

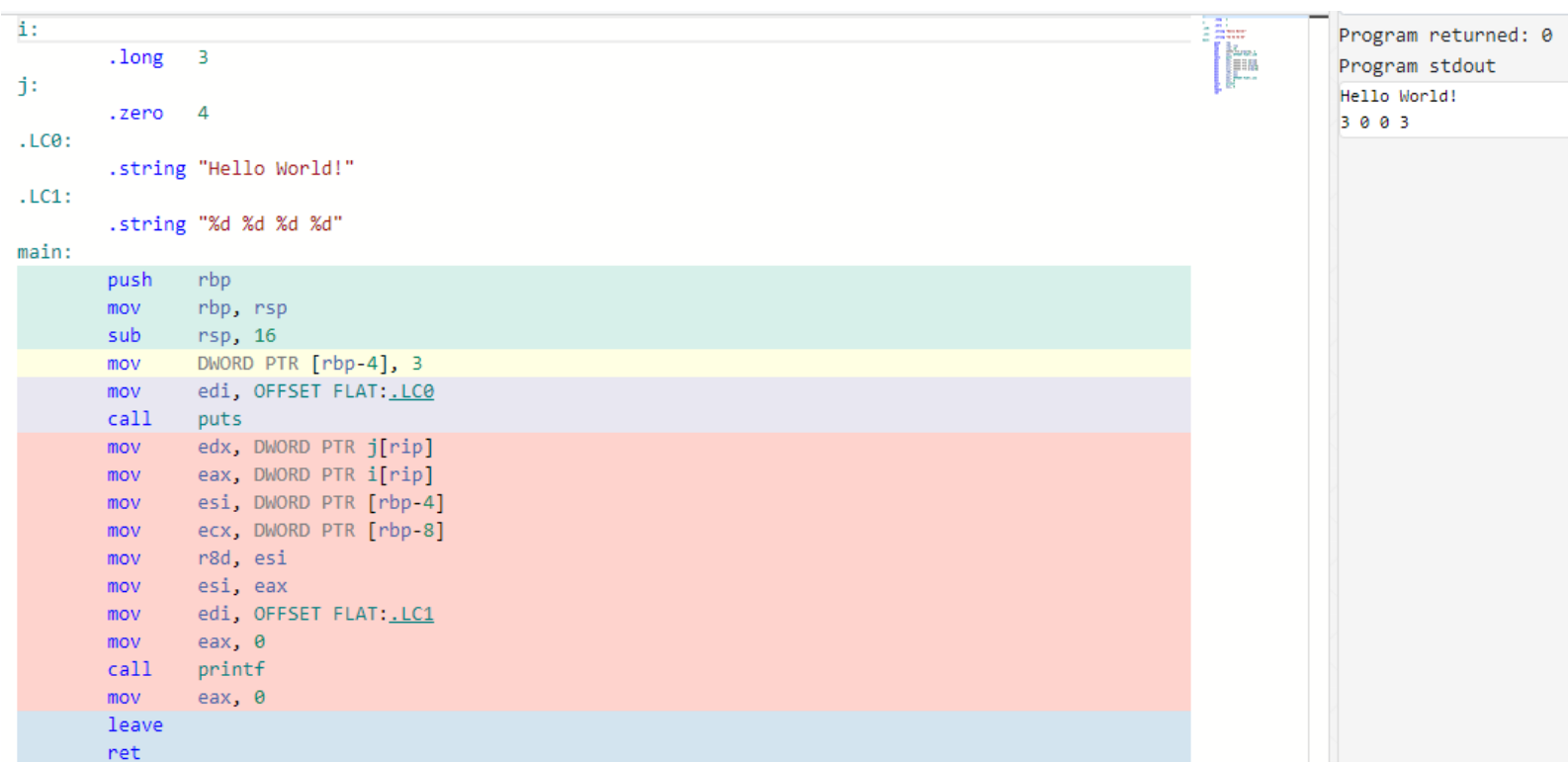
Sistemas Operacionais

Arthur do Prado Labaki – 11821BCC017

Máquina virtual Kali Linux

Atividade Prática – Unidade 2


1- Utilizando o site *godbolt.org*, o código em C pode ser compilado por diversos compiladores. Usando os compiladores x86-64 gcc 11.2 e x86-64 icc 2021.1.2 temos respectivamente:



```
i:
    .long 3
j:
    .zero 4
.LC0:
    .string "Hello World!"
.LC1:
    .string "%d %d %d %d"
main:
    push    rbp
    mov     rbp, rsp
    sub     rsp, 16
    mov     DWORD PTR [rbp-4], 3
    mov     edi, OFFSET FLAT:.LC0
    call    puts
    mov     edx, DWORD PTR j[rip]
    mov     eax, DWORD PTR i[rip]
    mov     esi, DWORD PTR [rbp-4]
    mov     ecx, DWORD PTR [rbp-8]
    mov     r8d, esi
    mov     esi, eax
    mov     edi, OFFSET FLAT:.LC1
    mov     eax, 0
    call    printf
    mov     eax, 0
    leave
    ret
```

Program returned: 0
Program stdout
Hello World!
3 0 0 3

.L_2_STRING.0:		
.long	1819043144	
.long	1867980911	
.long	560229490	
.word	10	
.L_2_STRING.1:		
.long	622879781	
.long	1680154724	
.long	6563104	
main:		
push	rbp	#5.7
mov	rbp, rsp	#5.7
sub	rsp, 32	#5.7
mov	QWORD PTR [-8+rbp], rbx	#5.7[spill]
mov	DWORD PTR [-32+rbp], 3	#7.10
mov	eax, offset flat: .L_2_STRING.0	#8.5
mov	rdi, rax	#8.5
mov	eax, 0	#8.5
call	printf	#8.5
mov	DWORD PTR [-28+rbp], eax	#8.5
mov	eax, offset flat: .L_2_STRING.1	#9.5
mov	edx, DWORD PTR i[rip]	#9.5
mov	ecx, DWORD PTR j[rip]	#9.5
mov	ebx, DWORD PTR [-24+rbp]	#9.5
mov	esi, DWORD PTR [-32+rbp]	#9.5
mov	rdi, rax	#9.5
mov	DWORD PTR [-16+rbp], esi	#9.5[spill]
mov	esi, edx	#9.5
mov	edx, ecx	#9.5
mov	ecx, ebx	#9.5
mov	eax, DWORD PTR [-16+rbp]	#9.5[spill]
mov	r8d, eax	#9.5
mov	eax, 0	#9.5
call	printf	#9.5
mov	DWORD PTR [-20+rbp], eax	#9.5
mov	eax, 0	#10.5
mov	rbx, QWORD PTR [-8+rbp]	#10.5[spill]
leave		#10.5
ret		#10.5
i:		
.long	3	

	Program returned: 0
	Program stdout
	Hello World!
	3 0 4198464 3

Código:

```
#include <stdio.h>

int i=3;
int j;
main(){
    int w;
    int z=3;
    printf("Hello World!\n");
    printf("%d %d %d %d",i,j,w,z);
}
```

Comparando as duas compilações, a diferença vista entre os compiladores é em suas atribuições das variáveis.

No GCC, o `w` é atribuído a pilha no espaço `[rbp-8]`, logo na parte de seu respectivo `printf` (área vermelha). Assim o local do registrador de `w` tem como valor o 0.

Já no o ICC, o `w` que é atribuído a pilha no espaço `[-24+rbp]` não recebe nenhum valor, devido a escolha de variáveis do tipo `long`, tendo como valor do registrador “lixo de memória”.

Em suma, a diferença é que o GCC armazena a variável antes da `main` enquanto o ICC armazena a variável depois da `main`.

2- Para linkar os arquivos `passcode` com o código em `c`, foi necessário utilizar a biblioteca “`gcc-miltilib`”, pois o arquivo é de 32 bits e o sistema operacional usado é de 64 bits.

```
(kali㉿kali)-[~/Desktop/S0/Ativ2]
$ gcc -m32 main.c passcode.o -o main

(kali㉿kali)-[~/Desktop/S0/Ativ2]
$ ./main
ABCDEFGHIIJ
```

Com isso, obtemos a senha que é “`ABCDEFGHIIJ`”.

3- Para conseguir modificar o arquivo executável, primeiro temos que identificar as chamadas das funções e seus endereços em seu código `assembly`. Para isso, foi necessário utilizar o site *onlinedisassembler.com* para fazer o `disassembly`.

Assim foi possível identificar a parte do código onde ocorre as chamadas das funções.

```
141
142 .text:08048228 <main>:
143     .text:08048228 8d 4c 24 04      lea     0x4(%esp),%ecx
144     .text:0804822c 83 e4 f0      and     $0xffffffff0,%esp
145     .text:0804822f ff 71 fc      pushl   -0x4(%ecx)
146     .text:08048232 55      push    %ebp
147     .text:08048233 89 e5      mov     %esp,%ebp
148     .text:08048235 51      push    %ecx
149     .text:08048236 83 ec 04      sub     $0x4,%esp
150     .text:08048239 e8 0e 00 00 00    call    0x0804824c <f1>
151     .text:0804823e e8 21 00 00 00    call    0x08048264 <f2>
152     .text:08048243 83 c4 04      add     $0x4,%esp
153     .text:08048246 59      pop     %ecx
154     .text:08048247 5d      pop     %ebp
155     .text:08048248 8d 61 fc      lea     -0x4(%ecx),%esp
156     .text:0804824b c3      ret
157
158 .text:0804824c <f1>:
159     .text:0804824c 55      push    %ebp
160     .text:0804824d 89 e5      mov     %esp,%ebp
161     .text:0804824f 83 ec 08      sub     $0x8,%esp
162     .text:08048252 83 ec 0c      sub     $0xc,%esp
163     .text:08048255 68 48 2a 0b 08    push    $0x80b2a48
164     .text:0804825a e8 31 0f 00 00    call    0x08049190 <_IO_puts>
165     .text:0804825f 83 c4 10      add     $0x10,%esp
166     .text:08048262 c9      leave
167     .text:08048263 c3      ret
168
169 .text:08048264 <f2>:
170     .text:08048264 55      push    %ebp
171     .text:08048265 89 e5      mov     %esp,%ebp
172     .text:08048267 83 ec 08      sub     $0x8,%esp
173     .text:0804826a 83 ec 0c      sub     $0xc,%esp
174     .text:0804826d 68 4b 2a 0b 08    push    $0x80b2a4b
175     .text:08048272 e8 19 0f 00 00    call    0x08049190 <_IO_puts>
176     .text:08048277 83 c4 10      add     $0x10,%esp
177     .text:0804827a c9      leave
178     .text:0804827b c3      ret
179     .text:0804827c 90      nop
180     .text:0804827d 90      nop
181     .text:0804827e 90      nop
182     .text:0804827f 90      nop
183
```

Para trocar as funções, temos que modificar seu código hexadecimal. O código hex “e8” remete a uma chamada relativa a instrução anterior e tanto o “0e” para f1 ou “21” para f2 refere-se a quantidade de bytes relativos a instrução anterior.

Para isso devemos subtrair o endereço da tag de f2 com o endereço da chamada da função f1 na main (08048264 na linha 169 – 08048239 da linha 150).

Também devemos fazer o mesmo com o endereço da tag de f1 com o endereço da chamada da função f2 na main (0804824c na linha 158 – 0804823e na linha 151).

Com isso temos “2b” e “0e” respectivamente. Porém a chamada da instrução é feita somando 5 bytes, então devemos retirar esses 5 bytes, resultando em “26” e “09”.

Utilizando o site *hexed.it* conseguimos editar o código hex do programa. Alterando o “0e” para “26” e “21” para “09” temos:

```
71 FC 55 89 E5 51 83 EC 04 E8 26 00 00 00 E8 09
00 00 00 83 C4 04 59 5D 8D 61 FC C3 55 89 E5 83
```

```
(kaliⓈkali)-[~/Downloads]
└─$ chmod +x prog01new.exe

(kaliⓈkali)-[~/Downloads]
└─$ ./prog01new.exe
F2
F1
```

4- Utilizando o comando time e anotando o tempo real de execução dos dois programas, temos:

	1	2	3	4	5	6	7	8	9	10	11	Média
Prog2	0,049	0,042	0,041	0,041	0,040	0,039	0,040	0,038	0,039	0,040	0,044	0,040
Prog3	0,020	0,018	0,017	0,017	0,015	0,014	0,018	0,017	0,017	0,017	0,010	0,016

Com isso, percebe-se que o prog3 é mais eficiente que o prog2, visto que, sua média é quase tres vezes menos.

5- Anexado