução passa por devolver um dos argumentos passados à função de autenticação. Da análise do ponto anterior sabe-se que um dos argumentos será a *password* introduzida pelo utilizador, e o outro será a *password* interna ao programa.
No entanto, é impossível saber qual dos argumentos corresponde à *password* pretendida.

Versão 1 – devolver o 2º argumento (encontra-se em `%ebp + 12`)

```
autentica:
    pushl   %ebp
    movl    %esp, %ebp
    movl    12(%ebp), %eax
    leave
    ret
```

Versão 2 – devolver o 1º argumento (encontra-se em `%ebp + 8`)

```
autentica:
    pushl   %ebp
    movl    %esp, %ebp
    movl    8(%ebp), %eax
    movl    (%eax), %eax
    leave
    ret
```

De notar na **versão 2** que o valor a ser devolvido, em `%eax`, não é o valor recebido no 1º argumento pois, pela análise do código feita previamente, esse valor é utilizado como endereço para carregar a *password* que é efetivamente usada na comparação. Se fosse devolvido o valor em `8(%ebp)` o programa iria enviar para o monitor o endereço da *password*, e não a *password*.

iv. Criar um novo executável `hacked`.

```
gcc -Wall -O2 main.o autentica_modificado.s -o hacked
```

v. Verificar se a *password* é enviada para o monitor através da execução de `hacked`.

Versão 1 – incorreta

```
                              ./hacked
Password: 1234

Log: Authentication output: 1234
Log: ...
Log: Security breach detected.
```

Versão 2 – correta

```
                              ./hacked
Password: 1234

Log: Authentication output: 1505335290
Log: ...
Log: Security breach detected.
```