

Uso de funções e pilha em Assembly

- Vamos investigar o funcionamento de passagem de parâmetros em assembly.
- Vamos usar o Evans Debugger (EDB), Debugador gráfico para ambiente Gnu Linux, para analisar o programa ao lado.
- Para cada execução, usamos F7 (step into), para executar cada linha.
- Observe os valores dos registradores RIP, RBP, RSP. Observe que o resultado da soma de 4 e 5 ficará no acumulador RAX

```
• Seja o seguinte programa:  
• section .text  
•  
• global _start  
•  
• _start:  
•  
• push 4 ;empilha 1o argumento, -8 bytes  
• push 5 ;empilha 2o argumento, -8 bytes  
• call soma ;empilha IP de retorno e chama a função soma  
• mov rcx,rcx  
• ;exit(0)  
• mov eax,1  
• mov ebx,0  
• int 0x80  
•  
• soma:  
• ;ao chegar aqui já guardou o RIP na pilha, -8 bytes  
• push rbp ;salva rpb, -8 bytes  
• ;usa-se o bp para guardar o valor de sp no stack frame  
• ;o bp é a base para aceder os argumentos da pilha  
• mov rbp,rsp  
• mov rax,[rbp+24] ;1o arg  
• add rax,[rbp+16] ;2o arg  
• ;caso precisasse o sp pode ser modificado á vontade  
• mov rsp,rbp  
• pop rbp ;recupera bp antigo  
• ret ;salta para o
```

Uso de funções e pilha

Posição da pilha, após empilhar 4 (push 4).

RSP, que apontava para o endereço 0xd3482aa0 é diminuído de 8 bytes. Ou seja, $RSP=RSP-8$

Veja em Stack, o valor 4 no endereço

0xd3482a98, atual endereço de RSP.

The screenshot shows the edb debugger interface with the following components:

- Assembly Window:** Displays assembly instructions. The instruction `push 4` at address `00000000:00400080` is highlighted with a red arrow. The instruction `push 5` at address `00000000:00400082` is highlighted with a green arrow. Other instructions include `call fooSomapilha!soma`, `mov rcx, rax`, `mov eax, 1`, `mov ebx, 0`, `int 0x80`, `push rbp`, `mov rbp, rsp`, `mov rax, [rbp+0x18]`, `add rax, [rbp+0x10]`, `mov rsp, rbp`, `pop rbp`, `ret`, and `add [rax], al`.
- Registers Window:** Shows the state of registers. RSP is `00007fffd3482a98`. RIP is `0000000000400082`.
- Data Dump Window:** Shows memory data. The address `0x0000000000400000-0x0000000000401000` is selected. The stack data shows the value `0000000000000004` at address `00007fff:d3482a98`.
- Stack Window:** Shows the stack contents. The value `0000000000000004` is highlighted at address `00007fff:d3482a98`. Other stack entries include ASCII strings like `"/home/ojacques/Documents/Asse`, `"XDG_VTNR=7"`, `"SSH_AGENT_PID=1664"`, `"XDG_SESSION_ID=c2"`, `"XDG_GREETER_DATA_DIR=/var/lib,`, `"QT_STYLE_OVERRIDE=gtk"`, `"TERM=xterm-256color"`, `"SHELL=/bin/bash"`, `"VTE_VERSION=4205"`, `"QT_LINUX_ACCESSIBILITY_ALWAYS`, `"WINDOWID=96469057"`, `"GTK_MODULES=gail:atk-bridge"`, and `"USER=ojacques"`.

Uso de funções e pilha

Posição da pilha, após empilhar 5.

Veja em Stack, que 4 esta no endereço 0xd3482a98, e 5 está na posição 0xd3482a90.

Ou seja, o endereço decresceu 8 bytes (64 bits).

A pilha decresce ou cresce para baixo.

The screenshot shows the edb debugger interface with the following components:

- Assembly View:** Shows assembly instructions. The instruction `call fooSomapilha!soma` at address `00000000:00400084` is highlighted in green. Below it, instructions for `mov rcx, rax`, `mov eax, 1`, `mov ebx, 0`, `int 0x80`, `push rbp`, `mov rbp, rsp`, `mov rax, [rbp+0x18]`, `add rax, [rbp+0x10]`, `mov rsp, rbp`, `pop rbp`, and `ret` are visible. A label `soma` is placed below the `ret` instruction.
- Registers:** A list of registers on the right side. `RSP` is highlighted with the value `00007fffd3482a90`. Other registers like `RAX`, `RDX`, `RBX`, `RBP`, `RSI`, `RDI`, `R8`, `R9`, `R10`, `R11`, `R12`, `R13`, `R14`, `R15`, and `RIP` are also listed.
- Stack:** A window at the bottom right showing memory addresses and their contents. The address `00007fff:d3482a90` contains `0000000000000005` (5), and `00007fff:d3482a98` contains `0000000000000004` (4). The stack grows downwards.
- Data Dump:** A window at the bottom left showing a hex dump of memory. The address `0x0000000000400000-0x0000000000401000` is selected.

Annotations in the image include:

- An arrow pointing from the text "Posição da pilha, após empilhar 5." to the `call` instruction.
- An arrow pointing from the text "Veja em Stack, que 4 esta no endereço 0xd3482a98, e 5 está na posição 0xd3482a90." to the stack window.
- An arrow pointing from the text "Ou seja, o endereço decresceu 8 bytes (64 bits)." to the stack window.
- An arrow pointing from the text "A pilha decresce ou cresce para baixo." to the stack window.
- Text labels in the assembly view: "Posicoes antes da chamada de soma" (Addresses before the function call), "Endereco apos chamada funcao" (Address after function call), and "soma" (Function name).

Uso de funções e pilha

Posição após a chamada da função soma. Repare que fica guardado no topo da pilha o valor 0x400089. Esse valor é o endereço da próxima instrução após a execução da função soma.

RSP aponta para o topo da pilha (TOS). O topo é o endereço 0xd3482a88. Observe que $0xd3482a90 - 0x8 = 0xd3482a88$, pois estamos em notação hexadecimal.

The screenshot shows the EDB debugger interface with the following components:

- Assembly View:** Shows assembly instructions. The instruction `mov rcx, rax` at address `00400089` is highlighted. A blue arrow points to the `[rsp]` register in the instruction list, and another blue arrow points to the value `00000000:00400089` in the instruction list. A text box labeled "Endereço de retorno apos chamada da funcao" points to this value.
- Registers Panel:** Shows the state of registers. `RSP` is `00007fff:d3482a88`. `RIP` is `00000000:00400098`.
- Stack Panel:** Shows the stack dump. The top of the stack is at `00007fff:d3482a88` with the value `0000000000400089`. A text box labeled "RSP aponta para o topo da pilha (TOS). O topo é o endereço 0xd3482a88." points to this address. Below it, the value `0000000000000005` is shown at `00007fff:d3482a90`.
- Registers List:** Shows the state of other registers: `RAX`, `RCX`, `RDY`, `RBX`, `RBP`, `RSI`, `RDI`, `R8`, `R9`, `R10`, `R11`, `R12`, `R13`, `R14`, `R15`, and `RIP`.
- Registers Panel:** Shows the state of registers: `RAX`, `RCX`, `RDY`, `RBX`, `RSP`, `RBP`, `RSI`, `RDI`, `R8`, `R9`, `R10`, `R11`, `R12`, `R13`, `R14`, `R15`, and `RIP`.

Uso de funções e pilha

Dentro da função, a pilha ainda decresce com o salvamento de RBP, registrador de base. RSP agora aponta para 0xd3482a80, diminuindo mais 8 bytes. Na linha 0x400099 será atribuído a RBP o valor do topo da pilha atual (TOS), pela instrução `mov rbp, rsp`. O valor anterior de RBP foi salvo com `push rbp`.

The screenshot shows the edb debugger interface with the following components:

- Assembly View:** Displays assembly instructions. The instruction `mov rbp, rsp` at address `00000000:00400099` is highlighted in green. A red arrow points from the text "Endereço do topo da pilha" to the `rsp` register in the Registers window, and another red arrow points from the same text to the `mov rbp, rsp` instruction. A black arrow points from the text "salva conteúdo de RBP na pilha" to the `push rbp` instruction at address `00000000:00400098`.
- Registers Window:** Shows the state of registers. `RSP` is highlighted with a red box and contains the value `00007fffd3482a80`. `RIP` contains `0000000000400099`.
- Data Dump:** Shows a memory dump with a search filter `0x0000000000400000-0x0000000000401000`. The dump shows hex and ASCII data.
- Stack Window:** Shows the stack contents. The top of the stack is at `00007fff:d3482a80` and contains `0000000000000000`. Below it, the return address `00007fff:d3482a88` is visible, which corresponds to the `ret` instruction in the assembly view.
- Registers Window (Bottom):** Shows the current values of `C` (0), `ES` (0000), `P` (0), and `CS` (0033).

Uso de funções e pilha

A instrução atual, `mov rax, [rbp+0x18]` é o mesmo que `mov rax, [rbp+24]`. Essa instrução acessa o primeiro valor empilhado, 4. Para somar 4 ao valor 5, fazemos `add rax, [rbp+0x10]`, que é o mesmo que `add rax, [rbp+16]`.

A região da pilha desde os parâmetros, endereço de retorno, valor de RBP salvo e variáveis que possam ser empilhadas é denominada **Stack Frame**.

The screenshot shows the edb debugger interface with the following components:

- Assembly View:** Displays assembly instructions. The current instruction is `mov rax, [rbp+0x18]`, highlighted in green. A callout points to it with the text "Atentar que 0x18=24d".
- Registers:** Shows the state of CPU registers. RBP is highlighted with a red box and labeled "Valor de RBP". Its value is `00007fffd3482a80`.
- Data Dump:** Shows memory contents. A callout points to the value `0x0000000000000004` at address `0x0000000040000000`, labeled "o argumento empilhado".
- Stack Frame:** Shows the stack frame structure. A callout points to the return address `00007fffd3482a80`, labeled "Stack Frame".

The assembly code shown is:

```
0000000000400080 ... push 4
0000000000400082 ... push 5
0000000000400084 ... call fooSomapilha!soma
0000000000400089 ... mov rcx, rax
000000000040008c ... mov eax, 1
0000000000400091 ... mov ebx, 0
0000000000400096 ... int 0x80
0000000000400098 ... push rbp
0000000000400099 ... mov rbp, rsp
000000000040009c ... mov rax, [rbp+0x18]
00000000004000a0 ... add rax, [rbp+0x10]
00000000004000a4 ... mov rsp, rbp
00000000004000a7 ... pop rbp
00000000004000a8 ... ret
00000000004000a9 ... add [rax], al
00000000004000ab ... add [rcx], al
00000000004000ad ... add [rax], al
00000000004000af ... add [rax], al
```

Uso de funções e pilha

A instrução `mov rsp,rbp` faz com que RSP volte a apontar para o local onde estava armazenado o valor antigo de RBP.

Após isso, `pop rbp` recupera o valor antigo de RBP

Quando a instrução `ret` for executada, ela RETornará para a próxima linha após a chamada da função soma.

Essa linha está armazenada no TOS apontada por RSP (0x400089).

The screenshot shows the EDB debugger interface for the file `fooSomapilha [7414]`. The assembly window displays the following code:

```
000000000400080 ... push 4
000000000400082 ... push 5
000000000400084 ... call fooSomapilha!soma
000000000400089 ... mov rcx, rax
00000000040008c ... mov eax, 1
000000000400091 ... mov ebx, 0
000000000400096 ... int 0x80
000000000400098 ... push rbp
000000000400099 ... mov rbp, rsp
00000000040009c ... mov rax, [rbp+0x18]
0000000004000a0 ... add rax, [rbp+0x10]
0000000004000a4 ... mov rsp, rbp
0000000004000a7 ... pop rbp ;retirou RBP da pilha
0000000004000a8 ... ret
0000000004000a9 ... add [rax], al
0000000004000ab ... add [rcx], al
0000000004000ad ... add [rax], al
0000000004000af ... add [rax], al
```

The registers window shows the current state of registers, with RSP at `00007fffd3482a88` and RBP at `0000000000000000`.

The stack window shows the return address `return to 0x000000000400089` at the top of the stack, which is the address of the instruction `mov rcx, rax` in the assembly window.

Annotations in red text and boxes explain the state of the stack and registers:

- A red box highlights the instruction `mov rcx, rax` at address `000000000400089`, with an arrow pointing to the RSP register value in the registers window.
- A red box highlights the instruction `ret` at address `0000000004000a8`, with an arrow pointing to the return address in the stack window.
- Red text explains: "Quando RET for executado, o programa saltara para o endereço que esta no topo da pilha. Ou seja, o endereço de retorno."

Uso de funções e pilha

Após voltar da função soma, executa-se `mov rcx, rax`, salvando o resultado da soma em RCX. É sempre recomendado que os resultados venham através do acumulador (RAX).

Veja que o valor de RAX é 9.

The screenshot shows the EDB debugger interface for the file `fooSomapilha`. The assembly window displays the following code:

```
0000000000400080 ... push 4
0000000000400082 ... push 5
0000000000400084 ... call fooSomapilha.soma
0000000000400089 ... mov rcx, rax
000000000040008c ... mov eax, 1
0000000000400091 ... mov ebx, 0
0000000000400096 ... int 0x80
0000000000400098 ... push rbp
0000000000400099 ... mov rbp, rsp
000000000040009c ... mov rax, [rbp+0x18]
00000000004000a0 ... add rax, [rbp+0x10]
00000000004000a4 ... mov rsp, rbp
00000000004000a7 ... pop rbp
00000000004000a8 ... ret
00000000004000a9 ... add [rax], al
00000000004000ab ... add [rcx], al
00000000004000ad ... add [rax], al
00000000004000af ... add [rax], al
```

The registers window shows the following values:

Register	Value
RAX	0000000000000009
RCX	0000000000000009
RDX	0000000000000000
RBX	0000000000000000
RSP	00007fffd3482a90
RBP	0000000000000000
RSI	0000000000000000
RDI	0000000000000000
R8	0000000000000000
R9	0000000000000000
R10	0000000000000000
R11	0000000000000000
R12	0000000000000000
R13	0000000000000000
R14	0000000000000000
R15	0000000000000000
RIP	000000000040008c

The stack window shows the following data:

Address	Hex	ASCII
00007fffd3482a90	0000000000000005	
00007fffd3482a98	0000000000000004	
00007fffd3482aa0	0000000000000001	
00007fffd3482aa8	00007fffd348335b	ASCII "/home/ojacques/Documentos/Asse
00007fffd3482ab0	0000000000000000	
00007fffd3482ab8	00007fffd348338b	ASCII "XDG_VTNR=7"
00007fffd3482ac0	00007fffd3483396	ASCII "SSH_AGENT_PID=1664"
00007fffd3482ac8	00007fffd34833a9	ASCII "XDG_SESSION_ID=c2"
00007fffd3482ad0	00007fffd34833bb	ASCII "XDG_GREETER_DATA_DIR=/var/lib/T
00007fffd3482ad8	00007fffd34833ef	ASCII "QT_STYLE_OVERRIDE=gtk"
00007fffd3482ae0	00007fffd3483405	ASCII "TERM=xterm-256color"
00007fffd3482ae8	00007fffd3483419	ASCII "SHELL=/bin/bash"
00007fffd3482af0	00007fffd3483429	ASCII "VTE_VERSION=4205"
00007fffd3482af8	00007fffd348343a	ASCII "QT_LINUX_ACCESSIBILITY_ALWAYS_C
00007fffd3482b00	00007fffd348345d	ASCII "WINDOWID=96469057"
00007fffd3482b08	00007fffd348346f	ASCII "GTK_MODULES=gail:atk-bridge"