Debugando o Programa

+ O programa estudado será o protegido, fornecido pela DesecTools
+ A execução dele requer alguma senha q ainda não temos

```
┌──(root㉿DESKTOP-NJHHNK6)-[/home/kali/Downloads/LinuxTools]
└─# ./protegido
Entre com a senha: kajslkjslkfjd
Acesso Negado

┌──(root㉿DESKTOP-NJHHNK6)-[/home/kali/Downloads/LinuxTools]
└─# ./protegido uhwdohasod
Sem argumentos!

┌──(root㉿DESKTOP-NJHHNK6)-[/home/kali/Downloads/LinuxTools]
└─# ./protegido
Entre com a senha: 123
Essa senha foi desativada!
```

+ Vamos debugar o código com o gdb
+ Nossa interação com o programa será o envio de argumentos. Dito isso, ao tentar enviar uns 200 A's, vemos que o programa apresentará um erro

```
python3 -c 'print ("A"*200)' | ./protegido
```

```
┌──(root㉿DESKTOP-NJHHNK6)-[/home/kali/Downloads/LinuxTools]
└─# python3 -c 'print ("A" *200)' | ./protegido
Entre com a senha: Acesso Negado
zsh: done                python3 -c 'print ("A" *200)' |
zsh: segmentation fault  ./protegido
```

+ Usando o gdb

```
gdb -q ./protegido
```

(-q pra deixar o texto exposto mais limpo)

```
┌──(root㉿DESKTOP-NJHHNK6)-[/home/kali/Downloads/LinuxTools]
└─# gdb -q ./protegido
Reading symbols from ./protegido...
(No debugging symbols found in ./protegido)
```

+ Para ver as funções usadas:

```
info functions
```

```
All defined functions:

Non-debugging symbols:
0x00001000  _init
0x00001030  strcmp@plt
0x00001040  printf@plt
0x00001050  gets@plt
0x00001060  puts@plt
0x00001070  system@plt
0x00001080  exit@plt
0x00001090  __libc_start_main@plt
0x000010a0  __cxa_finalize@plt
0x000010b0  _start
0x000010f0  __x86.get_pc_thunk.bx
0x00001100  deregister_tm_clones
0x00001140  register_tm_clones
0x00001190  __do_global_dtors_aux
0x000011e0  frame_dummy
0x000011e5  __x86.get_pc_thunk.dx
0x000011e9  verifica
0x00001293  acessa
0x000012d3  main
0x00001330  __libc_csu_init
0x00001390  __libc_csu_fini
0x00001391  __x86.get_pc_thunk.bp
0x00001398  _fini
```

+ Quando n sabemos o q uma função faz, podemos perguntar para a man

```
man strcmp
```

```
strcmp(3)

NAME
       strcmp, strncmp - compare two strings
```

→ Compara duas strings

+ Para fazer um disassembly da função main:

```
disas main
```

```
(gdb) disas main
Dump of assembler code for function main:
   0x565562d3 <+0>:     lea    ecx,[esp+0x4]
   0x565562d7 <+4>:     and    esp,0xfffffff0
   0x565562da <+7>:     push   DWORD PTR [ecx-0x4]
   0x565562dd <+10>:    push   ebp
   0x565562de <+11>:    mov    ebp,esp
   0x565562e0 <+13>:    push   ebx
   0x565562e1 <+14>:    push   ecx
   0x565562e2 <+15>:    call   0x565560f0 <__x86.get_pc_thunk.bx>
   0x565562e7 <+20>:    add    ebx,0x2d19
   0x565562ed <+26>:    mov    eax,ecx
   0x565562ef <+28>:    cmp    DWORD PTR [eax],0x1
   0x565562f2 <+31>:    jle    0x56556310 <main+61>
   0x565562f4 <+33>:    sub    esp,0xc
   0x565562f7 <+36>:    lea    eax,[ebx-0x1f9a]
   0x565562fd <+42>:    push   eax
   0x565562fe <+43>:    call   0x56556060 <puts@plt>
   0x56556303 <+48>:    add    esp,0x10
   0x56556306 <+51>:    sub    esp,0xc
   0x56556309 <+54>:    push   0x1
   0x5655630b <+56>:    call   0x56556080 <exit@plt>
   0x56556310 <+61>:    call   0x565561e9 <verifica>
   0x56556315 <+66>:    mov    eax,0x0
   0x5655631a <+71>:    lea    esp,[ebp-0x8]
   0x5655631d <+74>:    pop    ecx
   0x5655631e <+75>:    pop    ebx
   0x5655631f <+76>:    pop    ebp
   0x56556320 <+77>:    lea    esp,[ecx-0x4]
   0x56556323 <+80>:    ret
```

→ Para que a syntax exibida seja a da Intel,

```
set disassembly-flavor intel
```

→ Pra executar o programa:

```
run
```

ou

```
r
```

```
(gdb) run
Starting program: /home/kali/Downloads/LinuxTools/protegido
[Thread debugging using libthread_db enabled]
Using host libthread_db library "/lib/x86_64-linux-gnu/libthread_db.so.1".
Entre com a senha: hçjhwefhwe
Acesso Negado
[Inferior 1 (process 25260) exited normally]
(gdb) r
Starting program: /home/kali/Downloads/LinuxTools/protegido
[Thread debugging using libthread_db enabled]
Using host libthread_db library "/lib/x86_64-linux-gnu/libthread_db.so.1".
Entre com a senha: ksjhfkhskf
Acesso Negado
[Inferior 1 (process 25439) exited normally]
```

→ Para passar argumentos vindos de um arquivo

```
run < file.txt
```

→ Para passar comandos:

```
run < < (comandos)
```

→ Ao fazer o disas main, vemos que ele faz uma call para a "verifica" em algum momento

→ Vamos fazer um disas verifica pra estudá-la

```
disas verifica
```

```
Dump of assembler code for function verifica:
   0x565561e9 <+0>:     push   ebp
   0x565561ea <+1>:     mov    ebp,esp
   0x565561ec <+3>:     push   ebx
   0x565561ed <+4>:     sub    esp,0x84
   0x565561f3 <+10>:    call   0x565560f0 <__x86.get_pc_thunk.bx>
   0x565561f8 <+15>:    add    ebx,0x2e08
   0x565561fe <+21>:    sub    esp,0xc
   0x56556201 <+24>:    lea    eax,[ebx-0x1ff8]
   0x56556207 <+30>:    push   eax
   0x56556208 <+31>:    call   0x56556040 <printf@plt>
   0x5655620d <+36>:    add    esp,0x10
   0x56556210 <+39>:    sub    esp,0xc
   0x56556213 <+42>:    lea    eax,[ebp-0x88]
   0x56556219 <+48>:    push   eax
   0x5655621a <+49>:    call   0x56556050 <gets@plt>
   0x5655621f <+54>:    add    esp,0x10
   0x56556222 <+57>:    sub    esp,0x8
   0x56556225 <+60>:    lea    eax,[ebx-0x1fe4]
   0x5655622b <+66>:    push   eax
   0x5655622c <+67>:    lea    eax,[ebp-0x88]
   0x56556232 <+73>:    push   eax
   0x56556233 <+74>:    call   0x56556030 <strcmp@plt>
   0x56556238 <+79>:    add    esp,0x10
   0x5655623b <+82>:    test   eax,eax
   0x5655623d <+84>:    jne    0x56556253 <verifica+106>
   0x5655623f <+86>:    sub    esp,0xc
   0x56556242 <+89>:    lea    eax,[ebx-0x1fe0]
   0x56556248 <+95>:    push   eax
   0x56556249 <+96>:    call   0x56556060 <puts@plt>
   0x5655624e <+101>:   add    esp,0x10
   0x56556251 <+104>:   jmp    0x56556289 <verifica+160>
   0x56556253 <+106>:   sub    esp,0x8
   0x56556256 <+109>:   lea    eax,[ebx-0x1fc5]
   0x5655625c <+115>:   push   eax
   0x5655625d <+116>:   lea    eax,[ebp-0x88]
   0x56556263 <+122>:   push   eax
   0x56556264 <+123>:   call   0x56556030 <strcmp@plt>
   0x56556269 <+128>:   add    esp,0x10
   0x5655626c <+131>:   test   eax,eax
```

→ Veja que com esse printf, concluímos que ela printa alguma coisa na tela
→ com o lea, verificamos que ela aloca espaço na memória [buffer]
→ Veja q ela está acima do gets. Então é provável que quando executarmos a gets, os argumentos recebidos vão ser alocados no espaço reservado do buffer
→ Para ver o tamanho do buffer:

```
python3 -x "print (0x88)"
```

```
┌──(root㉿DESKTOP-NJHHNK6)-[/home/kali]
└─# python3 -c "print (0x88)"
136
```

→ Veja os JNE (jump not equal) que indicam que há comparação do argumento passado com alguma coisa e as instruções pra caso seja igual e caso não
→ Vamos dar um disas na acessa

```
disas acessa
```

```
(gdb) disas acessa
Dump of assembler code for function acessa:
   0x56556293 <+0>:     push   ebp
   0x56556294 <+1>:     mov    ebp,esp
   0x56556296 <+3>:     push   ebx
   0x56556297 <+4>:     sub    esp,0x4
   0x5655629a <+7>:     call   0x565560f0 <__x86.get_pc_thunk.bx>
   0x5655629f <+12>:    add    ebx,0x2d61
   0x565562a5 <+18>:    sub    esp,0xc
   0x565562a8 <+21>:    lea    eax,[ebx-0x1fad]
   0x565562ae <+27>:    push   eax
   0x565562af <+28>:    call   0x56556060 <puts@plt>
   0x565562b4 <+33>:    add    esp,0x10
   0x565562b7 <+36>:    sub    esp,0xc
   0x565562ba <+39>:    lea    eax,[ebx-0x1fa1]
   0x565562c0 <+45>:    push   eax
   0x565562c1 <+46>:    call   0x56556070 <system@plt>
   0x565562c6 <+51>:    add    esp,0x10
   0x565562c9 <+54>:    mov    eax,0x0
   0x565562ce <+59>:    mov    ebx,DWORD PTR [ebp-0x4]
   0x565562d1 <+62>:    leave
   0x565562d2 <+63>:    ret
```

→ o puts indica que ela exibe alguma coisa na tela e o system indica que ela executa algum comando no sistema operacional
→ Quando executamos o envio dos 200 A's, vemos o programa dar o crash e então podemos analisar os registradores com info registers ou i r

```
python3 -c 'print("A"*200)' > arg.txt
```

```
(gdb) run < arg.txt
Starting program: /home/kali/Downloads/LinuxTools/protegido < arg.txt
[Thread debugging using libthread_db enabled]
Using host libthread_db library "/lib/x86_64-linux-gnu/libthread_db.so.1".
Entre com a senha: Acesso Negado

Program received signal SIGSEGV, Segmentation fault.
0x41414141 in ?? ()
```

```
(gdb) i r
eax            0x0                 0
ecx            0xf7e1f9b8          -136185416
edx            0x0                 0
ebx            0x41414141          1094795585
esp            0xffffd310          0xffffd310
ebp            0x41414141          0x41414141
esi            0x56556330          1448436528
edi            0xf7ffcba0          -134231136
eip            0x41414141          0x41414141
eflags         0x10282             [ SF IF RF ]
cs             0x23                35
ss             0x2b                43
ds             0x2b                43
es             0x2b                43
fs             0x0                 0
gs             0x63                99
```

→ Veja a sobrescrição do EIP pelos A's, assim como o EBX
→ Tem outro comando que serve muito bem para vermos as informações da memória:
o x

```
help x
```

(gdb) help x
Examine memory: x/FMT ADDRESS.
ADDRESS is an expression for the memory address to examine.
FMT is a repeat count followed by a format letter and a size letter.
Format letters are o(octal), x(hex), d(decimal), u(unsigned decimal),
  t(binary), f(float), a(address), i(instruction), c(char), s(string)
  and z(hex, zero padded on the left).
Size letters are b(byte), h(halfword), w(word), g(giant, 8 bytes).
The specified number of objects of the specified size are printed
according to the format. If a negative number is specified, memory is
examined backward from the address.

Defaults for format and size letters are those previously used.
Default count is 1. Default address is following last thing printed
with this command or "print".

```
x/16xb $esp
```

```
(gdb) x/16xb $esp
0xffffd310:     0x41    0x41    0x41    0x41    0x41    0x41    0x41    0x41
0xffffd318:     0x41    0x41    0x41    0x41    0x41    0x41    0x41    0x41
```

- 16 indica a qtd de bytes exibidos
- x (o 2º) que queremos a resposta em hexadecimal
- b byte por byte

```
x/16xw $esp
```

```
(gdb) x/16xw $esp
0xffffd310:     0x41414141      0x41414141      0x41414141      0x41414141
0xffffd320:     0x41414141      0x41414141      0x41414141      0x41414141
0xffffd330:     0x41414141      0x41414141      0x41414141      0x41414141
0xffffd340:     0x41414141      0x41414141      0x00000000      0xffffd3e4
```

→ Isso equivale em linha de comando aos "Follow in Dump"