

Laboratório 2 - BOMB

Alunos: Livia Barbosa Fonseca DRE:118039721

Luiz Rodrigo Lace DRE: 118049873

Introdução

Utilizamos o comando “gdb bomb” para inicializar o executável em binário e depois o comando “layout asm”, conseguimos visualizar o código de montagem do programa. O que é retornado são as seguintes rotinas:

Utilizando o comando gdb

```
luiz@Jarvis: ~/Área de Trabalho/labs comp prog/lab2/02-bomb
luiz@Jarvis:~/Área de Trabalho/labs comp prog/lab2/02-bomb$ gdb bomb
GNU gdb (Ubuntu 9.2-0ubuntu1~20.04) 9.2
Copyright (C) 2020 Free Software Foundation, Inc.
License GPLv3+: GNU GPL version 3 or later <http://gnu.org/licenses/gpl.html>
This is free software: you are free to change and redistribute it.
There is NO WARRANTY, to the extent permitted by law.
Type "show copying" and "show warranty" for details.
This GDB was configured as "x86_64-linux-gnu".
Type "show configuration" for configuration details.
For bug reporting instructions, please see:
<http://www.gnu.org/software/gdb/bugs/>.
Find the GDB manual and other documentation resources online at:
<http://www.gnu.org/software/gdb/documentation/>.

For help, type "help".
Type "apropos word" to search for commands related to "word"...
Reading symbols from bomb...
(No debugging symbols found in bomb)
(gdb) █
```

Após utilizar o comando layout asm, chegamos nessa tela:

```
luiz@Jarvis: ~/Área de Trabalho/labs comp prog/lab2/02-bomb

0x16cb <quartaBomba+220>    call    0x1820 <__stack_chk_fail_local>
0x16d0 <quartaBomba+225>    mov     -0x4(%ebp),%ebx
0x16d3 <quartaBomba+228>    leave
0x16d4 <quartaBomba+229>    ret
0x16d5 <main>               lea     0x4(%esp),%ecx
0x16d9 <main+4>              and     $0xffffffff0,%esp
0x16dc <main+7>              pushl   -0x4(%ecx)
0x16df <main+10>            push    %ebp
0x16e0 <main+11>            mov     %esp,%ebp
0x16e2 <main+13>            push    %esi
0x16e3 <main+14>            push    %ebx
0x16e4 <main+15>            push    %ecx
0x16e5 <main+16>            sub     $0xc,%esp
0x16e8 <main+19>            call    0x179a <__x86.get_pc_thunk.si>
0x16ed <main+24>            add     $0x2913,%esi
0x16f3 <main+30>            mov     %ecx,%ebx

exec No process in:
(gdb) █
```

Copiando todo o conteúdo, na ordem, chegamos no seguinte código:

```
0x122d <boom>          push  %ebp
0x122e <boom+1>         mov   %esp,%ebp
0x1230 <boom+3>         push  %ebx
0x1231 <boom+4>         sub   $0x4,%esp
0x1234 <boom+7>         call  0x1130 <__x86.get_pc_thunk.bx>
0x1239 <boom+12>        add   $0x2dc7,%ebx
0x123f <boom+18>        sub   $0xc,%esp
0x1242 <boom+21>        lea   -0x1ff8(%ebx),%eax
0x1248 <boom+27>        push  %eax
0x1249 <boom+28>        call  0x10a0 <puts@plt>
0x124e <boom+33>        add   $0x10,%esp
0x1251 <boom+36>        sub   $0xc,%esp
0x1254 <boom+39>        push  $0x1
0x1256 <boom+41>        call  0x10b0 <exit@plt>

0x125b <inicializa>     push  %ebp
0x125c <inicializa+1>   mov   %esp,%ebp
0x125e <inicializa+3>   push  %ebx
0x125f <inicializa+4>   sub   $0x824,%esp
0x1265 <inicializa+10>  call  0x1130 <__x86.get_pc_thunk.bx>
0x126a <inicializa+15>  add   $0x2d96,%ebx
0x1270 <inicializa+21>  mov   0x8(%ebp),%eax
0x1273 <inicializa+24>  mov   %eax,-0x81c(%ebp)
0x1279 <inicializa+30>  mov   %gs:0x14,%eax
0x127f <inicializa+36>  mov   %eax,-0xc(%ebp)
0x1282 <inicializa+39>  xor   %eax,%eax
0x1284 <inicializa+41>  sub   $0x8,%esp
0x1287 <inicializa+44>  lea   -0x1fe5(%ebx),%eax
0x128d <inicializa+50>  push  %eax
0x128e <inicializa+51>  lea   -0x1fe3(%ebx),%eax
0x1294 <inicializa+57>  push  %eax
0x1295 <inicializa+58>  call  0x10e0 <fopen@plt>
0x129a <inicializa+63>  add   $0x10,%esp
0x129d <inicializa+66>  mov   %eax,-0x810(%ebp)
0x12a3 <inicializa+72>  cmpl  $0x0,-0x810(%ebp)
0x12aa <inicializa+79>  je    0x12f2 <inicializa+151>
0x12ac <inicializa+81>  pushl -0x810(%ebp)
0x12b2 <inicializa+87>  push  $0x1d
0x12b4 <inicializa+89>  push  $0x1
0x12b6 <inicializa+91>  lea   -0x1fd8(%ebx),%eax
0x12bc <inicializa+97>  push  %eax
0x12bd <inicializa+98>  call  0x1090 <fwrite@plt>
0x12c2 <inicializa+103> add   $0x10,%esp
0x12c5 <inicializa+106> sub   $0xc,%esp
0x12c8 <inicializa+109> pushl -0x810(%ebp)
0x12ce <inicializa+115> call  0x1060 <fclose@plt>
```

0x12d3 <inicializa+120>	add \$0x10,%esp
0x12d6 <inicializa+123>	sub \$0xc,%esp
0x12d9 <inicializa+126>	push \$0x1
0x12db <inicializa+128>	call 0x1070 <sleep@plt>
0x12e0 <inicializa+133>	add \$0x10,%esp
0x12e3 <inicializa+136>	nop
0x12e4 <inicializa+137>	mov -0xc(%ebp),%eax
0x12e7 <inicializa+140>	sub %gs:0x14,%eax
0x12ee <inicializa+147>	je 0x1324 <inicializa+201>
0x12f0 <inicializa+149>	jmp 0x131f <inicializa+196>
0x12f2 <inicializa+151>	sub \$0xc,%esp
0x12f5 <inicializa+154>	lea -0x1fb8(%ebx),%eax
0x12fb <inicializa+160>	push %eax
0x12fc <inicializa+161>	call 0x10a0 <puts@plt>
0x1301 <inicializa+166>	add \$0x10,%esp
0x1304 <inicializa+169>	sub \$0xc,%esp
0x1307 <inicializa+172>	pushl -0x810(%ebp)
0x130d <inicializa+178>	call 0x1060 <fclose@plt>
0x1312 <inicializa+183>	add \$0x10,%esp
0x1315 <inicializa+186>	sub \$0xc,%esp
0x1318 <inicializa+189>	push \$0x1
0x131a <inicializa+191>	call 0x10b0 <exit@plt>
0x131f <inicializa+196>	call 0x1820 <__stack_chk_fail_local>
0x1324 <inicializa+201>	mov -0x4(%ebp),%ebx
0x1327 <inicializa+204>	leave
0x1328 <inicializa+205>	ret
0x1329 <primeiraBomba>	push %ebp
0x132a <primeira Bomba+1>	mov %esp,%ebp
0x132c <primeiraBomba+3>	push %ebx
0x132d <primeiraBomba+4>	sub \$0x4,%esp
0x1330 <primeiraBomba+7>	call 0x1130 <__x86.get_pc_thunk.bx>
0x1335 <primeiraBomba+12>	add \$0x2ccb,%ebx
0x133b <primeiraBomba+18>	mov 0x8(%ebp),%eax
0x133e <primeiraBomba+21>	movzbl (%eax),%eax
0x1341 <primeiraBomba+24>	cmp \$0x4d,%al
0x1343 <primeiraBomba+26>	je 0x1353 <primeiraBomba+42>
0x1345 <primeiraBomba+28>	mov 0x8(%ebp),%eax
0x1348 <primeiraBomba+31>	movzbl (%eax),%eax
0x134b <primeiraBomba+34>	cmp \$0x6d,%al
0x134d <primeiraBomba+36>	jne 0x144c <primeiraBomba+291>
0x1353 <primeiraBomba+42>	mov 0x8(%ebp),%eax
0x1356 <primeiraBomba+45>	add \$0x1,%eax
0x1359 <primeiraBomba+48>	movzbl (%eax),%eax
0x135c <primeiraBomba+51>	cmp \$0x61,%al
0x135e <primeiraBomba+53>	jne 0x144c <primeiraBomba+291>
0x1364 <primeiraBomba+59>	mov 0x8(%ebp),%eax
0x1367 <primeiraBomba+62>	add \$0x2,%eax

0x136a <primeiraBomba+65>	movzbl (%eax),%eax
0x136d <primeiraBomba+68>	cmp \$0x62,%al
0x136f <primeiraBomba+70>	jne 0x144c <primeiraBomba+291>
0x1375 <primeiraBomba+76>	mov 0x8(%ebp),%eax
0x1378 <primeiraBomba+79>	add \$0x3,%eax
0x137b <primeiraBomba+82>	movzbl (%eax),%eax
0x137e <primeiraBomba+85>	cmp \$0x20,%al
0x1380 <primeiraBomba+87>	jne 0x144c <primeiraBomba+291>
0x1386 <primeiraBomba+93>	mov 0x8(%ebp),%eax
0x1389 <primeiraBomba+96>	add \$0x4,%eax
0x138c <primeiraBomba+99>	movzbl (%eax),%eax
0x138f <primeiraBomba+102>	cmp \$0x2d,%al
0x1391 <primeiraBomba+104>	jne 0x144c <primeiraBomba+291>
0x1397 <primeiraBomba+110>	mov 0x8(%ebp),%eax
0x139a <primeiraBomba+113>	add \$0x5,%eax
0x139d <primeiraBomba+116>	movzbl (%eax),%eax
0x13a0 <primeiraBomba+119>	cmp \$0x20,%al
0x13a2 <primeiraBomba+121>	jne 0x144c <primeiraBomba+291>
0x13a8 <primeiraBomba+127>	mov 0x8(%ebp),%eax
0x13ab <primeiraBomba+130>	add \$0x6,%eax
0x13ae <primeiraBomba+133>	movzbl (%eax),%eax
0x13b1 <primeiraBomba+136>	cmp \$0x33,%al
0x13b3 <primeiraBomba+138>	jne 0x144c <primeiraBomba+291>
0x13b9 <primeiraBomba+144>	mov 0x8(%ebp),%eax
0x13bc <primeiraBomba+147>	add \$0x7,%eax
0x13bf <primeiraBomba+150>	movzbl (%eax),%eax
0x13c2 <primeiraBomba+153>	cmp \$0x35,%al
0x13c4 <primeiraBomba+155>	jne 0x144c <primeiraBomba+291>
0x13ca <primeiraBomba+161>	mov 0x8(%ebp),%eax
0x13cd <primeiraBomba+164>	add \$0x8,%eax
0x13d0 <primeiraBomba+167>	movzbl (%eax),%eax
0x13d3 <primeiraBomba+170>	cmp \$0x33,%al
0x13d5 <primeiraBomba+172>	jne 0x144c <primeiraBomba+291>
0x13d7 <primeiraBomba+174>	mov 0x8(%ebp),%eax
0x13da <primeiraBomba+177>	add \$0x9,%eax
0x13dd <primeiraBomba+180>	movzbl (%eax),%eax
0x13e0 <primeiraBomba+183>	cmp \$0x20,%al
0x13e2 <primeiraBomba+185>	jne 0x144c <primeiraBomba+291>
0x13e4 <primeiraBomba+187>	mov 0x8(%ebp),%eax
0x13e7 <primeiraBomba+190>	add \$0xa,%eax
0x13ea <primeiraBomba+193>	movzbl (%eax),%eax
0x13ed <primeiraBomba+196>	cmp \$0x2d,%al
0x13ef <primeiraBomba+198>	jne 0x144c <primeiraBomba+291>
0x13f1 <primeiraBomba+200>	mov 0x8(%ebp),%eax
0x13f4 <primeiraBomba+203>	add \$0xb,%eax
0x13f7 <primeiraBomba+206>	movzbl (%eax),%eax

0x13fa <primeiraBomba+209>	cmp \$0x20,%al
0x13fc <primeiraBomba+211>	jne 0x144c <primeiraBomba+29>
0x13fe <primeiraBomba+213>	mov 0x8(%ebp),%eax
0x1401 <primeiraBomba+216>	add \$0xc,%eax
0x1404 <primeiraBomba+219>	movzbl (%eax),%eax
0x1407 <primeiraBomba+222>	cmp \$0x32,%al
0x1409 <primeiraBomba+224>	jne 0x144c <primeiraBomba+29>
0x140b <primeiraBomba+226>	mov 0x8(%ebp),%eax
0x140e <primeiraBomba+229>	add \$0xd,%eax
0x1411 <primeiraBomba+232>	movzbl (%eax),%eax
0x1414 <primeiraBomba+235>	cmp \$0x30,%al
0x1416 <primeiraBomba+237>	jne 0x144c <primeiraBomba+29>
0x1418 <primeiraBomba+239>	mov 0x8(%ebp),%eax
0x141b <primeiraBomba+242>	add \$0xe,%eax
0x141e <primeiraBomba+245>	movzbl (%eax),%eax
0x1421 <primeiraBomba+248>	cmp \$0x32,%al
0x1423 <primeiraBomba+250>	jne 0x144c <primeiraBomba+29>
0x1425 <primeiraBomba+252>	mov 0x8(%ebp),%eax
0x1428 <primeiraBomba+255>	add \$0xf,%eax
0x142b <primeiraBomba+258>	movzbl (%eax),%eax
0x142e <primeiraBomba+261>	cmp \$0x30,%al
0x1430 <primeiraBomba+263>	jne 0x144c <primeiraBomba+29>
0x1432 <primeiraBomba+265>	mov 0x8(%ebp),%eax
0x1435 <primeiraBomba+268>	add \$0x10,%eax
0x1438 <primeiraBomba+271>	movzbl (%eax),%eax
0x143b <primeiraBomba+274>	cmp \$0x2e,%al
0x143d <primeiraBomba+276>	jne 0x144c <primeiraBomba+29>
0x143f <primeiraBomba+278>	mov 0x8(%ebp),%eax
0x1442 <primeiraBomba+281>	add \$0x11,%eax
0x1445 <primeiraBomba+284>	movzbl (%eax),%eax
0x1448 <primeiraBomba+287>	cmp \$0x32,%al
0x144a <primeiraBomba+289>	je 0x1451 <primeiraBomba+296>
0x144c <primeiraBomba+291>	call 0x122d <boom>
0x1451 <primeiraBomba+296>	sub \$0xc,%esp
0x1454 <primeiraBomba+299>	lea -0x1f44(%ebx),%eax
0x145a <primeiraBomba+305>	push %eax
0x145b <primeiraBomba+306>	call 0x10a0 <puts@plt>
0x1460 <primeiraBomba+311>	add \$0x10,%esp
0x1463 <primeiraBomba+314>	nop
0x1464 <primeiraBomba+315>	mov -0x4(%ebp),%ebx
0x1467 <primeiraBomba+318>	leave
0x1468 <primeiraBomba+319>	ret
0x1469 <segundaBomba>	push %ebp
0x146a <segundaBomba+1>	mov %esp,%ebp

0x146c <segundaBomba+3>	push %ebx
0x146d <segundaBomba+4>	sub \$0x64,%esp
0x1470 <segundaBomba+7>	call 0x1130 <__x86.get_pc_thunk.bx>
0x1475 <segundaBomba+12>	add \$0x2b8b,%ebx
0x147b <segundaBomba+18>	mov 0x8(%ebp),%eax
0x147e <segundaBomba+21>	mov %eax,-0x5c(%ebp)
0x1481 <segundaBomba+24>	mov %gs:0x14,%eax
0x1487 <segundaBomba+30>	mov %eax,-0xc(%ebp)
0x148a <segundaBomba+33>	xor %eax,%eax
0x148c <segundaBomba+35>	movl \$0x2062614c,-0x4c(%ebp)
0x1493 <segundaBomba+42>	movl \$0x42203a32,-0x48(%ebp)
0x149a <segundaBomba+49>	movl \$0x626d6f,-0x44(%ebp)
0x14a1 <segundaBomba+56>	movl \$0x0,-0x40(%ebp)
0x14a8 <segundaBomba+63>	movl \$0x0,-0x3c(%ebp)
0x14af <segundaBomba+70>	movl \$0x0,-0x38(%ebp)
0x14b6 <segundaBomba+77>	movl \$0x0,-0x34(%ebp)
0x14bd <segundaBomba+84>	movl \$0x0,-0x30(%ebp)
0x14c4 <segundaBomba+91>	movl \$0x0,-0x2c(%ebp)
0x14cb <segundaBomba+98>	movl \$0x0,-0x28(%ebp)
0x14d2 <segundaBomba+105>	movl \$0x0,-0x24(%ebp)
0x14d9 <segundaBomba+112>	movl \$0x0,-0x20(%ebp)
0x14e0 <segundaBomba+119>	movl \$0x0,-0x1c(%ebp)
0x14e7 <segundaBomba+126>	movl \$0x0,-0x18(%ebp)
0x14ee <segundaBomba+133>	movl \$0x0,-0x14(%ebp)
0x14f5 <segundaBomba+140>	movl \$0x0,-0x10(%ebp)
0x14fc <segundaBomba+147>	sub \$0x8,%esp
0x14ff <segundaBomba+150>	pushl -0x5c(%ebp)
0x1502 <segundaBomba+153>	lea -0x4c(%ebp),%eax
0x1505 <segundaBomba+156>	push %eax
0x1506 <segundaBomba+157>	call 0x1040 <strcmp@plt>
0x150b <segundaBomba+162>	add \$0x10,%esp
0x150e <segundaBomba+165>	test %eax,%eax
0x1510 <segundaBomba+167>	je 0x1517 <segundaBomba+174>
0x1512 <segundaBomba+169>	call 0x122d <boom>
0x1517 <segundaBomba+174>	sub \$0xc,%esp
0x151a <segundaBomba+177>	lea -0x1f20(%ebx),%eax
0x1520 <segundaBomba+183>	push %eax
0x1521 <segundaBomba+184>	call 0x10a0 <puts@plt>
0x1526 <segundaBomba+189>	add \$0x10,%esp
0x1529 <segundaBomba+192>	nop
0x152a <segundaBomba+193>	mov -0xc(%ebp),%eax
0x152d <segundaBomba+196>	sub %gs:0x14,%eax
0x1534 <segundaBomba+203>	je 0x153b <segundaBomba+210>
0x1536 <segundaBomba+205>	call 0x1820 <__stack_chk_fail_local>
0x153b <segundaBomba+210>	mov -0x4(%ebp),%ebx
0x153e <segundaBomba+213>	leave

0x153f <segundaBomba+214>	ret
0x1540 <terceiraBomba>	push %ebp
0x1541 <terceiraBomba+1>	mov %esp,%ebp
0x1543 <terceiraBomba+3>	push %ebx
0x1544 <terceiraBomba+4>	sub \$0x14,%esp
0x1547 <terceiraBomba+7>	call 0x1130 <__x86.get_pc_thunk.bx>
0x154c <terceiraBomba+12>	add \$0x2ab4,%ebx
0x1552 <terceiraBomba+18>	lea -0x1f00(%ebx),%eax
0x1558 <terceiraBomba+24>	mov %eax,-0xc(%ebp)
0x155b <terceiraBomba+27>	sub \$0x8,%esp
0x155e <terceiraBomba+30>	pushl 0x8(%ebp)
0x1561 <terceiraBomba+33>	pushl -0xc(%ebp)
0x1564 <terceiraBomba+36>	call 0x1040 <strcmp@plt>
0x1569 <terceiraBomba+41>	add \$0x10,%esp
0x156c <terceiraBomba+44>	test %eax,%eax
0x156e <terceiraBomba+46>	jne 0x1577 <terceiraBomba+55>
0x1570 <terceiraBomba+48>	call 0x122d <boom>
0x1575 <terceiraBomba+53>	jmp 0x15e9 <terceiraBomba+169>
0x1577 <terceiraBomba+55>	sub \$0x8,%esp
0x157a <terceiraBomba+58>	pushl 0x8(%ebp)
0x157d <terceiraBomba+61>	lea -0x1efa(%ebx),%eax
0x1583 <terceiraBomba+67>	push %eax
0x1584 <terceiraBomba+68>	call 0x1040 <strcmp@plt>
0x1589 <terceiraBomba+73>	add \$0x10,%esp
0x158c <terceiraBomba+76>	test %eax,%eax
0x158e <terceiraBomba+78>	jne 0x1597 <terceiraBomba+87>
0x1590 <terceiraBomba+80>	call 0x122d <boom>
0x1595 <terceiraBomba+85>	jmp 0x15e9 <terceiraBomba+169>
0x1597 <terceiraBomba+87>	sub \$0x8,%esp
0x159a <terceiraBomba+90>	pushl 0x8(%ebp)
0x159d <terceiraBomba+93>	lea -0x1eec(%ebx),%eax
0x15a3 <terceiraBomba+99>	push %eax
0x15a4 <terceiraBomba+100>	call 0x1040 <strcmp@plt>
0x15a9 <terceiraBomba+105>	add \$0x10,%esp
0x15ac <terceiraBomba+108>	test %eax,%eax
0x15ae <terceiraBomba+110>	jne 0x15b7 <terceiraBomba+119>
0x15b0 <terceiraBomba+112>	call 0x122d <boom>
0x15b5 <terceiraBomba+117>	jmp 0x15e9 <terceiraBomba+169>
0x15b7 <terceiraBomba+119>	sub \$0x8,%esp
0x15ba <terceiraBomba+122>	pushl 0x8(%ebp)
0x15bd <terceiraBomba+125>	lea -0x1edc(%ebx),%eax
0x15c3 <terceiraBomba+131>	push %eax
0x15c4 <terceiraBomba+132>	call 0x1040 <strcmp@plt>
0x15c9 <terceiraBomba+137>	add \$0x10,%esp
0x15cc <terceiraBomba+140>	test %eax,%eax

0x15ce <terceiraBomba+142>	jne 0x15e4 <terceiraBomba+164>
0x15d0 <terceiraBomba+144>	sub \$0xc,%esp
0x15d3 <terceiraBomba+147>	lea -0x1ebc(%ebx),%eax
0x15d9 <terceiraBomba+153>	push %eax
0x15da <terceiraBomba+154>	call 0x10a0 <puts@plt>
0x15df <terceiraBomba+159>	add \$0x10,%esp
0x15e2 <terceiraBomba+162>	jmp 0x15e9 <terceiraBomba+169>
0x15e4 <terceiraBomba+164>	call 0x122d <boom>
0x15e9 <terceiraBomba+169>	nop
0x15ea <terceiraBomba+170>	mov -0x4(%ebp),%ebx
0x15ed <terceiraBomba+173>	leave
0x15ee <terceiraBomba+174>	ret
0x15ef <quartaBomba>	push %ebp
0x15f0 <quartaBomba+1>	mov %esp,%ebp
0x15f2 <quartaBomba+3>	push %ebx
0x15f3 <quartaBomba+4>	sub \$0x824,%esp
0x15f9 <quartaBomba+10>	call 0x1130 <__x86.get_pc_thunk.bx>
0x15fe <quartaBomba+15>	add \$0x2a02,%ebx
0x1604 <quartaBomba+21>	mov 0x8(%ebp),%eax
0x1607 <quartaBomba+24>	mov %eax,-0x81c(%ebp)
0x160d <quartaBomba+30>	mov %gs:0x14,%eax
0x1613 <quartaBomba+36>	mov %eax,-0xc(%ebp)
0x1616 <quartaBomba+39>	xor %eax,%eax
0x1618 <quartaBomba+41>	sub \$0x8,%esp
0x161b <quartaBomba+44>	lea -0x1e9b(%ebx),%eax
0x1621 <quartaBomba+50>	push %eax
0x1622 <quartaBomba+51>	lea -0x1fe3(%ebx),%eax
0x1628 <quartaBomba+57>	push %eax
0x1629 <quartaBomba+58>	call 0x10e0 <fopen@plt>
0x162e <quartaBomba+63>	add \$0x10,%esp
0x1631 <quartaBomba+66>	mov %eax,-0x810(%ebp)
0x1637 <quartaBomba+72>	cmpl \$0x0,-0x810(%ebp)
0x163e <quartaBomba+79>	je 0x167a <quartaBomba+139>
0x1640 <quartaBomba+81>	sub \$0x4,%esp
0x1643 <quartaBomba+84>	pushl -0x810(%ebp)
0x1649 <quartaBomba+90>	push \$0x800
0x164e <quartaBomba+95>	lea -0x80c(%ebp),%eax
0x1654 <quartaBomba+101>	push %eax
0x1655 <quartaBomba+102>	call 0x1050 <fgets@plt>
0x165a <quartaBomba+107>	add \$0x10,%esp
0x165d <quartaBomba+110>	sub \$0xc,%esp
0x1660 <quartaBomba+113>	lea -0x80c(%ebp),%eax
0x1666 <quartaBomba+119>	push %eax
0x1667 <quartaBomba+120>	call 0x10c0 <strlen@plt>

0x166c <quartaBomba+125>	add \$0x10,%esp
0x166f <quartaBomba+128>	sub \$0x1,%eax
0x1672 <quartaBomba+131>	movb \$0x0,-0x80c(%ebp,%eax,1)
0x167a <quartaBomba+139>	sub \$0xc,%esp
0x167d <quartaBomba+142>	pushl -0x810(%ebp)
0x1683 <quartaBomba+148>	call 0x1060 <fclose@plt>
0x1688 <quartaBomba+153>	add \$0x10,%esp
0x168b <quartaBomba+156>	sub \$0x8,%esp
0x168e <quartaBomba+159>	pushl -0x81c(%ebp)
0x1694 <quartaBomba+165>	lea -0x80c(%ebp),%eax
0x169a <quartaBomba+171>	push %eax
0x169b <quartaBomba+172>	call 0x1040 <strcmp@plt>
0x16a0 <quartaBomba+177>	add \$0x10,%esp
0x16a3 <quartaBomba+180>	test %eax,%eax
0x16a5 <quartaBomba+182>	je 0x16ac <quartaBomba+189>
0x16a7 <quartaBomba+184>	call 0x122d <boom>
0x16ac <quartaBomba+189>	sub \$0xc,%esp
0x16af <quartaBomba+192>	lea -0x1e98(%ebx),%eax
0x16b5 <quartaBomba+198>	push %eax
0x16b6 <quartaBomba+199>	call 0x10a0 <puts@plt>
0x16bb <quartaBomba+204>	add \$0x10,%esp
0x16be <quartaBomba+207>	nop
0x16bf <quartaBomba+208>	mov -0xc(%ebp),%eax
0x16c2 <quartaBomba+211>	sub %gs:0x14,%eax
0x16c9 <quartaBomba+218>	je 0x16d0 <quartaBomba+225>
0x16cb <quartaBomba+220>	call 0x1820 <__stack_chk_fail_local>
0x16d0 <quartaBomba+225>	mov -0x4(%ebp),%ebx
0x16d3 <quartaBomba+228>	leave
0x16d4 <quartaBomba+229>	ret
0x16d5 <main>	lea 0x4(%esp),%ecx
0x16d9 <main+4>	and \$0xffffffff0,%esp
0x16dc <main+7>	pushl -0x4(%ecx)
0x16df <main+10>	push %ebp
0x16e0 <main+11>	mov %esp,%ebp
0x16e2 <main+13>	push %esi
0x16e3 <main+14>	push %ebx
0x16e4 <main+15>	push %ecx
0x16e5 <main+16>	sub \$0xc,%esp
0x16e8 <main+19>	call 0x179a <__x86.get_pc_thunk.si>
0x16ed <main+24>	add \$0x2913,%esi
0x16f3 <main+30>	mov %ecx,%ebx
0x16f5 <main+32>	mov 0x4(%ebx),%eax
0x16f8 <main+35>	mov (%eax),%eax
0x16fa <main+37>	sub \$0xc,%esp
0x16fd <main+40>	push %eax

0x16fe <main+41>	call 0x125b <inicializa>
0x1703 <main+46>	add \$0x10,%esp
0x1706 <main+49>	cmpl \$0x1,(%ebx)
0x1709 <main+52>	jle 0x1785 <main+176>
0x170b <main+54>	mov 0x4(%ebx),%eax
0x170e <main+57>	add \$0x4,%eax
0x1711 <main+60>	mov (%eax),%eax
0x1713 <main+62>	sub \$0xc,%esp
0x1716 <main+65>	push %eax
0x1717 <main+66>	call 0x1329 <primeiraBomba>
0x171c <main+71>	add \$0x10,%esp
0x171f <main+74>	cmpl \$0x2,(%ebx)
0x1722 <main+77>	jle 0x1785 <main+176>
0x1724 <main+79>	mov 0x4(%ebx),%eax
0x1727 <main+82>	add \$0x8,%eax
0x172a <main+85>	mov (%eax),%eax
0x172c <main+87>	sub \$0xc,%esp
0x172f <main+90>	push %eax
0x1730 <main+91>	call 0x1469 <segundaBomba>
0x1735 <main+96>	add \$0x10,%esp
0x1738 <main+99>	cmpl \$0x3,(%ebx)
0x173b <main+102>	jle 0x1785 <main+176>
0x173d <main+104>	mov 0x4(%ebx),%eax
0x1740 <main+107>	add \$0xc,%eax
0x1743 <main+110>	mov (%eax),%eax
0x1745 <main+112>	sub \$0xc,%esp
0x1748 <main+115>	push %eax
0x1749 <main+116>	call 0x1540 <terceiraBomba>
0x174e <main+121>	add \$0x10,%esp
0x1751 <main+124>	cmpl \$0x4,(%ebx)
0x1754 <main+127>	jle 0x1785 <main+176>
0x1756 <main+129>	mov 0x4(%ebx),%eax
0x1759 <main+132>	add \$0x10,%eax
0x175c <main+135>	mov (%eax),%eax
0x175e <main+137>	sub \$0xc,%esp
0x1761 <main+140>	push %eax
0x1762 <main+141>	call 0x15ef <quartaBomba>
0x1767 <main+146>	add \$0x10,%esp
0x176a <main+149>	sub \$0xc,%esp
0x176d <main+152>	lea -0x1e78(%esi),%eax
0x1773 <main+158>	push %eax
0x1774 <main+159>	mov %esi,%ebx
0x1776 <main+161>	call 0x10a0 <puts@plt>
0x177b <main+166>	add \$0x10,%esp
0x177e <main+169>	mov \$0x0,%eax
0x1783 <main+174>	jmp 0x178f <main+186>

0x1785 <main+176>	call 0x122d <boom>
0x178a <main+181>	mov \$0x0,%eax
0x178f <main+186>	lea -0xc(%ebp),%esp
0x1792 <main+189>	pop %ecx
0x1793 <main+190>	pop %ebx
0x1794 <main+191>	pop %esi
0x1795 <main+192>	pop %ebp
0x1796 <main+193>	lea -0x4(%ecx),%esp
0x1799 <main+196>	ret

Para analisar melhor o funcionamento do programa, vamos colocar breakpoints no início de cada rotina com o comando break (rotina)

```

luiz@Jarvis: ~/Área de Trabalho/labs comp prog/lab2/02-bo...
<http://www.gnu.org/software/gdb/bugs/>.
Find the GDB manual and other documentation resources online at:
  <http://www.gnu.org/software/gdb/documentation/>.

For help, type "help".
Type "apropos word" to search for commands related to "word"...
Reading symbols from bomb...
(No debugging symbols found in bomb)
(gdb) break main
Ponto de parada 1 at 0x16e5
(gdb) break inicializa
Ponto de parada 2 at 0x125f
(gdb) break primeiraBomba
Ponto de parada 3 at 0x132d
(gdb) break segundaBomba
Ponto de parada 4 at 0x146d
(gdb) break terceiraBomba
Ponto de parada 5 at 0x1544
(gdb) break quartaBomba
Ponto de parada 6 at 0x15f3
(gdb) break boom
Ponto de parada 7 at 0x1231
(gdb)

```

Após os breakpoints terem sido colocados, vamos rodar o programa com o comando run, sem passar parâmetros, e a cada rotina vamos passar o comando continue, para continuar a execução do mesmo.

```
luiz@Jarvis: ~/Área de Trabalho/labs comp prog/lab2/02-bo...
(gdb) break terceiraBomba
Ponto de parada 5 at 0x1544
(gdb) break quartaBomba
Ponto de parada 6 at 0x15f3
(gdb) break boom
Ponto de parada 7 at 0x1231
(gdb) run
Starting program: /home/luiz/Área de Trabalho/labs comp prog/lab2/02-bomb/bomb

Breakpoint 1, 0x565566e5 in main ()
(gdb) continue
Continuing.

Breakpoint 2, 0x5655625f in inicializa ()
(gdb) continue
Continuing.
c
Breakpoint 7, 0x56556231 in boom ()
(gdb) continue
Continuing.
Explodiu!
[Inferior 1 (process 263226) exited with code 01]
(gdb)
```

Aqui vemos que, como não foi passado nenhuma senha na hora da execução do programa, as bombas explodem. E também descobrimos que é a rotina boom quem explode as bombas, visto que foi na execução dela que a mensagem “Explodiu!” foi retornada.

MAIN

Vamos começar agora, analisando o programa pela main até a chamada da próxima rotina pois será mais didático. Depois seguiremos para as próximas rotinas.

```
lea 0x4(%esp),%ecx
O registrador %ecx está recebendo o primeiro parâmetro da função main
que é argc
and $0xffffffff0,%esp
Estamos fazendo um “and” de 0xffffffff0 = (-16) com %esp, ou seja,
estamos alinhando o topo da pilha em um múltiplo de 16.
pushl -0x4(%ecx)
Estamos colocando o conteúdo que está em %esp - 4, a RIP do SO, no topo
da pilha.
push %ebp
Colocamos o registrador %ebp no topo da pilha.
mov %esp,%ebp
Estamos fazendo %esp apontar para %ebp, ou seja, estamos criando um
registro de ativação da função main.
```

```

push    %esi
Estamos salvando o registrador %esi na pilha
push    %ebx
Estamos salvando o registrador %ebx na pilha
push    %ecx
Estamos salvando o registrador %ecx na pilha
sub     $0xc,%esp
Como 0xc = 12, estamos fazendo sub $12,esp , isto é, estamos aumentando
3 espaços na pilha de 4 bytes
call    0x179a <__x86.get_pc_thunk.si>
O registrador %esi pega o valor do program counter (PC) que é a posição
em memória da instrução seguinte que será executada.
add     $0x2913,%esi
Adicionamos o valor 0x2913 (10515) no registrador %esi
mov     %ecx,%ebx
O registrador %ebx recebe a posição de memória que %ecx aponta, ou
seja, ele irá receber o primeiro argumento da função.
mov     0x4(%ebx),%eax
O registrador %eax vai receber o conteúdo de %ebx + 0x4, ou seja, o
segundo parâmetro da função (argv).
mov     (%eax),%eax
O registrador %eax vai receber o valor de argv[0].
sub     $0xc,%esp
Iremos abrir 3 espaços na pilha
push    %eax
Colocamos o valor de argv[0] no topo da pilha.
call    0x125b <inicializa>
Chamamos a função "inicializa"

```

A pilha de memória fica assim após essa sequência de comandos:

Endereço	Conteúdo da Pilha <4 bytes>	Comentários
%ebp		Base da pilha
...	...	
%ecx+4	argv	Segundo parâmetro da main
%ecx	argc	Primeiro parâmetro da main
%ecx - 4	RIP SO	end de retorno a SO
	...	espaço de alinhamento
%esp+52 = %esp+8 (nx16)		Alinhamento com múltiplo de 16
%esp+48 = %esp+4	RIP SO	end de retorno SO
%esp+44 = %ebp	OFP	Base da main
%esp + 40 = %ebp - 4	%esi	

%esp + 36 = %ebp - 8(nx16)	%ebx	
%esp + 32 = %ebp - 12	%ecx	
%esp + 28 = %ebp - 16		
%esp+24 = %ebp - 20		
%esp +20 = %ebp - 24(nx16)		
%esp + 16 = %ebp - 28		
%esp + 12 = %ebp - 32		
%esp +8 = %ebp - 36		
%esp +4 = %ebp - 40(nx16)	%eax = arg[0]	Primeiro parâmetro de inicialização
%esp = %ebp - 44	RIP Main	End de retorno à Main

INICIALIZA

Vamos analisar agora o código quando a função “inicializa” é chamada. A sub-rotina “inicializa” vai servir como uma função de verificação e inicialização, onde tenta abrir o arquivo e faz o tratamento caso dê errado, como podemos ver nas linhas a seguir:

```

0x125b <inicializa>      push    %ebp
0x125c <inicializa+1>    mov     %esp,%ebp
0x125e <inicializa+3>    push    %ebx
0x125f <inicializa+4>    sub     $0x824,%esp
0x1265 <inicializa+10>   call    0x1130 <__x86.get_pc_thunk.bx>
0x126a <inicializa+15>   add     $0x2d96,%ebx
0x1270 <inicializa+21>   mov     0x8(%ebp),%eax
0x1273 <inicializa+24>   mov     %eax,-0x81c(%ebp)
0x1279 <inicializa+30>   mov     %gs:0x14,%eax
0x127f <inicializa+36>   mov     %eax,-0xc(%ebp)
0x1282 <inicializa+39>   xor     %eax,%eax
0x1284 <inicializa+41>   sub     $0x8,%esp
0x1287 <inicializa+44>   lea     -0x1fe5(%ebx),%eax
0x128d <inicializa+50>   push    %eax
0x128e <inicializa+51>   lea     -0x1fe3(%ebx),%eax
0x1294 <inicializa+57>   push    %eax
0x1295 <inicializa+58>   call    0x10e0 <fopen@plt>
Tenta abrir o arquivo de entrada do programa (passado na instrução de
execução do programa)
0x129a <inicializa+63>   add     $0x10,%esp
0x129d <inicializa+66>   mov     %eax,-0x810(%ebp)
0x12a3 <inicializa+72>   cmpl    $0x0,-0x810(%ebp)
0x12aa <inicializa+79>   je      0x12f2 <inicializa+151>
0x12ac <inicializa+81>   pushl   -0x810(%ebp)
0x12b2 <inicializa+87>   push    $0x1d

```

```

0x12b4 <inicializa+89> push    $0x1
0x12b6 <inicializa+91> lea    -0x1fd8(%ebx),%eax
0x12bc <inicializa+97> push    %eax
0x12bd <inicializa+98> call    0x1090 <fwrite@plt>
Escreve na pilha
0x12c2 <inicializa+103> add     $0x10,%esp
0x12c5 <inicializa+106> sub     $0xc,%esp
0x12c8 <inicializa+109> pushl   -0x810(%ebp)
0x12ce <inicializa+115> call    0x1060 <fclose@plt>
Fechamento do ponteiro para o arquivo.
0x12d3 <inicializa+120> add     $0x10,%esp
0x12d6 <inicializa+123> sub     $0xc,%esp
0x12d9 <inicializa+126> push    $0x1
0x12db <inicializa+128> call    0x1070 <sleep@plt>
Espera 1 segundo
0x12e0 <inicializa+133> add     $0x10,%esp
0x12e3 <inicializa+136> nop
0x12e4 <inicializa+137> mov     -0xc(%ebp),%eax
0x12e7 <inicializa+140> sub     %gs:0x14,%eax
0x12ee <inicializa+147> je      0x1324 <inicializa+201>
Verifica se a proteção da pilha foi preservada
0x12f0 <inicializa+149> jmp     0x131f <inicializa+196>
0x12f2 <inicializa+151> sub     $0xc,%esp
0x12f5 <inicializa+154> lea     -0x1fb8(%ebx),%eax
0x12fb <inicializa+160> push    %eax
0x12fc <inicializa+161> call    0x10a0 <puts@plt>
Caso não consiga abrir o arquivo de forma adequada, escreve uma
mensagem de erro.
0x1301 <inicializa+166> add     $0x10,%esp
0x1304 <inicializa+169> sub     $0xc,%esp
0x1307 <inicializa+172> pushl   -0x810(%ebp)
0x130d <inicializa+178> call    0x1060 <fclose@plt>
Fecha o arquivo.
0x1312 <inicializa+183> add     $0x10,%esp
0x1315 <inicializa+186> sub     $0xc,%esp
0x1318 <inicializa+189> push    $0x1
0x131a <inicializa+191> call    0x10b0 <exit@plt>
Termina a execução do programa.
0x131f <inicializa+196> call    0x1820 <__stack_chk_fail_local>
Corrupção da pilha caso a proteção da pilha não seja preservada
0x1324 <inicializa+201> mov     -0x4(%ebp),%ebx
0x1327 <inicializa+204> leave
0x1328 <inicializa+205> ret

```

Correndo tudo bem na sub-rotina inicializa, vamos voltar para a main:

```
add    $0x10,%esp
Iremos somar 0x10 ao registrador %esp, ou seja, estamos desalocando 4
espaços na pilha (16 bytes).
cmpl    $0x1, (%ebx)
Vamos realizar uma comparação entre o registrador %ebx e 0x1, ou seja,
estamos verificando se %ebx (parâmetro argc da função) é menor ou igual
a 1 (argc <= 1).
jle 0x1785 <main+176>
Se a condição anterior for atendida, ou seja, se argc <= 1, não teremos
as senhas necessárias para desarmar as bombas. Iremos chamar a função
boom, que se encontra em <main+176>.
mov 0x4(%ebx), %eax
Estamos salvando o conteúdo do registrador %ebx + 4 (parâmetro argv) no
registrador %eax.
add $0x4, %eax
Iremos somar 4 ao endereço do registrador %eax.
mov (%eax), %eax
Passamos para o registrador %eax o valor do ponteiro argv[1].
sub $0xc, %esp
Vamos abrir 3 espaços na pilha de 4 bytes
push    %eax
Coloca no topo da pilha argv[1] como parâmetro da chamada da próxima
função.
call 0x1329 <primeiraBomba>
Chama a função "primeiraBomba"
```

A nossa pilha vai se encontrar desta forma quando a função primeiraBomba for chamada:

Endereço	Conteúdo da Pilha <4 bytes>	Comentários
%ebp		Base da pilha
...	...	
%ecx+4	argv	Segundo parâmetro da main
%ecx	argc	Primeiro parâmetro da main
%ecx - 4	RIP SO	end de retorno a SO
	...	espaço de alinhamento
%esp+52 = %esp+8 (nx16)		Alinhamento com múltiplo de 16
%esp+48 = %esp+4	RIP SO	end de retorno SO
%esp+44 = %ebp	OFP	Base da main
%esp + 40 = %ebp - 4	%esi	

%esp + 36 = %ebp - 8 (nx16)	%ebx	
%esp + 32 = %ebp - 12	%ecx	
%esp + 28 = %ebp - 16		
%esp+24 = %ebp - 20		
%esp +20 = %ebp - 24 (nx16)		
%esp + 16 = %ebp - 28		
%esp + 12 = %ebp - 32		
%esp +8 = %ebp - 36		
%esp +4 = %ebp - 40 (nx16)	%eax = arg[1]	Primeiro parâmetro da “primeiraBomba”
%esp = %ebp - 44	RIP Main	End de retorno à Main

PRIMEIRA BOMBA

Vamos analisar agora o código de montagem da primeira bomba:

```

0x1329 <primeiraBomba>      push    %ebp
0x132a <primeiraBomba+1>    mov     %esp,%ebp
0x132c <primeiraBomba+3>    push    %ebx
0x132d <primeiraBomba+4>    sub     $0x4,%esp
0x1330 <primeiraBomba+7>    call    0x1130 <__x86.get_pc_thunk.bx>
0x1335 <primeiraBomba+12>   add     $0x2ccb,%ebx
0x133b <primeiraBomba+18>   mov     0x8(%ebp),%eax
0x133e <primeiraBomba+21>   movzbl (%eax),%eax
0x1341 <primeiraBomba+24>   cmp     $0x4d,%al
0x1343 <primeiraBomba+26>   je      0x1353 <primeiraBomba+42>
0x1345 <primeiraBomba+28>   mov     0x8(%ebp),%eax
0x1348 <primeiraBomba+31>   movzbl (%eax),%eax
0x134b <primeiraBomba+34>   cmp     $0x6d,%al
0x134d <primeiraBomba+36>   jne     0x144c <primeiraBomba+291>
0x1353 <primeiraBomba+42>   mov     0x8(%ebp),%eax
0x1356 <primeiraBomba+45>   add     $0x1,%eax
0x1359 <primeiraBomba+48>   movzbl (%eax),%eax
0x135c <primeiraBomba+51>   cmp     $0x61,%al
0x135e <primeiraBomba+53>   jne     0x144c <primeiraBomba+291>
0x1364 <primeiraBomba+59>   mov     0x8(%ebp),%eax
0x1367 <primeiraBomba+62>   add     $0x2,%eax
0x136a <primeiraBomba+65>   movzbl (%eax),%eax
0x136d <primeiraBomba+68>   cmp     $0x62,%al
0x136f <primeiraBomba+70>   jne     0x144c <primeiraBomba+291>
0x1375 <primeiraBomba+76>   mov     0x8(%ebp),%eax
0x1378 <primeiraBomba+79>   add     $0x3,%eax
0x137b <primeiraBomba+82>   movzbl (%eax),%eax

```

```
0x137e <primeiraBomba+85>      cmp $0x20,%al
0x1380 <primeiraBomba+87>      jne 0x144c <primeiraBomba+291>
0x1386 <primeiraBomba+93>      mov 0x8(%ebp),%eax
0x1389 <primeiraBomba+96>      add $0x4,%eax
0x138c <primeiraBomba+99>      movzbl (%eax),%eax
0x138f <primeiraBomba+102>     cmp $0x2d,%al
0x1391 <primeiraBomba+104>     jne 0x144c <primeiraBomba+291>
0x1397 <primeiraBomba+110>     mov 0x8(%ebp),%eax
0x139a <primeiraBomba+113>     add $0x5,%eax
0x139d <primeiraBomba+116>     movzbl (%eax),%eax
0x13a0 <primeiraBomba+119>     cmp $0x20,%al
0x13a2 <primeiraBomba+121>     jne 0x144c <primeiraBomba+291>
0x13a8 <primeiraBomba+127>     mov 0x8(%ebp),%eax
0x13ab <primeiraBomba+130>     add $0x6,%eax
0x13ae <primeiraBomba+133>     movzbl (%eax),%eax
0x13b1 <primeiraBomba+136>     cmp $0x33,%al
0x13b3 <primeiraBomba+138>     jne 0x144c <primeiraBomba+291>
0x13b9 <primeiraBomba+144>     mov 0x8(%ebp),%eax
0x13bc <primeiraBomba+147>     add $0x7,%eax

0x13bf <primeiraBomba+150>     movzbl (%eax),%eax
0x13c2 <primeiraBomba+153>     cmp $0x35,%al
0x13c4 <primeiraBomba+155>     jne 0x144c <primeiraBomba+291>
0x13ca <primeiraBomba+161>     mov 0x8(%ebp),%eax
0x13cd <primeiraBomba+164>     add $0x8,%eax
0x13d0 <primeiraBomba+167>     movzbl (%eax),%eax
0x13d3 <primeiraBomba+170>     cmp $0x33,%al
0x13d5 <primeiraBomba+172>     jne 0x144c <primeiraBomba+291>
0x13d7 <primeiraBomba+174>     mov 0x8(%ebp),%eax
0x13da <primeiraBomba+177>     add $0x9,%eax
0x13dd <primeiraBomba+180>     movzbl (%eax),%eax
0x13e0 <primeiraBomba+183>     cmp $0x20,%al
0x13e2 <primeiraBomba+185>     jne 0x144c <primeiraBomba+291>
0x13e4 <primeiraBomba+187>     mov 0x8(%ebp),%eax
0x13e7 <primeiraBomba+190>     add $0xa,%eax
0x13ea <primeiraBomba+193>     movzbl (%eax),%eax
0x13ed <primeiraBomba+196>     cmp $0x2d,%al
0x13ef <primeiraBomba+198>     jne 0x144c <primeiraBomba+291>
0x13f1 <primeiraBomba+200>     mov 0x8(%ebp),%eax
0x13f4 <primeiraBomba+203>     add $0xb,%eax
0x13f7 <primeiraBomba+206>     movzbl (%eax),%eax
0x13fa <primeiraBomba+209>     cmp $0x20,%al
0x13fc <primeiraBomba+211>     jne 0x144c <primeiraBomba+291>
```

```

0x13fe <primeiraBomba+213>    mov 0x8(%ebp),%eax
0x1401 <primeiraBomba+216>    add $0xc,%eax
0x1404 <primeiraBomba+219>    movzbl (%eax),%eax
0x1407 <primeiraBomba+222>    cmp $0x32,%al
0x1409 <primeiraBomba+224>    jne 0x144c <primeiraBomba+291>
0x140b <primeiraBomba+226>    mov 0x8(%ebp),%eax
0x140e <primeiraBomba+229>    add $0xd,%eax
0x1411 <primeiraBomba+232>    movzbl (%eax),%eax

0x1414 <primeiraBomba+235>    cmp $0x30,%al
0x1416 <primeiraBomba+237>    jne 0x144c <primeiraBomba+291>
0x1418 <primeiraBomba+239>    mov 0x8(%ebp),%eax
0x141b <primeiraBomba+242>    add $0xe,%eax
0x141e <primeiraBomba+245>    movzbl (%eax),%eax
0x1421 <primeiraBomba+248>    cmp $0x32,%al
0x1423 <primeiraBomba+250>    jne 0x144c <primeiraBomba+291>
0x1425 <primeiraBomba+252>    mov 0x8(%ebp),%eax
0x1428 <primeiraBomba+255>    add $0xf,%eax
0x142b <primeiraBomba+258>    movzbl (%eax),%eax

0x142e <primeiraBomba+261>    cmp $0x30,%al
0x1430 <primeiraBomba+263>    jne 0x144c <primeiraBomba+291>
0x1432 <primeiraBomba+265>    mov 0x8(%ebp),%eax
0x1435 <primeiraBomba+268>    add $0x10,%eax
0x1438 <primeiraBomba+271>    movzbl (%eax),%eax
0x143b <primeiraBomba+274>    cmp $0x2e,%al
0x143d <primeiraBomba+276>    jne 0x144c <primeiraBomba+291>
0x143f <primeiraBomba+278>    mov 0x8(%ebp),%eax
0x1442 <primeiraBomba+281>    add $0x11,%eax
0x1445 <primeiraBomba+284>    movzbl (%eax),%eax

0x1448 <primeiraBomba+287>    cmp $0x32,%al
0x144a <primeiraBomba+289>    je 0x1451 <primeiraBomba+296>
0x144c <primeiraBomba+291>    call 0x122d <boom>
0x1451 <primeiraBomba+296>    sub $0xc,%esp
0x1454 <primeiraBomba+299>    lea -0x1f44(%ebx),%eax
0x145a <primeiraBomba+305>    push %eax
0x145b <primeiraBomba+306>    call 0x10a0 <puts@plt>
0x1460 <primeiraBomba+311>    add $0x10,%esp
0x1463 <primeiraBomba+314>    nop
0x1464 <primeiraBomba+315>    mov -0x4(%ebp),%ebx
0x1467 <primeiraBomba+318>    leave
0x1468 <primeiraBomba+319>    ret

```

De forma resumida, o que temos neste código de montagem é:

Como no início do programa nós temos um registro de ativação sendo criado, podemos ter uma ideia de que o programa foi compilado sem otimização.

Recebemos o valor do program counter + 0x2ccb , no comando **add \$0x2ccb,%ebx**.

Em seguida temos o comando movzbl, que estende com 0 um byte em 32 bits. O que nos interessa é o último byte , visto que o restante é 0. Esse valor é alocado no registrador %eax.

O byte que foi alocado (%al) será então comparado com outros bytes com o comando cmp, que faz a diferença entre dois números hexadecimais. Caso o resultado da comparação seja diferente de 0 vamos para um desvio onde a função boom é chamada e a bomba explode.

O que percebemos é que o comando cmp está comparando um caracter com outro e que caso a comparação seja diferente de 0, a bomba é acionada. Logo o número que estamos comparando com o char é o char esperado para formar a senha para desarmar a bomba, de acordo com a tabela ASCII.

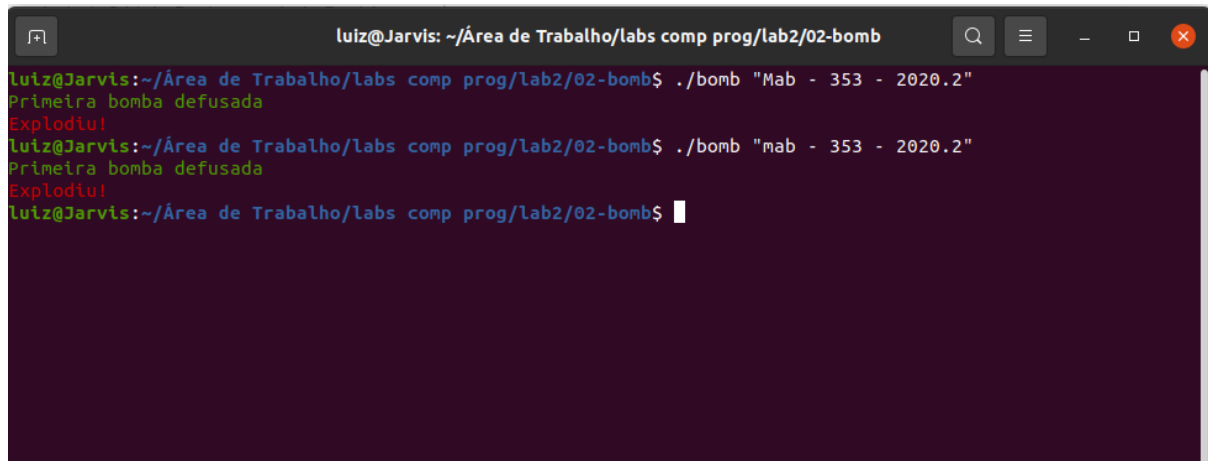
O primeiro cmp verifica se o primeiro caractere é um M maiúsculo, caso seja, haverá um desvio onde não leremos o m minúsculo, que também será aceito na senha.

Aqui temos os comandos relacionados com os valores da tabela ASCII:

```
cmp $0x4d,%al, 0x4d = M
cmp $0x6d,%al, 0x6d = m
cmp $0x61,%al, 0x61 = a
cmp $0x62,%al, 0x62 = b
cmp $0x20,%al, 0x20 = (espaço)
cmp $0x2d,%al, 0x2d = -
cmp $0x20,%al, 0x20 = (espaço)
cmp $0x33,%al, 0x33 = 3
cmp $0x35,%al, 0x35 = 5
cmp $0x33,%al, 0x33 = 3
cmp $0x20,%al, 0x20 = (espaço)
cmp $0x2d,%al, 0x2d = -
cmp $0x20,%al, 0x20 = (espaço)
cmp $0x32,%al, 0x32 = 2
cmp $0x30,%al, 0x30 = 0
cmp $0x32,%al, 0x32 = 2
```

```
cmp $0x30,%a1, 0x30 = 0
cmp $0x2e,%a1, 0x2e = .
cmp $0x32,%a1, 0x32 = 2
```

Vamos testar então se “**Mab - 353 - 2020.2**” e/ou “**mab - 353 - 2020.2**” servem como senhas para desarmar a primeira bomba.



```
luiz@Jarvis: ~/Área de Trabalho/labs comp prog/lab2/02-bomb
luiz@Jarvis:~/Área de Trabalho/labs comp prog/lab2/02-bomb$ ./bomb "Mab - 353 - 2020.2"
Primeira bomba defusada
Explodiu!
luiz@Jarvis:~/Área de Trabalho/labs comp prog/lab2/02-bomb$ ./bomb "mab - 353 - 2020.2"
Primeira bomba defusada
Explodiu!
luiz@Jarvis:~/Área de Trabalho/labs comp prog/lab2/02-bomb$
```

Como podemos ver a primeira bomba foi desarmada tanto com m maiúsculo quanto m minúsculo, entretanto a mensagem de “Explodiu!”, continua sendo retornada visto que ainda precisamos desarmar mais 3 bombas.

Seguimos então para a rotina de volta para a main, de onde paramos.

```
add $0x10,%esp
Iremos adicionar 0x10 ao registrador %esp, ou seja, estamos desalocando
4 espaços de 4 bytes da pilha.
cmpl    $0x2, (%ebx)
Iremos comparar %ebx com 0x2, ou seja, estamos verificando se argc é
menor ou igual a 2.
jle 0x1785 <main+176>
Se a comparação anterior for atendida, não teremos todas as senhas
necessárias para desarmar a bomba. Logo, iremos chamar a função "boom".
mov 0x4(%ebx), %eax
Iremos salvar %ebx + 4 no registrador %eax.
add $0x8, %eax
Iremos adicionar 8 ao endereço do registrador %eax.
mov (%eax), %eax
O registrador %eax irá receber o valor do ponteiro argv[2].
sub $0xc, %esp
Iremos abrir mais 3 espaços de 4 bytes na pilha.
push    %eax
```

Coloca no topo da pilha `argv[2]` como parâmetro da chamada da próxima função.

```
call 0x1469 <segundaBomba>
```

Iremos chamar a função "segundaBomba".

Temos o estado da pilha quando a função "segundaBomba" é chamada:

Endereço	Conteúdo da Pilha <4 bytes>	Comentários
%ebp		Base da pilha
...	...	
%ecx+4	argv	Segundo parâmetro da main
%ecx	argc	Primeiro parâmetro da main
%ecx - 4	RIP SO	end de retorno a SO
	...	espaço de alinhamento
%esp+52 = %esp+8 (nx16)		Alinhamento com múltiplo de 16
%esp+48 = %esp+4	RIP SO	end de retorno SO
%esp+44 = %ebp	OFP	Base da main
%esp + 40 = %ebp - 4	%esi	
%esp + 36 = %ebp - 8(nx16)	%ebx	
%esp + 32 = %ebp - 12	%ecx	
%esp + 28 = %ebp - 16		
%esp+24 = %ebp - 20		
%esp +20 = %ebp - 24(nx16)		
%esp + 16 = %ebp - 28		
%esp + 12 = %ebp - 32		
%esp +8 = %ebp - 36		
%esp +4 = %ebp - 40(nx16)	%eax = argv[2]	Primeiro parâmetro da "segundaBomba"
%esp = %ebp - 44	RIP Main	End de retorno à Main

SEGUNDA BOMBA

Analisando o código da função segundaBomba, vamos ter:

```
0x1469 <segundaBomba>          push    %ebp
0x146a <segundaBomba+1>         mov     %esp,%ebp
0x146c <segundaBomba+3>         push    %ebx
0x146d <segundaBomba+4>         sub     $0x64,%esp
0x1470 <segundaBomba+7>         call    0x1130 <__x86.get_pc_thunk.bx>
0x1475 <segundaBomba+12>        add     $0x2b8b,%ebx
0x147b <segundaBomba+18>        mov     0x8(%ebp),%eax
0x147e <segundaBomba+21>        mov     %eax,-0x5c(%ebp)
0x1481 <segundaBomba+24>        mov     %gs:0x14,%eax
0x1487 <segundaBomba+30>        mov     %eax,-0xc(%ebp)
0x148a <segundaBomba+33>        xor     %eax,%eax
0x148c <segundaBomba+35>        movl    $0x2062614c,-0x4c(%ebp)
0x1493 <segundaBomba+42>        movl    $0x42203a32,-0x48(%ebp)
0x149a <segundaBomba+49>        movl    $0x626d6f,-0x44(%ebp)
0x14a1 <segundaBomba+56>        movl    $0x0,-0x40(%ebp)
0x14a8 <segundaBomba+63>        movl    $0x0,-0x3c(%ebp)
0x14af <segundaBomba+70>        movl    $0x0,-0x38(%ebp)
0x14b6 <segundaBomba+77>        movl    $0x0,-0x34(%ebp)
0x14bd <segundaBomba+84>        movl    $0x0,-0x30(%ebp)
0x14c4 <segundaBomba+91>        movl    $0x0,-0x2c(%ebp)
0x14cb <segundaBomba+98>        movl    $0x0,-0x28(%ebp)
0x14d2 <segundaBomba+105>       movl    $0x0,-0x24(%ebp)
0x14d9 <segundaBomba+112>       movl    $0x0,-0x20(%ebp)
0x14e0 <segundaBomba+119>       movl    $0x0,-0x1c(%ebp)
0x14e7 <segundaBomba+126>       movl    $0x0,-0x18(%ebp)
0x14ee <segundaBomba+133>       movl    $0x0,-0x14(%ebp)
0x14f5 <segundaBomba+140>       movl    $0x0,-0x10(%ebp)
0x14fc <segundaBomba+147>       sub     $0x8,%esp
0x14ff <segundaBomba+150>       pushl   -0x5c(%ebp)
0x1502 <segundaBomba+153>       lea     -0x4c(%ebp),%eax
0x1505 <segundaBomba+156>       push    %eax
0x1506 <segundaBomba+157>       call    0x1040 <strcmp@plt>
0x150b <segundaBomba+162>       add     $0x10,%esp
0x150e <segundaBomba+165>       test    %eax,%eax
0x1510 <segundaBomba+167>       je      0x1517 <segundaBomba+174>
0x1512 <segundaBomba+169>       call    0x122d <boom>
0x1517 <segundaBomba+174>       sub     $0xc,%esp
0x151a <segundaBomba+177>       lea     -0x1f20(%ebx),%eax
0x1520 <segundaBomba+183>       push    %eax
0x1521 <segundaBomba+184>       call    0x10a0 <puts@plt>
0x1526 <segundaBomba+189>       add     $0x10,%esp
```

```

0x1529 <segundaBomba+192>    nop
0x152a <segundaBomba+193>    mov     -0xc(%ebp),%eax
0x152d <segundaBomba+196>    sub     %gs:0x14,%eax
0x1534 <segundaBomba+203>    je      0x153b <segundaBomba+210>
0x1536 <segundaBomba+205>    call    0x1820 <__stack_chk_fail_local>
0x153b <segundaBomba+210>    mov     -0x4(%ebp),%ebx
0x153e <segundaBomba+213>    leave
0x153f <segundaBomba+214>    ret

```

Novamente temos o registro de ativação. Depois temos uma alocação de 100 bytes na pilha (**sub \$0x64,%esp**), %eax receberá o argv[2] (**mov 0x8(%ebp),%eax**) que **será salvo na pilha (mov %eax,-0x5c(%ebp))**.

Vamos então ter os comandos:

```

movl    $0x2062614c,-0x4c(%ebp)
movl    $0x42203a32,-0x48(%ebp)
movl    $0x626d6f,-0x44(%ebp)

```

E mais alguns movl para passar o valor 0x0 para as posições de -0x40(%ebp) até -0x10(%ebp).

Focando apenas nessa parte, vamos ter a seguinte pilha

Endereço	Pilha <4 Bytes>
%ebp	OFP
%ebp - 0x10 0x0 %ebp - 0x14	0x0 0x0
%ebp - 0x18	0x0
%ebp - 0x1c	0x0
%ebp - 0x20	0x0
%ebp - 0x24	0x0
%ebp - 0x28	0x0
%ebp - 0x2c	0x0
%ebp - 0x30	0x0

%ebp - 0x34	0x0
%ebp - 0x38	0x0
%ebp - 0x3c	0x0
%ebp - 0x40	0x0
%esp - 0x44	0x00626d6f
%esp - 0x48	0x42203a32
%esp - 0x4c	0x2062614c

Assim que a função strcmp for chamada (0x1040 <strcmp@plt>) recebendo como argumentos tanto argv[2] (pushl -0x5c(%ebp)) e a posição de memória %ebp-0x4c (lea -0x4c(%ebp), %eax e push %eax). Assim, a posição de memória -0x4c(%ebp) será interpretada como a primeira posição de um array de caracteres, cujos bytes serão:

0x4c , 0x61, 0x62, 0x20, 0x32, 0x3a, 0x20, 0x42, 0x6f, 0x6d, 0x62, 0x00

Fazendo a tradução desses números utilizando a tabela ASCII, chegamos em “**Lab 2: Bomb**”, ou seja, caso argv[2] = ” Lab 2: Bomb”, então strcmp será igual a zero e não seremos desviados para a função boom, que explodiria a segunda bomba.

Vamos testar então se “**Lab 2: Bomb**” funciona como a nossa senha.

```

luiz@Jarvis: ~/Área de Trabalho/labs comp prog/lab2/02-bomb
luiz@Jarvis:~/Área de Trabalho/labs comp prog/lab2/02-bomb$ ./bomb "Mab - 353 - 2020.2" "Lab 2: Bomb"
Primeira bomba defusada
Segunda bomba defusada
Explodiu!
luiz@Jarvis:~/Área de Trabalho/labs comp prog/lab2/02-bomb$

```

Conferimos então que Lab 2: Bomb é realmente a nossa senha, mas como podemos ver ainda está retornando a mensagem “Explodiu!”, visto que ainda precisamos descobrir 2 senhas.

Novamente, vamos seguir para a rotina de volta a main, de onde paramos.

```

add $0x10,%esp
Iremos adicionar 0x10 ao registrador %esp, ou seja, estamos desalocando
4 espaços de 4 bytes da pilha.
cmpl    $0x3, (%ebx)
Iremos realizar uma comparação entre 0x3 e %ebx, ou seja, estamos
verificando se argc <=3.

```

```
jle 0x1785 <main+176>
```

Se a condição anterior for atendida, ou seja, se não possuímos as senhas para desarmar todas as bombas, iremos chamar a função "boom".

```
mov 0x4(%ebx),%eax
```

O registrador %eax vai receber %ebx + 4.

```
add $0xc,%eax
```

Iremos somar 0xc ao registrador %eax.

```
mov (%eax),%eax
```

Aqui o registrador %eax irá receber o valor do ponteiro argv[3].

```
sub $0xc,%esp
```

Vamos abrir mais 3 espaços de 4 bytes na pilha.

```
push %eax
```

Coloca no topo da pilha argv[3] como parâmetro da chamada da próxima função.

```
call 0x1540 <terceiraBomba>
```

Vamos chamar a função "terceiraBomba".

Assim que chamamos a função "terceira bomba", a pilha se encontra assim:

Endereço	Conteúdo da Pilha <4 bytes>	Comentários
%ebp		Base da pilha
...	...	
%ecx+4	argv	Segundo parâmetro da main
%ecx	argc	Primeiro parâmetro da main
%ecx - 4	RIP SO	end de retorno a SO
	...	espaço de alinhamento
%esp+52 = %ebp+8 (nx16)		Alinhamento com múltiplo de 16
%esp+48 = %ebp+4	RIP SO	end de retorno SO
%esp+44 = %ebp	OFP	Base da main
%esp + 40 = %ebp - 4	%esi	
%esp + 36 = %ebp - 8(nx16)	%ebx	
%esp + 32 = %ebp - 12	%ecx	
%esp + 28 = %ebp - 16		
%esp+24 = %ebp - 20		
%esp +20 = %ebp - 24(nx16)		
%esp + 16 = %ebp - 28		
%esp + 12 = %ebp - 32		
%esp +8 = %ebp - 36		
%esp +4 = %ebp - 40(nx16)	%eax = argv[3]	Primeiro parâmetro da "terceiraBomba"
%esp = %ebp - 44	RIP Main	End de retorno à Main

TERCEIRA BOMBA

Analisando o código da função terceiraBomba, vamos ter:

```
0x1540 <terceiraBomba>      push    %ebp
0x1541 <terceiraBomba+1>     mov     %esp,%ebp 1
0x1543 <terceiraBomba+3>     push    %ebx
0x1544 <terceiraBomba+4>     sub     $0x14,%esp
0x1547 <terceiraBomba+7>     call    0x1130 <__x86.get_pc_thunk.bx>
0x154c <terceiraBomba+12>    add     $0x2ab4,%ebx ---
0x1552 <terceiraBomba+18>    lea     -0x1f00(%ebx),%eax
0x1558 <terceiraBomba+24>    mov     %eax,-0xc(%ebp)
0x155b <terceiraBomba+27>    sub     $0x8,%esp
0x155e <terceiraBomba+30>    pushl   0x8(%ebp)
0x1561 <terceiraBomba+33>    pushl   -0xc(%ebp)
0x1564 <terceiraBomba+36>    call    0x1040 <strcmp@plt>
0x1569 <terceiraBomba+41>    add     $0x10,%esp
0x156c <terceiraBomba+44>    test    %eax,%eax
0x156e <terceiraBomba+46>    jne     0x1577 <terceiraBomba+55>
0x1570 <terceiraBomba+48>    call    0x122d <boom>
0x1575 <terceiraBomba+53>    jmp     0x15e9 <terceiraBomba+169>
0x1577 <terceiraBomba+55>    sub     $0x8,%esp
0x157a <terceiraBomba+58>    pushl   0x8(%ebp)
0x157d <terceiraBomba+61>    lea     -0x1efa(%ebx),%eax
0x1583 <terceiraBomba+67>    push    %eax
0x1584 <terceiraBomba+68>    call    0x1040 <strcmp@plt>
0x1589 <terceiraBomba+73>    add     $0x10,%esp
0x158c <terceiraBomba+76>    test    %eax,%eax
0x158e <terceiraBomba+78>    jne     0x1597 <terceiraBomba+87>
0x1590 <terceiraBomba+80>    call    0x122d <boom>
0x1595 <terceiraBomba+85>    jmp     0x15e9 <terceiraBomba+169>
0x1597 <terceiraBomba+87>    sub     $0x8,%esp
0x159a <terceiraBomba+90>    pushl   0x8(%ebp)
0x159d <terceiraBomba+93>    lea     -0x1eec(%ebx),%eax
0x15a3 <terceiraBomba+99>    push    %eax
0x15a4 <terceiraBomba+100>    call    0x1040 <strcmp@plt>
0x15a9 <terceiraBomba+105>    add     $0x10,%esp
0x15ac <terceiraBomba+108>    test    %eax,%eax
0x15ae <terceiraBomba+110>    jne     0x15b7 <terceiraBomba+119>
0x15b0 <terceiraBomba+112>    call    0x122d <boom>
0x15b5 <terceiraBomba+117>    jmp     0x15e9 <terceiraBomba+169>
0x15b7 <terceiraBomba+119>    sub     $0x8,%esp
0x15ba <terceiraBomba+122>    pushl   0x8(%ebp)
0x15bd <terceiraBomba+125>    lea     -0x1edc(%ebx),%eax
0x15c3 <terceiraBomba+131>    push    %eax
```

```

0x15c4 <terceiraBomba+132>    call    0x1040 <strcmp@plt>
0x15c9 <terceiraBomba+137>    add     $0x10,%esp
0x15cc <terceiraBomba+140>    test    %eax,%eax
0x15ce <terceiraBomba+142>    jne     0x15e4 <terceiraBomba+164>
0x15d0 <terceiraBomba+144>    sub     $0xc,%esp
0x15d3 <terceiraBomba+147>    lea     -0x1ebc(%ebx),%eax
0x15d9 <terceiraBomba+153>    push    %eax
0x15da <terceiraBomba+154>    call    0x10a0 <puts@plt>
0x15df <terceiraBomba+159>    add     $0x10,%esp
0x15e2 <terceiraBomba+162>    jmp     0x15e9 <terceiraBomba+169>
0x15e4 <terceiraBomba+164>    call    0x122d <boom>
0x15e9 <terceiraBomba+169>    nop
0x15ea <terceiraBomba+170>    mov     -0x4(%ebp),%ebx
0x15ed <terceiraBomba+173>    leave
0x15ee <terceiraBomba+174>    ret

```

Analisando o código acima, podemos perceber a repetição de comandos lea e comandos strcmp, o que nos dá a ideia de que algumas palavras/caracteres estão sendo comparados com algum que o programa espera receber e caso essa comparação seja diferente de zero, o programa desvia para a função boom e explode e a bomba.

Colocamos então breakpoints nas funções strcmp e vamos analisar o que está sendo comparado nessas linhas.

```

0x1564 <terceiraBomba+36>    call    0x1040 <strcmp@plt>
0x1584 <terceiraBomba+68>    call    0x1040 <strcmp@plt>
0x15a4 <terceiraBomba+100>   call    0x1040 <strcmp@plt>
0x15c4 <terceiraBomba+132>   call    0x1040 <strcmp@plt>

```

Com a ajuda do gdb, colocamos os seguintes breakpoints. Depois analisamos o que estava sendo comparado.

```

luiz@Jarvis: ~/Área de Trabalho/labs comp prog/lab2/02-bomb
0x5655659d <+93>: lea     -0x1eec(%ebx),%eax
0x565565a3 <+99>: push    %eax
0x565565a4 <+100>: call    0x56556040 <strcmp@plt>
0x565565a9 <+105>: add     $0x10,%esp
0x565565ac <+108>: test    %eax,%eax
0x565565ae <+110>: jne     0x565565b7 <terceiraBomba+119>
0x565565b0 <+112>: call    0x5655622d <boom>
--Type <RET> for more, q to quit, c to continue without paging--
0x565565b5 <+117>: jmp     0x565565e9 <terceiraBomba+169>
0x565565b7 <+119>: sub     $0x8,%esp
0x565565ba <+122>: pushl   0x8(%ebp)
0x565565bd <+125>: lea     -0x1edc(%ebx),%eax
0x565565c3 <+131>: push    %eax
0x565565c4 <+132>: call    0x56556040 <strcmp@plt>
0x565565c9 <+137>: add     $0x10,%esp
0x565565cc <+140>: test    %eax,%eax
0x565565ce <+142>: jne     0x565565e4 <terceiraBomba+164>
0x565565d0 <+144>: sub     $0xc,%esp
0x565565d3 <+147>: lea     -0x1ebc(%ebx),%eax
0x565565d9 <+153>: push    %eax
0x565565da <+154>: call    0x565560a0 <puts@plt>
0x565565df <+159>: add     $0x10,%esp
0x565565e2 <+162>: jmp     0x565565e9 <terceiraBomba+169>
0x565565e4 <+164>: call    0x5655622d <boom>
0x565565e9 <+169>: nop
0x565565ea <+170>: mov     -0x4(%ebp),%ebx
0x565565ed <+173>: leave
0x565565ee <+174>: ret
End of assembler dump.
(gdb) break *0x56556564
Ponto de parada 2 at 0x56556564
(gdb) break *0x56556584
Ponto de parada 3 at 0x56556584
(gdb) break *0x565565a4
Ponto de parada 4 at 0x565565a4
(gdb) break *0x565565c4
Ponto de parada 5 at 0x565565c4
(gdb)

```

Na nossa primeira parada foi printado: ""

```
luiz@Jarvis: ~/Área de Trabalho/labs comp prog/lab2/02-bomb

0x565565ba <+122>: pushl 0x8(%ebp)
0x565565bd <+125>: lea -0x1edc(%ebx),%eax
0x565565c3 <+131>: push %eax
0x565565c4 <+132>: call 0x56556040 <strcmp@plt>
0x565565c9 <+137>: add $0x10,%esp
0x565565cc <+140>: test %eax,%eax
0x565565ce <+142>: jne 0x565565e4 <terceiraBomba+164>
0x565565d0 <+144>: sub $0xc,%esp
0x565565d3 <+147>: lea -0x1ebc(%ebx),%eax
0x565565d9 <+153>: push %eax
0x565565da <+154>: call 0x565560a0 <puts@plt>
0x565565df <+159>: add $0x10,%esp
0x565565e2 <+162>: jmp 0x565565e9 <terceiraBomba+169>
0x565565e4 <+164>: call 0x5655622d <boom>
0x565565e9 <+169>: nop
0x565565ea <+170>: mov -0x4(%ebp),%ebx
0x565565ed <+173>: leave
0x565565ee <+174>: ret
End of assembler dump.
(gdb) break *0x56556564
Ponto de parada 2 at 0x56556564
(gdb) break *0x56556584
Ponto de parada 3 at 0x56556584
(gdb) break *0x565565a4
Ponto de parada 4 at 0x565565a4
(gdb) break *0x565565c4
Ponto de parada 5 at 0x565565c4
(gdb) run "mab - 353 - 2020.2" "Lab 2: Bomb" ""
The program being debugged has been started already.
Start it from the beginning? (y or n) y
Starting program: /home/luiz/Área de Trabalho/labs comp prog/lab2/02-bomb/bomb "mab - 353 - 2020.2" "Lab 2: Bomb" ""
Primeira bomba defusada
Segunda bomba defusada

Breakpoint 2, 0x56556564 in terceiraBomba ()
(gdb) x /s ($ebp-0xc)
0xffffd08c: ""
(gdb)
```

Na segunda parada, temos: “Programação”

```
luiz@Jarvis: ~/Área de Trabalho/labs comp prog/lab2/02-bomb

0x565565ce <+142>: jne 0x565565e4 <terceiraBomba+164>
0x565565d0 <+144>: sub $0xc,%esp
0x565565d3 <+147>: lea -0x1ebc(%ebx),%eax
0x565565d9 <+153>: push %eax
0x565565da <+154>: call 0x565560a0 <puts@plt>
0x565565df <+159>: add $0x10,%esp
0x565565e2 <+162>: jmp 0x565565e9 <terceiraBomba+169>
0x565565e4 <+164>: call 0x5655622d <boom>
0x565565e9 <+169>: nop
0x565565ea <+170>: mov -0x4(%ebp),%ebx
0x565565ed <+173>: leave
0x565565ee <+174>: ret
End of assembler dump.
(gdb) break *0x56556564
Ponto de parada 2 at 0x56556564
(gdb) break *0x56556584
Ponto de parada 3 at 0x56556584
(gdb) break *0x565565a4
Ponto de parada 4 at 0x565565a4
(gdb) break *0x565565c4
Ponto de parada 5 at 0x565565c4
(gdb) run "mab - 353 - 2020.2" "Lab 2: Bomb" ""
The program being debugged has been started already.
Start it from the beginning? (y or n) y
Starting program: /home/luiz/Área de Trabalho/labs comp prog/lab2/02-bomb/bomb "mab - 353 - 2020.2" "Lab 2: Bomb" ""
Primeira bomba defusada
Segunda bomba defusada

Breakpoint 2, 0x56556564 in terceiraBomba ()
(gdb) x /s ($ebp-0xc)
0xffffd08c: ""
(gdb) c
Continuing.

Breakpoint 3, 0x56556584 in terceiraBomba ()
(gdb) x /s ($eax)
0x56557106: "Programação"
(gdb)
```

Na terceira parada, temos o seguinte conteúdo: “Computadores\t\$!”

```
luiz@Jarvis: ~/Área de Trabalho/labs comp prog/lab2/02-bomb

0x565565e2 <+162>: jmp 0x565565e9 <terceiraBomba+169>
0x565565e4 <+164>: call 0x5655622d <boom>
0x565565e9 <+169>: nop
0x565565ea <+170>: mov -0x4(%ebp),%ebx
0x565565ed <+173>: leave
0x565565ee <+174>: ret
End of assembler dump.
(gdb) break *0x56556564
Ponto de parada 2 at 0x56556564
(gdb) break *0x56556584
Ponto de parada 3 at 0x56556584
(gdb) break *0x565565a4
Ponto de parada 4 at 0x565565a4
(gdb) break *0x565565c4
Ponto de parada 5 at 0x565565c4
(gdb) run "mab - 353 - 2020.2" "Lab 2: Bomb" ""
The program being debugged has been started already.
Start it from the beginning? (y or n) y
Starting program: /home/luiz/Área de Trabalho/labs comp prog/lab2/02-bomb/bomb "mab - 353 - 2020.2" "Lab 2: Bomb" ""
Primeira bomba defusada
Segunda bomba defusada

Breakpoint 2, 0x56556564 in terceiraBomba ()
(gdb) x /s ($ebp-0xc)
0xffffd08c: ""
(gdb) c
Continuing.

Breakpoint 3, 0x56556584 in terceiraBomba ()
(gdb) x /s ($eax)
0x56557106: "Programação"
(gdb) c
Continuing.

Breakpoint 4, 0x565565a4 in terceiraBomba ()
(gdb) x /s ($eax)
0x56557114: "Computadores\t$!"
(gdb) █
```

Na quarta parada foi printado: “**Programação Computadores\t\$!**”

```
luiz@Jarvis: ~/Área de Trabalho/labs comp prog/lab2/02-bomb

Ponto de parada 2 at 0x56556564
(gdb) break *0x56556584
Ponto de parada 3 at 0x56556584
(gdb) break *0x565565a4
Ponto de parada 4 at 0x565565a4
(gdb) break *0x565565c4
Ponto de parada 5 at 0x565565c4
(gdb) run "mab - 353 - 2020.2" "Lab 2: Bomb" ""
The program being debugged has been started already.
Start it from the beginning? (y or n) y
Starting program: /home/luiz/Área de Trabalho/labs comp prog/lab2/02-bomb/bomb "mab - 353 - 2020.2" "Lab 2: Bomb" ""
Primeira bomba defusada
Segunda bomba defusada

Breakpoint 2, 0x56556564 in terceiraBomba ()
(gdb) x /s ($ebp-0xc)
0xffffd08c: ""
(gdb) c
Continuing.

Breakpoint 3, 0x56556584 in terceiraBomba ()
(gdb) x /s ($eax)
0x56557106: "Programação"
(gdb) c
Continuing.

Breakpoint 4, 0x565565a4 in terceiraBomba ()
(gdb) x /s ($eax)
0x56557114: "Computadores\t$!"
(gdb) x /s ($ebx-0x1eec)
0x56557114: "Computadores\t$!"
(gdb) c
Continuing.

Breakpoint 5, 0x565565c4 in terceiraBomba ()
(gdb) x /s ($eax)
0x56557124: "Programação Computadores\t$!"
(gdb) █
```

No nosso último breakpoint temos o seguinte conteúdo sendo impresso “**Programação Computadores\t\$**”. Logo, temos que a nossa senha será composta pela junção dos conteúdos printados na nossa segunda e terceira parada, ou seja, temos que as palavras Programação e Computadores\t\$ de forma separada são senhas falsas, e para que a bomba não detone, é preciso que sejam passadas juntas. . Agora vamos testar se de fato essa é a nossa senha.

```
luiz@Jarvis: ~/Área de Trabalho/labs comp prog/lab2/02-bomb
luiz@Jarvis:~/Área de Trabalho/labs comp prog/lab2/02-bomb$ ./bomb "Mab - 353 - 2020.2" "Lab 2: Bomb" "Programação Computadores\t$"
Primeira bomba defusada
Segunda bomba defusada
Explodiu!
luiz@Jarvis:~/Área de Trabalho/labs comp prog/lab2/02-bomb$
```

Podemos observar que quando passamos a senha “**Programação Computadores\t\$**”, a terceira bomba explode. Isso acontece, pois essa senha é composta pelo caractere “\t” que representa o Tab e não levamos isso em consideração na hora de escrevermos a nossa senha. Logo, o programa não irá considerar essa senha como chave para defusar a bomba.

```
luiz@Jarvis: ~/Área de Trabalho/labs comp prog/lab2/02-bomb
luiz@Jarvis:~/Área de Trabalho/labs comp prog/lab2/02-bomb$ ./bomb "Mab - 353 - 2020.2" "Lab 2: Bomb" "Programação Computadores \$!"
Primeira bomba defusada
Segunda bomba defusada
Terceira bomba defusada
Explodiu!
luiz@Jarvis:~/Área de Trabalho/labs comp prog/lab2/02-bomb$
```

Escrevemos novamente a senha, porém com o caractere Tab. No terminal, podemos escrever o tab como “ctrl + v + tab”. Podemos observar que essa senha foi aceita e conseguimos defusar a terceira bomba.

Agora que conseguimos encontrar a terceira senha, vamos voltar para a função main e analisar o que está acontecendo:

```
add $0x10,%esp
Vamos adicionar 0x10 ao registrador %esp, ou seja, estamos desalocando
4 espaços de 4 bytes da pilha.
cmpl $0x4, (%ebx)
Iremos realizar uma comparação entre 0x4 e %ebx, ou seja, estamos
verificando se argc <= 4.
jle 0x1785 <main+176>
Se a condição anterior for atendida, ou seja, se não possuímos as
senhas para desarmar todas as bombas, iremos chamar a função "main".
mov 0x4(%ebx), %eax
O registrador %eax vai receber %ebx + 4.
```

```

add $0x10,%eax
Iremos somar 0x10 ao registrador %eax.
mov (%eax),%eax
Aqui o registrador %eax irá receber o valor do ponteiro argv[4].
sub $0xc,%esp
Iremos abrir mais 3 espaços de 4 bytes na pilha.
push %eax
Coloca no topo da pilha argv[4] como parâmetro da chamada da próxima
função.
call 0x15ef <quartaBomba>
Iremos chamar a função "quartaBomba".

```

Nesse momento, a pilha se encontra da seguinte forma:

Endereço	Conteúdo da Pilha <4 bytes>	Comentários
%ebp		Base da pilha
...	...	
%ecx+4	argv	Segundo parâmetro da main
%ecx	argc	Primeiro parâmetro da main
%ecx - 4	RIP SO	end de retorno a SO
	...	espaço de alinhamento
%esp+52 = %ebp+8 (nx16)		Alinhamento com múltiplo de 16
%esp+48 = %ebp+4	RIP SO	end de retorno SO
%esp+44 = %ebp	OFP	Base da main
%esp + 40 = %ebp - 4	%esi	
%esp + 36 = %ebp - 8(nx16)	%ebx	
%esp + 32 = %ebp - 12	%ecx	
%esp + 28 = %ebp - 16		
%esp+24 = %ebp - 20		
%esp +20 = %ebp - 24(nx16)		
%esp + 16 = %ebp - 28		
%esp + 12 = %ebp - 32		
%esp +8 = %ebp - 36		
%esp +4 = %ebp - 40(nx16)	%eax = argv[4]	Primeiro parâmetro da "quartaBomba"
%esp = %ebp - 44	RIP Main	End de retorno à Main

QUARTA BOMBA

Analisando o código da função quartaBomba, teremos:

```
0x15ef <quartaBomba>          push    %ebp
0x15f0 <quartaBomba+1>        mov     %esp,%ebp
0x15f2 <quartaBomba+3>        push    %ebx
0x15f3 <quartaBomba+4>        sub     $0x824,%esp
0x15f9 <quartaBomba+10>       call    0x1130 <__x86.get_pc_thunk.bx>
0x15fe <quartaBomba+15>       add     $0x2a02,%ebx
0x1604 <quartaBomba+21>       mov     0x8(%ebp),%eax
0x1607 <quartaBomba+24>       mov     %eax,-0x81c(%ebp)
0x160d <quartaBomba+30>       mov     %gs:0x14,%eax
0x1613 <quartaBomba+36>       mov     %eax,-0xc(%ebp)
0x1616 <quartaBomba+39>       xor     %eax,%eax
0x1618 <quartaBomba+41>       sub     $0x8,%esp
0x161b <quartaBomba+44>       lea     -0x1e9b(%ebx),%eax
0x1621 <quartaBomba+50>       push    %eax
0x1622 <quartaBomba+51>       lea     -0x1fe3(%ebx),%eax
0x1628 <quartaBomba+57>       push    %eax

0x1629 <quartaBomba+58>       call    0x10e0 <fopen@plt>
0x162e <quartaBomba+63>       add     $0x10,%esp
0x1631 <quartaBomba+66>       mov     %eax,-0x810(%ebp)
0x1637 <quartaBomba+72>       cmpl    $0x0,-0x810(%ebp)
0x163e <quartaBomba+79>       je      0x167a <quartaBomba+139>
0x1640 <quartaBomba+81>       sub     $0x4,%esp
0x1643 <quartaBomba+84>       pushl   -0x810(%ebp)
0x1649 <quartaBomba+90>       push    $0x800
0x164e <quartaBomba+95>       lea     -0x80c(%ebp),%eax
0x1654 <quartaBomba+101>      push    %eax
0x1655 <quartaBomba+102>      call    0x1050 <fgets@plt>
0x165a <quartaBomba+107>      add     $0x10,%esp
0x165d <quartaBomba+110>      sub     $0xc,%esp
0x1660 <quartaBomba+113>      lea     -0x80c(%ebp),%eax
0x1666 <quartaBomba+119>      push    %eax
0x1667 <quartaBomba+120>      call    0x10c0 <strlen@plt>

0x166c <quartaBomba+125>      add     $0x10,%esp
0x166f <quartaBomba+128>      sub     $0x1,%eax
0x1672 <quartaBomba+131>      movb    $0x0,-0x80c(%ebp,%eax,1)
0x167a <quartaBomba+139>      sub     $0xc,%esp
0x167d <quartaBomba+142>      pushl   -0x810(%ebp)
0x1683 <quartaBomba+148>      call    0x1060 <fclose@plt>
0x1688 <quartaBomba+153>      add     $0x10,%esp
```

```

0x168b <quartaBomba+156>    sub $0x8,%esp
0x168e <quartaBomba+159>    pushl  -0x81c(%ebp)
0x1694 <quartaBomba+165>    lea  -0x80c(%ebp),%eax
0x169a <quartaBomba+171>    push  %eax
0x169b <quartaBomba+172>    call  0x1040 <strcmp@plt>
0x16a0 <quartaBomba+177>    add $0x10,%esp
0x16a3 <quartaBomba+180>    test  %eax,%eax
0x16a5 <quartaBomba+182>    je   0x16ac <quartaBomba+189>
0x16a7 <quartaBomba+184>    call  0x122d <boom>
0x16ac <quartaBomba+189>    sub $0xc,%esp
0x16af <quartaBomba+192>    lea  -0x1e98(%ebx),%eax
0x16b5 <quartaBomba+198>    push  %eax
0x16b6 <quartaBomba+199>    call  0x10a0 <puts@plt>
0x16bb <quartaBomba+204>    add $0x10,%esp
0x16be <quartaBomba+207>    nop
0x16bf <quartaBomba+208>    mov  -0xc(%ebp),%eax
0x16c2 <quartaBomba+211>    sub  %gs:0x14,%eax
0x16c9 <quartaBomba+218>    je   0x16d0 <quartaBomba+225>
0x16cb <quartaBomba+220>    call  0x1820 <__stack_chk_fail_local>
0x16d0 <quartaBomba+225>    mov  -0x4(%ebp),%ebx
0x16d3 <quartaBomba+228>    leave
0x16d4 <quartaBomba+229>    ret

```

Primeiramente, colocamos break points nas seguintes linhas, pois imaginamos que a senha poderia estar dentro de um arquivo, visto que dentro da função “quartaBomba” possuímos as funções “fopen”, “fgets” e “fclose” que operam com arquivos.

```

call  0x565560e0 <fopen@plt>
je     0x5655667a <quartaBomba+139>
call  0x56556050 <fgets@plt>
call  0x565560c0 <strlen@plt>
call  0x56556060 <fclose@plt>
call  0x56556040 <strcmp@plt>

```

Agora iremos analisar o conteúdo que está sendo passado antes dos breakpoints mencionados para termos uma noção do que está sendo manipulado.

Na nossa primeira parada temos o seguinte conteúdo sendo impresso:

```
luiz@Jarvis: ~/Área de Trabalho/labs comp prog/lab2/02-bomb
0x565566d3 <+228>: leave
0x565566d4 <+229>: ret
End of assembler dump.
(gdb) break *0x56556629
Ponto de parada 2 at 0x56556629
(gdb) break *0x5655663e
Ponto de parada 3 at 0x5655663e
(gdb) break *0x56556655
Ponto de parada 4 at 0x56556655
(gdb) break *0x56556667
Ponto de parada 5 at 0x56556667
(gdb) break *0x56556683
Ponto de parada 6 at 0x56556683
(gdb) break *0x5655669b
Ponto de parada 7 at 0x5655669b
(gdb) break *0x565566b6
Ponto de parada 8 at 0x565566b6
(gdb) run "Mab - 353 - 2020.2" "Lab 2: Bomb" "Programação Computadores \${}" ""
The program being debugged has been started already.
Start it from the beginning? (y or n) y
Starting program: /home/luiz/Área de Trabalho/labs comp prog/lab2/02-bomb/bomb "Mab - 353 - 2020.2" "Lab 2: Bomb" "Programação Computador
es\${}" ""
Primeira bomba defusada
segunda bomba defusada
terceira bomba defusada

Breakpoint 2, 0x56556629 in quartaBomba ()
(gdb) x /s ($eax)
0x5655701d: "/tmp/.lv.1"
(gdb)
```

Na nossa segunda parada, temos:

```
luiz@Jarvis: ~/Área de Trabalho/labs comp prog/lab2/02-bomb
Ponto de parada 3 at 0x5655663e
(gdb) break *0x56556655
Ponto de parada 4 at 0x56556655
(gdb) break *0x56556667
Ponto de parada 5 at 0x56556667
(gdb) break *0x56556683
Ponto de parada 6 at 0x56556683
(gdb) break *0x5655669b
Ponto de parada 7 at 0x5655669b
(gdb) break *0x565566b6
Ponto de parada 8 at 0x565566b6
(gdb) run "Mab - 353 - 2020.2" "Lab 2: Bomb" "Programação Computadores \${}" ""
The program being debugged has been started already.
Start it from the beginning? (y or n) y
Starting program: /home/luiz/Área de Trabalho/labs comp prog/lab2/02-bomb/bomb "Mab - 353 - 2020.2" "Lab 2: Bomb" "Programação Computador
es\${}" ""
Primeira bomba defusada
segunda bomba defusada
terceira bomba defusada

Breakpoint 2, 0x56556629 in quartaBomba ()
(gdb) x /s ($eax)
0x5655701d: "/tmp/.lv.1"
(gdb) c
Continuing.

Breakpoint 3, 0x5655663e in quartaBomba ()
(gdb) x /s ($ebp-0x810)
0xfffffc868: "\240\241UV\030\377\375\367\220\331\377\367\352SUV"
(gdb)
```

Na terceira parada, temos:

```
luiz@Jarvis: ~/Área de Trabalho/labs comp prog/lab2/02-bomb
Ponto de parada 6 at 0x56556683
(gdb) break *0x5655669b
Ponto de parada 7 at 0x5655669b
(gdb) break *0x565566b6
Ponto de parada 8 at 0x565566b6
(gdb) run "Mab - 353 - 2020.2" "Lab 2: Bomb" "Programação Computadores \${!}" ""
The program being debugged has been started already.
Start it from the beginning? (y or n) y
Starting program: /home/luiz/Área de Trabalho/labs comp prog/lab2/02-bomb/bomb "Mab - 353 - 2020.2" "Lab 2: Bomb" "Programação Computador
es\${!}" ""
Primeira bomba defusada
Segunda bomba defusada
Terceira bomba defusada

Breakpoint 2, 0x56556629 in quartaBomba ()
(gdb) x /s ($eax)
0x5655701d: "/tmp/.lv.1"
(gdb) c
Continuing.

Breakpoint 3, 0x5655663e in quartaBomba ()
(gdb) x /s ($ebp-0x810)
0xfffffc868: "\240\241UV\030\377\375\367\220\331\377\367\352SUV"
(gdb) c
Continuing.

Breakpoint 4, 0x56556655 in quartaBomba ()
(gdb) x /s ($eax)
0xfffffc86c: "\030\377\375\367\220\331\377\367\352SUV"
(gdb)
```

Na quarta parada, temos:

```
luiz@Jarvis: ~/Área de Trabalho/labs comp prog/lab2/02-bomb
The program being debugged has been started already.
Start it from the beginning? (y or n) y
Starting program: /home/luiz/Área de Trabalho/labs comp prog/lab2/02-bomb/bomb "Mab - 353 - 2020.2" "Lab 2: Bomb" "Programação Computador
es\${!}" ""
Primeira bomba defusada
Segunda bomba defusada
Terceira bomba defusada

Breakpoint 2, 0x56556629 in quartaBomba ()
(gdb) x /s ($eax)
0x5655701d: "/tmp/.lv.1"
(gdb) c
Continuing.

Breakpoint 3, 0x5655663e in quartaBomba ()
(gdb) x /s ($ebp-0x810)
0xfffffc868: "\240\241UV\030\377\375\367\220\331\377\367\352SUV"
(gdb) c
Continuing.

Breakpoint 4, 0x56556655 in quartaBomba ()
(gdb) x /s ($eax)
0xfffffc86c: "\030\377\375\367\220\331\377\367\352SUV"
(gdb) c
Continuing.

Breakpoint 5, 0x56556667 in quartaBomba ()
(gdb) x /s ($eax)
0xfffffc86c: "eW91dHUuYmUvd1l4eHY1bjJFdmsK\n"
(gdb)
```

Na quinta parada, temos:

```
Terceira bomba defusada

Breakpoint 2, 0x56556629 in quartaBomba ()
(gdb) x /s ($eax)
0x5655701d: "/tmp/.lv.1"
(gdb) c
Continuing.

Breakpoint 3, 0x5655663e in quartaBomba ()
(gdb) x /s ($ebp-0x810)
0xfffffc868: "\240\241UV\030\377\375\367\220\331\377\367\352SUV"
(gdb) c
Continuing.

Breakpoint 4, 0x56556655 in quartaBomba ()
(gdb) x /s ($eax)
0xfffffc86c: "\030\377\375\367\220\331\377\367\352SUV"
(gdb) c
Continuing.

Breakpoint 5, 0x56556667 in quartaBomba ()
(gdb) x /s ($eax)
0xfffffc86c: "eW91dHUuYmUvd1l4eHY1bjJFdmsK\n"
(gdb) c
Continuing.

Breakpoint 6, 0x56556683 in quartaBomba ()
(gdb) x /s ($ebp-0x810)
0xfffffc868: "\240\241UVeW91dHUuYmUvd1l4eHY1bjJFdmsK"
(gdb) █
```

Na sexta parada, temos:

```
Continuing.

Breakpoint 3, 0x5655663e in quartaBomba ()
(gdb) x /s ($ebp-0x810)
0xfffffc868: "\240\241UV\030\377\375\367\220\331\377\367\352SUV"
(gdb) c
Continuing.

Breakpoint 4, 0x56556655 in quartaBomba ()
(gdb) x /s ($eax)
0xfffffc86c: "\030\377\375\367\220\331\377\367\352SUV"
(gdb) c
Continuing.

Breakpoint 5, 0x56556667 in quartaBomba ()
(gdb) x /s ($eax)
0xfffffc86c: "eW91dHUuYmUvd1l4eHY1bjJFdmsK\n"
(gdb) c
Continuing.

Breakpoint 6, 0x56556683 in quartaBomba ()
(gdb) x /s ($ebp-0x810)
0xfffffc868: "\240\241UVeW91dHUuYmUvd1l4eHY1bjJFdmsK"
(gdb) c
Continuing.

Breakpoint 7, 0x5655669b in quartaBomba ()
(gdb) x /s ($eax)
0xfffffc86c: "eW91dHUuYmUvd1l4eHY1bjJFdmsK"
(gdb) █
```

Depois de todas as alterações que o conteúdo do arquivo passou, chegamos em um resultado final.

Podemos observar que **"eW91dHUuYmUvd1l4eHY1bjJFdmsK"** possivelmente será nossa quarta senha, visto que nesse breakpoint teremos a última comparação sendo realizada na rotina com o risco de desviar para a função boom. Agora falta testar para ver se isso realmente é verdade.

```
lulz@Jarvis: ~/Área de Trabalho/labs comp prog/lab2/02-bomb
lulz@Jarvis:~/Área de Trabalho/labs comp prog/lab2/02-bomb$ ./bomb "Mab - 353 - 2020.2" "Lab 2: Bomb" "Programação Computadores \!" "eW91dHUuYmUvd1l4eHY1bjJFdmsK"
Primeira bomba defusada
Segunda bomba defusada
Terceira bomba defusada
Quarta bomba defusada
Todas as bombas defusadas. Parabéns!
lulz@Jarvis:~/Área de Trabalho/labs comp prog/lab2/02-bomb$
```

Aqui podemos ver que de fato **"eW91dHUuYmUvd1l4eHY1bjJFdmsK"** era nossa senha para desarmar a quarta bomba e última bomba, que juntamente com a mensagem de que a quarta bomba foi defusada, também é retornada uma mensagem de "Todas as bombas defusadas. Parabéns", que é passada pela main no seguinte e último código:

```
add $0x10,%esp
sub $0xc,%esp
lea -0x1e78(%esi),%eax
push %eax
mov %esi,%ebx
call 0x10a0 <puts@plt>
add $0x10,%esp
mov $0x0,%eax
jmp 0x178f <main+186>
call 0x122d <boom>
mov $0x0,%eax
lea -0xc(%ebp),%esp
pop %ecx
pop %ebx
pop %esi
pop %ebp
lea -0x4(%ecx),%esp
ret
```

Colocando um breakpoint na função "puts" para ter certeza do que está sendo passado como parâmetro, conseguimos comprovar nossas suspeitas:

```
lulz@Jarvis: ~/Área de Trabalho/labs comp prog/lab2/02-bomb
0x56556756 <+129>: mov    0x4(%ebx),%eax
0x56556759 <+132>: add    $0x10,%eax
0x5655675c <+135>: mov    (%eax),%eax
0x5655675e <+137>: sub    $0xc,%esp
0x56556761 <+140>: push   %eax
0x56556762 <+141>: call   0x565567ef <quartaBomba>
0x56556767 <+146>: add    $0x10,%esp
0x5655676a <+149>: sub    $0xc,%esp
--Type <RET> for more, q to quit, c to continue without paging--
0x5655676d <+152>: lea    -0x1e78(%esi),%eax
0x56556773 <+158>: push   %eax
0x56556774 <+159>: mov    %esi,%ebx
=> 0x56556776 <+161>: call   0x565567a0 <puts@plt>
0x5655677b <+166>: add    $0x10,%esp
0x5655677e <+169>: mov    $0x0,%eax
0x56556783 <+174>: jmp     0x5655678f <main+186>
0x56556785 <+176>: call   0x5655672d <boom>
0x5655678a <+181>: mov    $0x0,%eax
0x5655678f <+186>: lea    -0xc(%ebp),%esp
0x56556792 <+189>: pop     %ecx
0x56556793 <+190>: pop     %ebx
0x56556794 <+191>: pop     %esi
0x56556795 <+192>: pop     %ebp
0x56556796 <+193>: lea    -0x4(%ecx),%esp
0x56556799 <+196>: ret
End of assembler dump.
(gdb) x /s ($eax)
0x56557188:    "\033[34mTodas as bombas defusadas. Parabéns!\033[0m"
(gdb)
```