

# **Programação de Sistemas Computacionais**

## **Serie 02 de Exercícios**

**Semestre de Verão de 2009/2010**

**Autor:**

31401 – Nuno Cancelo

## **Indície**

Fase 1.....	3
Fase 2.....	4
Fase 3.....	5
Fase 4.....	6
Conclusão.....	9
Bibliografia.....	10

## Fase 1

No decorrer desta fase foram encontradas diversas dificuldades na utilização do programa 'insight' e na interpretação do código assembly inerente a cada fase.

Na análise do pedaço de código da fase um, e após de se ter organizado o ecrã com as janelas mais relevantes do insight (código em modo mixed, memória, registos gerais, stack e breakpoints), verifica-se alguns pontos interessantes.

```

push    ebp
mov     ebp,esp
push    0x804b060
call    0x8048a3c <unscramble>
mov     DWORD PTR [esp],0x804b060
push    DWORD PTR [ebp+0x8]
call    0x80484d0 <strcmp@plt>
add     esp,0x8
test    eax,eax
sete    al
movzx   eax,al
leave
ret

```

<----- A  
 <----- B  
 <----- C  
 <----- D  
 <----- E

No ponto **A** verifica-se o carregamento no stack de uma posição de memória, que é o argumento da função "unscramble" no ponto **B**.

Em **C** e **D** verifica-se o carregamento do endereço de memória (o mesmo da função anterior) e a posição de memória onde está localizada a string introduzida.

Em **E** é chamado a função strcmp, que vai comparar os argumentos passados previamente.

No ponto **C**, ao aceder-mos à posição de memória **0x804b060** podemos verificar qual é a string de comparação para ultrapassar esta fase.

Memory					
Addresses					
Address	0x804b060				Target is LITTLE endian
	0	4	8	C	ASCII
0x0804b060	0x74736f50	0x74736520	0x63636f20	0x6f697361	Post est occasio
0x0804b070	0x6c616320	0x002e6176	0x00000000	0x00000000	calva.....
0x0804b080	0x00000000	0x00000000	0x00000000	0x00000000	.....
0x0804b090	0x00000000	0x00000000	0x00000000	0x00000000	.....
0x0804b0a0	0x00000000	0x00000000	0x00000000	0x00000000	.....

Assim a primeira palavra-passe é:

**Post est occasio calva.**

## Fase 2

Esta segunda fase foi interessante, devido à recursividade utilizada para obtenção do maior dígito introduzido, que se tornou complicado observar.

<pre>Phase2: push    ebp Fase 4:mov    ebp,esp push    0x0 push    0xffffffff push    DWORD PTR [ebp+0x8] call    0x80488a8 &lt;func&gt; add     esp,0xc cmp     eax,DWORD PTR ds:0x804b0e0 sete    al movzx   eax,al leave ret</pre>	
<pre>func: push    ebp mov     ebp,esp push    esi push    ebx mov     esi,DWORD PTR [ebp+0x8] mov     edx,DWORD PTR [ebp+0xc] mov     ecx,DWORD PTR [ebp+0x10] mov     al,BYTE PTR [esi] test    al,al jne     0x80488c4 &lt;func+28&gt; lea     eax,[ecx+ecx*2] lea     eax,[eax+edx*4] jmp     0x80488f6 &lt;func+78&gt; movsx   eax,al lea     ebx,[eax-0x30] sub     eax,0x32 cmp     eax,0x7 ja      0x80488f3 &lt;func+75&gt; lea     eax,[ecx+0x1]</pre>	<pre>push    eax mov     eax,ebx cmp     ebx,edx jge     0x80488de &lt;func+54&gt; mov     eax,edx push    eax lea     eax,[esi+0x1] push    eax call    0x80488a8 &lt;func&gt; add     esp,0xc test    eax,eax js      0x80488f3 &lt;func+75&gt; add     eax,ebx jmp     0x80488f6 &lt;func+78&gt; or      eax,0xffffffff lea     esp,[ebp-0x8] pop     ebx pop     esi leave ret</pre>

Na interpretação deste código verifica-se as seguintes condições:

- é necessário ser dígitos de 2 até 9
- ao maior dígito encontrado é multiplicado por 4
- ao número de dígitos introduzido é multiplicado por 3
- a soma de todos os dígitos introduzidos
- a soma das três parcelas anteriores é igual ao conteúdo da posição de memória **0x804b0e0**. Neste caso é 0x2A.

Sabendo as operações necessárias tornou-se fácil a obtenção da palavra passe.

Ao executar uma tabela exaustiva de hipóteses, verificou-se que com estas condições pode-se obter 9 hipóteses, sendo elas as seguintes:

- 66
- 355 (e respectivas permutações)
- 2444 (e respectivas permutações)
- Fase 4:33333

## Fase 3

Esta fase foi uma das fases mais fáceis de interpretar, apesar de ter estado com alguns problemas na obtenção do resultado. Problemas os quais, posso assumir que será de "excesso de utilização" do debugger, uma vez que me mostrava um valor errado para a obtenção do resultado.

```
push    ebp
mov     ebp,esp
push    edi
push    esi
push    ebx
sub     esp,0x14
mov     edi,DWORD PTR [ebp+0x8]
push    edi
call    0x8048490 <strlen@plt>
mov     ebx,eax
mov     esi,DWORD PTR ds:0x804b0e8
mov     DWORD PTR [esp],esi
call    0x8048490 <strlen@plt>
add     esp,0x4
mov     DWORD PTR [ebp-0x20],eax
lea     eax,[ebp-0x4]
cmp     eax,0xb
ja      0x80487f0 <phase3+144>
movsx   edx,BYTE PTR [edi]
mov     ebx,0x0
mov     ecx,0x0
lea     eax,[edx-0x61]
cmp     eax,0x19
ja      0x80487f0 <phase3+144>
movsx   eax,BYTE PTR [esi+ecx]
sub     edx,eax
mov     eax,edx
js      0x80487b4 <phase3+84>
add     eax,0x61

jmp     0x80487b7 <phase3+87>
add     eax,0x7b
mov     BYTE PTR [ebp+ebx-0x1c],al
inc     ebx
inc     ecx
cmp     ecx,DWORD PTR [ebp-0x20]
jne     0x80487c7 <phase3+103>
mov     ecx,0x0
movsx   edx,BYTE PTR [edi+ebx]
test    edx,edx
jne     0x804879d <phase3+61>
mov     BYTE PTR [ebp+ebx-0x1c],0x0
push    DWORD PTR ds:0x804b0e4
lea     eax,[ebp-0x1c]
push    eax
call    0x80484d0 <strcmp@plt>
add     esp,0x8
test    eax,eax
sete    al
movzx   eax,al
jmp     0x80487f5 <phase3+149>
mov     eax,0x0
lea     esp,[ebp-0xc]
pop     ebx
pop     esi
pop     edi
leave
ret
```

Neste exercício, existem dois endereços de memória importantes. O endereço **0x804b0e8**, que contém a palavra "fogo" que servirá "descodificar" o índice da letra da palavra e o endereço **0x804b0e4** que contém o ponteiro para a palavra a comparar (0x0804919c → "aprovado").

Nesta codificação é realizada as seguintes operações:

- ao 1º e 5º carácter é retirado o valor da letra 'f'
- ao 2º e 6º carácter é retirado o valor da letra 'o'
- ao 3º e 7º carácter é retirado o valor da letra 'g'
- ao 4º e 8º carácter é retirado o valor da letra 'o'
- se o resultado for negativo soma-se o valor 0x7b
- se o resultado for positivo soma-se o valor 0x61

Sabendo estas operações procedeu-se à realização de uma tabela para obter todos os resultados e posteriormente filtrou-se os resultados de forma a obter no fim a palavra 'aprovado'.

Desta forma chegou-se ao resultado: **fdxcaojc**

## Fase 4

Esta última fase, demorou um pouco mais pois havia um bug no programa, pois era necessário três operações distintas e só estavam atribuídas duas, sendo que uma estava repetida.

```
Phase4
push    ebp
mov     ebp,esp
push    edi
push    esi
push    ebx
sub     esp,0x24
mov     DWORD PTR [ebp-0x10],0x7
lea     eax,[ebp-0x10]
push    eax
lea     eax,[ebp-0x2c]
push    eax
push    DWORD PTR [ebp+0x8]
call    0x80489ec <readNIntegers>
add     esp,0xc
cmp     DWORD PTR [ebp-0x10],0x0
jne     0x804889d <phase4+157>
cmp     BYTE PTR [eax],0x0
jne     0x804889d <phase4+157>
mov     ebx,DWORD PTR ds:0x804b0ec
mov     ecx,DWORD PTR ds:0x804b0f0
mov     edx,DWORD PTR ds:0x804b0f4
lea     esi,[ebp-0x28]
mov     edi,esi
lea     eax,[ebp-0x10]
mov     DWORD PTR [ebp-0x30],eax
mov     eax,DWORD PTR [edi]
cmp     eax,ebx
je      0x8048859 <phase4+89>
cmp     eax,ecx
je      0x8048859 <phase4+89>
cmp     eax,edx
jne     0x804889d <phase4+157>
add     edi,0x8
cmp     edi,DWORD PTR [ebp-0x30]
jne     0x804884b <phase4+75>
mov     eax,DWORD PTR [ebp-0x2c]
mov     ebx,DWORD PTR [esi]
cmp     DWORD PTR [ebx*8+0x804b110],0x0
jne     0x804889d <phase4+157>
push    DWORD PTR [esi+0x4]
push    eax
call    DWORD PTR [ebx*8+0x804b10c]
pop     edx
pop     ecx
mov     DWORD PTR [ebx*8+0x804b110],0x1
add     esi,0x8
cmp     esi,edi
jne     0x8048864 <phase4+100>
cmp     eax,DWORD PTR ds:0x804b0f8
sete    al
movzx   eax,al
jmp     0x804889f <phase4+159>
xor     eax,eax
lea     esp,[ebp-0xc]
pop     ebx
pop     esi
pop     edi
leave
ret
```

No código depressa se chegou à conclusão que seria necessário colocar 7 dígitos separados por espaço(0x20) ou por tab(0x9) e que esses dígitos seriam colocados, cada um deles, em diferentes endereços de memória, guardados desde ebp-0x30 até ebp-0x10, sendo o ebp-10 reservado para o número de dígitos lido.

Verificou-se também que as posições 2,4,6 estavam reservadas para os dígitos das operações, dígitos obtidos pelas posições de memória 0x804b0ec, 0x804b0f0 e 0x804b0f4. O endereço 0x804b0f8 está guardado o resultado final a ser comparado (neste caso o 0x10).

Após esta análise, tratou-se de se averiguar quais seriam as operações que estavam destinadas, pelo que pela análise dos respectivos códigos chega-se à conclusão que:

- Operação 2 é uma divisão
- Operação 3 é uma multiplicação
- Operação 8 é uma exponenciação

Uma vez indentificadas as operações, investigou-se a forma como são efectuados os cálculos:

$$AO_1BO_2CO_3D$$

$$AO_1B=R_1$$

$R_1 O_2 C = R_2$

$R_2 O_3 D = R F$  e por fim  $R_F = 0x10$

Desta forma foi descoberta a última palavra-passe:

1 3 1 2 2 8 4

```
Tem como objectivo descobrir a senha associada a cada uma das 4 fases
Boa sorte!
Senha 1: Post est occasio calva.
OK, esta era simples! Que tal a seguinte?
Senha 2: 2444
Bem conseguido! E agora?
Senha 3: fdxcaojc
Boa! Mais uma?
Senha 4: 1 3 1 2 2 8 4
Parabens! Resolveu todas as fases!
```



## Conclusão

Ao longo desta fase foi possível compreender algumas funcionalidades do assembly, para além de estar apto para entender o funcionamento de cada uma das fases (e respectivos programas).

Não é fácil ler código assembly sem saber o seu propósito, porém não torna impossível alcançar este propósito.

Durante toda a análise, percebeu-se a forma como a generalidade das funções trabalham e a sua utilidade em diversos contextos, para além de se verificar a utilização de outras funções que normalmente não seriam utilizadas devido ao seu desconhecimento.

## Bibliografia

- <http://www.asciitable.com/>
- <http://docs.sun.com/app/docs/doc/806-3773>