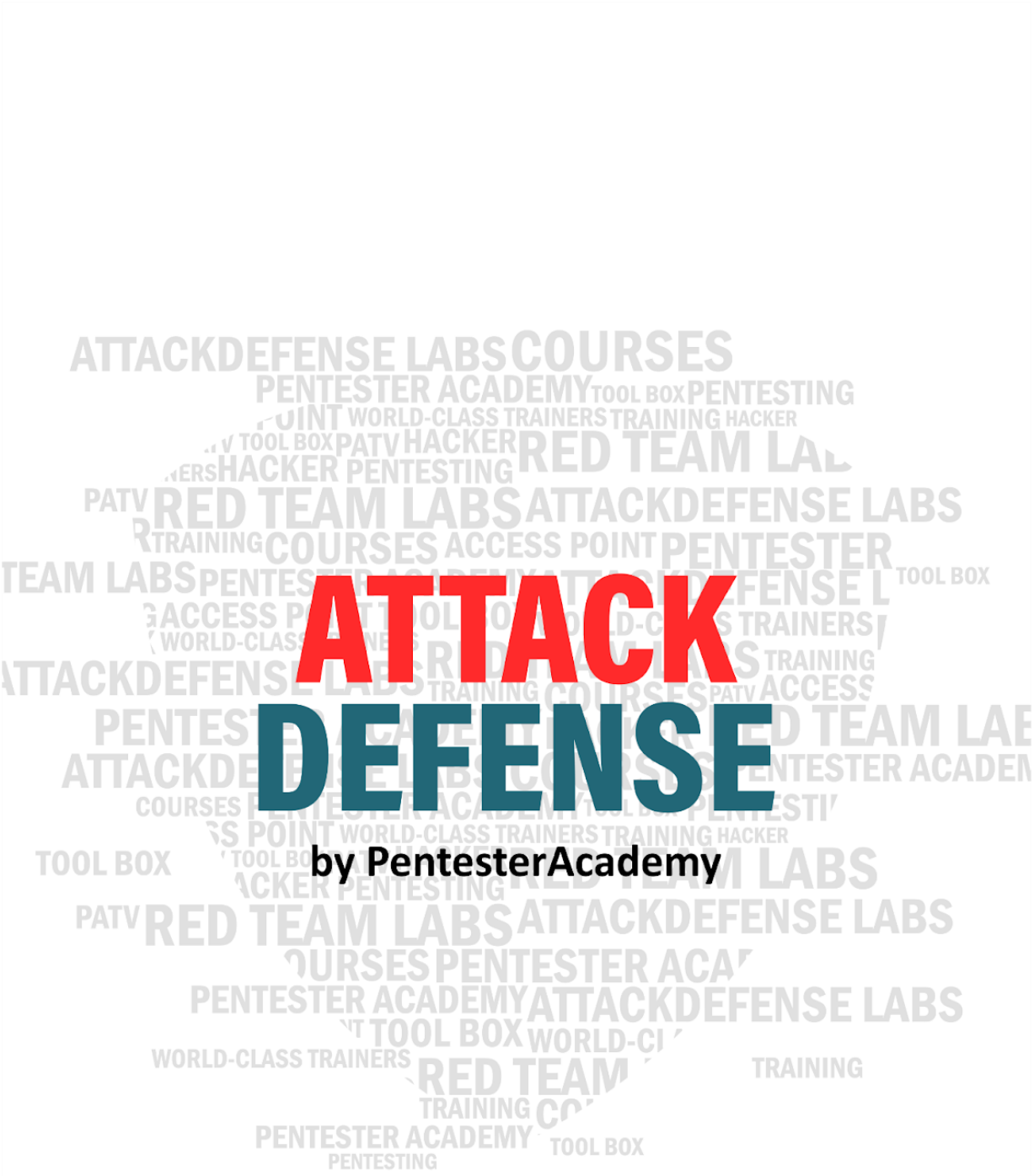
ATTACK
DEFENSE
by PentesterAcademy

Name	Examining Data
URL	https://www.attackdefense.com/challengedetails?cid=2170
Type	Reverse Engineering : GDB Basics

Important Note: This document illustrates all the important steps required to complete this lab. This is by no means a comprehensive step-by-step solution for this exercise. This is only provided as a reference to various commands needed to complete this exercise and for your further research on this topic. Also, note that the IP addresses and domain names might be different in your lab.

Objective: Learn how to examine the data files in GDB and check out different commands/options/methods.

Solution:

Step 1: Open sample3 binary using gdb.

Command: gdb -q sample3

```
root@localhost:~# gdb -q sample3
Reading symbols from sample3...
(gdb)
```

Step 2: List the source code.

Command: list

```
(gdb) list
17      struct SimpleStruct ss = { 10, 1.11 };
18      struct ComplexStruct cs = { &ss, { 0, 1, 2, 3, 4, 5, 6, 7, 8, 9 } };
19
20      int sum_f(int x,int y){
21              return x+y;
22      }
23
24      int sum_func(int num1, int num2) {
25              int tsum=0;
26              tsum = sum_f(num1,num2);
```

```
(gdb)
27              return tsum;
28      }
29
30      int main(int argc, char * argv[]) {
31              int a=0, b=0, result=0;
32
33              if (argc != 3) {
34                      printf("WRONG Params!! \n\n ./sample1 <num1> <num2> \n");
35                      exit(1);
36              }
```

```
(gdb)
37
38                  a = atoi(argv[1]);
39                  b = atoi(argv[2]);
40
41                  printf("Both numbers accepted for addition. \n");
42
43                  result = sum_func(a,b);
44
45                  printf("Sum is : %d \n", result);
46
```

Step 3: Set a breakpoint at line 21

Command: b 21

```
(gdb) b 21
Breakpoint 1 at 0x744: file sample3.c, line 21.
(gdb) run 3 5
Starting program: /root/sample3 3 5
Both numbers accepted for addition.

Breakpoint 1, sum_f (x=3, y=5) at sample3.c:21
21              return x+y;
```

Printing variables

Step 4: Print value of variable x

Command: p x

```
(gdb) p x
$1 = 3
```

Step 5: Print value of variable x in different formats

Hexadecimal

Command: p/x x

```
(gdb) p/x x
$6 = 0x3
```

Signed Decimal

Command: p/d x

```
(gdb) p/d x
$7 = 3
```

Unsigned Decimal

Command: p/u x

```
(gdb) p/u x
$8 = 3
```

Octal

Command: p/o x

```
(gdb) p/o x
$9 = 03
```

Integer in binary

Command: p/t x

```
(gdb) p/t x
$10 = 11
```

Prints as an address

Command: p/a x

```
(gdb) p/a x
$11 = 0x3
```

Regard it as integer and prints as character constant

Command: p/c x

```
(gdb) p/c x
$12 = 3 '\003'
```

Floating point

Command: p/f x

```
(gdb) p/f x
$13 = 4.20389539e-45
```

String

Command: p/s x

```
(gdb) p/s x
$14 = 3
```

Treated as Integer, printed as hexadecimal but padded with 0s to the size of integer

Command: p/z x

```
(gdb) p/z x
$15 = 0x00000003
```

Raw printing (disabling any pretty print)

Command: p/r x

```
(gdb) p/r x
$16 = 0x00000003
```

Step 6: Print value of variable a

Command: p a

```
(gdb) p a
No symbol "a" in current context.
```

The control is not in the main() function, hence the error. In this case, one can print the value by specifying the function name

Command: p main::a

```
(gdb) p main::a
$2 = 3
```

Step 7: Explore the definition of structure ComplexStruct

Command: explore struct ComplexStruct

The command checks the structure and provide option to explore the members of the structure using numeric menu

```
(gdb) explore struct ComplexStruct
'struct ComplexStruct' is a struct/class with the following fields:

  ss_p = <Enter 0 to explore this field of type 'struct SimpleStruct *'>
   arr = <Enter 1 to explore this field of type 'int [10]'>

Enter the field number of choice: 1
field 'arr' of 'struct ComplexStruct' is an array of 'int'.
the array element of field 'arr' of 'struct ComplexStruct' is of a scalar type 'int'.

Press enter to return to enclosing type:

Returning to enclosing type...

'struct ComplexStruct' is a struct/class with the following fields:

  ss_p = <Enter 0 to explore this field of type 'struct SimpleStruct *'>
   arr = <Enter 1 to explore this field of type 'int [10]'>

Enter the field number of choice:
(gdb)
```

Step 8: Explore the structure ss

Command: explore ss

```
(gdb) explore ss
The value of 'ss' is a struct/class of type 'struct SimpleStruct' with the following fields:

  i = 10 .. (Value of type 'int')
  d = 1.1100000000000001 .. (Value of type 'double')
```

Step 9: Explore the structure ss

Command: explore cs

The values of structure members can be explored and read.

```
(gdb) explore cs
The value of 'cs' is a struct/class of type 'struct ComplexStruct' with the following fields:

  ss_p = <Enter 0 to explore this field of type 'struct SimpleStruct *'>
   arr = <Enter 1 to explore this field of type 'int [10]'>

Enter the field number of choice: 0
'cs.ss_p' is a pointer to a value of type 'struct SimpleStruct'
Continue exploring it as a pointer to a single value [y/n]: y

  The value of '*(cs.ss_p)' is a struct/class of type 'struct SimpleStruct' with the following fields:

  i = 10 .. (Value of type 'int')
  d = 1.1100000000000001 .. (Value of type 'double')
```

```
Press enter to return to parent value:
The value of 'cs' is a struct/class of type 'struct ComplexStruct' with the following fields:

  ss_p = <Enter 0 to explore this field of type 'struct SimpleStruct *'>
   arr = <Enter 1 to explore this field of type 'int [10]'>

Enter the field number of choice: 1
'cs.arr' is an array of 'int'.
Enter the index of the element you want to explore in 'cs.arr': 0
'(cs.arr)[0]' is a scalar value of type 'int'.
(cs.arr)[0] = 0

Press enter to return to parent value:

Returning to parent value...

'cs.arr' is an array of 'int'.
Enter the index of the element you want to explore in 'cs.arr':

Returning to parent value...

--Type <RET> for more, q to quit, c to continue without paging--q
Quit
```

Step 10: Print the value of member i from structure ss

Command: print ss.i

```
(gdb) print ss.i
$1 = 10
(gdb)
```

Step 11: Print the value of member pointer ss_p from structure cs

Command: print *cs.ss_p

```
(gdb) print  *cs.ss_p
$2 = {i = 10, d = 1.1100000000000001}
```

Step 12: Print the value of member pointer ss_p from structure cs and apply pretty output.

Command: print -pretty -- *cs.ss_p

```
(gdb) print -pretty --  *cs.ss_p
$3 = {
  i = 10,
  d = 1.1100000000000001
}
```

Step 13: Print the value of member array arr from structure cs.

Command: p cs.arr

```
(gdb) p cs.arr
$3 = {0, 1, 2, 3, 4, 5, 6, 7, 8, 9}
```

Step 14: Print the value stored at index 2 of member array arr from structure cs.

Command: p cs.arr[2]

```
(gdb) p cs.arr[2]
$4 = 2
```

Step 15: Another way to print the first 5 members of array

Command: p *cs.arr@5

```
(gdb) p *cs.arr@5
$19 = {0, 1, 2, 3, 4}
```

Step 16: Print the value of member i pointed by pointer member ss_p from structure cs.

Command: p cs.ss_p.i

```
(gdb) p cs.ss_p.i
$5 = 10
```

Examining Memory

A memory location can be examined using command x/nfu addr

Where, n is repeat numeric count, f is format (x,c,d,c,s etc) and u is unit size

Supported Unit sizes:

- Byte b
- Half word (2 bytes) h
- Word (4 bytes) w
- Giant Word (8 bytes) g

Step 17: Print contents of memory a memory location

Print in 1 byte in hexadecimal format

Command: x/1xb 0x7fffffffe4a8

```
(gdb) x/1xb 0x7fffffffe4a8
0x7fffffffe4a8: 0xd0
```

Print in 5 bytes in hexadecimal format

Command: x/5xb 0x7fffffffe4a8

```
(gdb) x/5xb 0x7fffffffe4a8
0x7fffffffe4a8: 0xd0    0xe4    0xff    0xff    0xff
```

Print in 5 half words in hexadecimal format

Command: x/15xh 0x7fffffffe4a8

```
(gdb) x/5xh 0x7fffffffe4a8
0x7fffffffe4a8: 0xe4d0  0xffff  0x7fff  0x0000  0x4772
```

Print in 5 words in hexadecimal format

Command: x/5xw 0x7fffffffe4a8

```
(gdb) x/5xw 0x7fffffffe4a8
0x7fffffffe4a8: 0xffffe4d0      0x00007fff      0x55554772      0x00005555
0x7fffffffe4b8: 0x00000005
```

Print in 5 giant words in hexadecimal format

Command: x/5xg 0x7fffffffe4a8

```
(gdb) x/5xg 0x7fffffffe4a8
0x7fffffffe4a8: 0x00007fffffffe4d0      0x0000555555554772
0x7fffffffe4b8: 0x0000000300000005      0x0000555555554630
0x7fffffffe4c8: 0x00000000ffffe5f0
```

Print in 5 giant words in decimal format

Command: x/5dg 0x7fffffffe4a8

```
(gdb) x/5dg 0x7fffffffe4a8
0x7fffffffe4a8: 140737488348368 93824992233330
0x7fffffffe4b8: 12884901893     93824992233008
0x7fffffffe4c8: 4294960624
```

Print in 4 bytes in character format

Command: x/4cb 0x7fffffffe4a8

```
(gdb) x/4cb 0x7fffffffe4a8
0x7fffffffe4a8: -48 '\320'      -28 '\344'      -1 '\377'       -1 '\377'
```

Registers

Step 18: List main registers along with values

Command: info registers

```
(gdb) info registers
rax            0x3                 3
rbx            0x555555554830      93824992233520
rcx            0x7ffff7ee3057      140737352970327
rdx            0x5                 5
rsi            0x5                 5
rdi            0x3                 3
rbp            0x7fffffffe4a8      0x7fffffffe4a8
rsp            0x7fffffffe4a8      0x7fffffffe4a8
r8             0x25                37
r9             0x7c                124
r10            0x7ffff7fbdbe0      140737353866208
r11            0x246               582
r12            0x555555554630      93824992233008
r13            0x7fffffffe5f0      140737488348656
r14            0x0                 0
r15            0x0                 0
rip            0x555555554744      0x555555554744 <sum_f+10>
```

Step 19: List all registers along with values

Command: info all-registers

```
(gdb) info all-registers
rax            0x3                 3
rbx            0x555555554830      93824992233520
rcx            0x7ffff7ee3057      140737352970327
rdx            0x5                 5
rsi            0x5                 5
rdi            0x3                 3
rbp            0x7fffffffe4a8      0x7fffffffe4a8
rsp            0x7fffffffe4a8      0x7fffffffe4a8
r8             0x25                37
r9             0x7c                124
r10            0x7ffff7fbdbe0      140737353866208
r11            0x246               582
r12            0x555555554630      93824992233008
r13            0x7fffffffe5f0      140737488348656
r14            0x0                 0
```

```
ss             0x2b                43
ds             0x0                 0
es             0x0                 0
fs             0x0                 0
gs             0x0                 0
st0            0                   (raw 0x00000000000000000000)
st1            0                   (raw 0x00000000000000000000)
st2            0                   (raw 0x00000000000000000000)
st3            0                   (raw 0x00000000000000000000)
st4            0                   (raw 0x00000000000000000000)
--Type <RET> for more, q to quit, c to continue without paging--
st5            0                   (raw 0x00000000000000000000)
st6            0                   (raw 0x00000000000000000000)
st7            0                   (raw 0x00000000000000000000)
fctrl          0x37f               895
fstat          0x0                 0
ftag           0xffff              65535
fiseg          0x0                 0
fioff          0x0                 0
foseg          0x0                 0
```

```
xmm0           {v4_float = {0x2a000000, 0x0, 0x6a000000, 0x0}, v2_double = {0x0, 0x0}, v16_int8 = {0xa0, 0x62, 0x75, 0x55, 0x55, 0x55, 0x0, 0x0,
 0xa0, 0x66, 0x75, 0x55, 0x55, 0x55, 0x0, 0x0}, v8_int16 = {0x62a0, 0x5575, 0x5555, 0x0, 0x66a0, 0x5575, 0x5555, 0x0}, v4_int32 = {0x557562a0, 0
x5555, 0x557566a0, 0x5555}, v2_int64 = {0x5555557562a0, 0x5555557566a0}, uint128 = 0x5555557566a00000555557562a0}
xmm1           {v4_float = {0x2a000000, 0x0, 0x0, 0x0}, v2_double = {0x0, 0x0}, v16_int8 = {0xa0, 0x62, 0x75, 0x55, 0x55, 0x55, 0x0, 0x0, 0x0, 0
x0, 0x0, 0x0, 0x0, 0x0, 0x0, 0x0, 0x0}, v8_int16 = {0x62a0, 0x5575, 0x5555, 0x0, 0x0, 0x0, 0x0, 0x0}, v4_int32 = {0x557562a0, 0x5555, 0x0, 0x0}, v2_i
nt64 = {0x5555557562a0, 0x0}, uint128 = 0x5555557562a0}
xmm2           {v4_float = {0x6a000000, 0x0, 0x0, 0x0}, v2_double = {0x0, 0x0}, v16_int8 = {0xa0, 0x66, 0x75, 0x55, 0x55, 0x55, 0x0, 0x0, 0x0, 0
x0, 0x0, 0x0, 0x0, 0x0, 0x0, 0x0, 0x0}, v8_int16 = {0x66a0, 0x5575, 0x5555, 0x0, 0x0, 0x0, 0x0, 0x0}, v4_int32 = {0x557566a0, 0x5555, 0x0, 0x0}, v2_i
nt64 = {0x5555557566a0, 0x0}, uint128 = 0x5555557566a0}
xmm3           {v4_float = {0xffffffff, 0x0, 0xffffffff, 0xffffffff}, v2_double = {0x0, 0x7fffffffffffffff}, v16_int8 = {0x0, 0x0, 0xff, 0xff, 0
x0, 0x0, 0x0, 0x0, 0x0, 0xff, 0xff, 0xff, 0x0, 0xff, 0xff, 0xff}, v8_int16 = {0x0, 0xffff, 0x0, 0x0, 0xff00, 0xffff, 0xff00, 0xffff}, v4_int32 =
 {0xffff0000, 0x0, 0xffffff00, 0xffffff00}, v2_int64 = {0xffff0000, 0xffffff00ffffff00}, uint128 = 0xffffff00ffffff0000000000ffff0000}
xmm4           {v4_float = {0x0, 0x0, 0x0, 0x0}, v2_double = {0x0, 0x0}, v16_int8 = {0x0 <repeats 16 times>}, v8_int16 = {0x0, 0x0, 0x0, 0x0, 0x
0, 0x0, 0x0, 0x0}, v4_int32 = {0x0, 0x0, 0x0, 0x0}, v2_int64 = {0x0, 0x0}, uint128 = 0x0}
xmm5           {v4_float = {0x0, 0x0, 0x0, 0x0}, v2_double = {0x0, 0x0}, v16_int8 = {0x0 <repeats 16 times>}, v8_int16 = {0x0, 0x0, 0x0, 0x0, 0x
0, 0x0, 0x0, 0x0}, v4_int32 = {0x0, 0x0, 0x0, 0x0}, v2_int64 = {0x0, 0x0}, uint128 = 0x0}
xmm6           {v4_float = {0x0, 0x0, 0x0, 0x0}, v2_double = {0x0, 0x0}, v16_int8 = {0x20, 0x45, 0xfc, 0xf7, 0xff, 0x7f, 0x0, 0x0, 0x3, 0x0, 0x0
, 0x0, 0x0, 0x0, 0x0, 0x0}, v8_int16 = {0x4520, 0xf7fc, 0x7fff, 0x0, 0x3, 0x0, 0x0, 0x0}, v4_int32 = {0xf7fc4520, 0x7fff, 0x3, 0x0}, v2_int64 =
--Type <RET> for more, q to quit, c to continue without paging--
{0x7ffff7fc4520, 0x3}, uint128 = 0x300007ffff7fc4520}
```

Step 20: List registers of different groups along with value.s

Float group

Command: info registers float

```
(gdb) info registers float
st0            0                    (raw 0x00000000000000000000)
st1            0                    (raw 0x00000000000000000000)
st2            0                    (raw 0x00000000000000000000)
st3            0                    (raw 0x00000000000000000000)
st4            0                    (raw 0x00000000000000000000)
st5            0                    (raw 0x00000000000000000000)
st6            0                    (raw 0x00000000000000000000)
st7            0                    (raw 0x00000000000000000000)
fctrl          0x37f                895
fstat          0x0                  0
ftag           0xffff               65535
fiseg          0x0                  0
fioff          0x0                  0
foseg          0x0                  0
fooff          0x0                  0
fop            0x0                  0
```

System group

Command: info registers system

```
(gdb) info registers system
fs_base        0x7ffff7fc5540      140737353897280
gs_base        0x0                 0
orig_rax       0xffffffffffffffff  -1
```

Float group

Command: info registers general

```
(gdb) info registers general
rax            0x3                 3
rbx            0x555555554830      93824992233520
rcx            0x7ffff7ee3057      140737352970327
rdx            0x5                 5
rsi            0x5                 5
rdi            0x3                 3
rbp            0x7fffffffe4a8      0x7fffffffe4a8
rsp            0x7fffffffe4a8      0x7fffffffe4a8
r8             0x25                37
r9             0x7c                124
r10            0x7ffff7fbdbe0      140737353866208
r11            0x246               582
r12            0x555555554630      93824992233008
r13            0x7fffffffe5f0      140737488348656
r14            0x0                 0
r15            0x0                 0
rip            0x555555554744      0x555555554744 <sum_f+10>
eflags         0x216               [ PF AF IF ]
cs             0x33                51
ss             0x2b                43
ds             0x0                 0
es             0x0                 0
fs             0x0                 0
gs             0x0                 0
```

Step 21: Print values of Stack Pointer (SP) and Frame Pointer (RBP) register.

Commands:
info registers $sp
info registers $rbp

```
(gdb) info registers $sp
sp             0x7fffffffe4a8      0x7fffffffe4a8
(gdb) info registers $rbp
rbp            0x7fffffffe4a8      0x7fffffffe4a8
```

Step 22: Increment the value of Stack Pointer (SP) by 4.

Commands:
p/x $sp
set $sp += 4
p/x $sp

```
(gdb) p/x $sp
$20 = 0x7fffffffe4a8
```

```
(gdb) set $sp += 4
```

```
(gdb) p/x $sp
$27 = 0x7fffffffe4ac
```

Step 23: Print the value stored in Program Counter register

Command: p/x $pc

```
(gdb) p/x $pc
$21 = 0x555555554744
```

Step 24: Print the instruction which is to be executed next.

Command: x/i $pc

```
(gdb) x/i $pc
=> 0x555555554744 <sum_f+10>:    mov    edx,DWORD PTR [rbp-0x4]
```

Floating Point Hardware

Step 25: Check hardware-dependent information about the floating point unit.

Command: info float

```
(gdb) info float
  R7: Empty   0x00000000000000000000
  R6: Empty   0x00000000000000000000
  R5: Empty   0x00000000000000000000
  R4: Empty   0x00000000000000000000
  R3: Empty   0x00000000000000000000
  R2: Empty   0x00000000000000000000
  R1: Empty   0x00000000000000000000
=>R0: Empty   0x00000000000000000000

Status Word:         0x0000
                       TOP: 0
Control Word:        0x037f   IM DM ZM OM UM PM
                       PC: Extended Precision (64-bits)
                       RC: Round to nearest
Tag Word:            0xffff
Instruction Pointer: 0x00:0x00000000
Operand Pointer:     0x00:0x00000000
Opcode:              0x0000
(gdb)
```

Vector Unit

Step 26: Check vector unit information

Command: info vector

```
(gdb) info vector
xmm0           {v4_float = {0x2a000000, 0x0, 0x6a000000, 0x0}, v2_double = {0x0, 0x0}, v16_int8 = {0xa0, 0x62, 0x75, 0x55, 0x55, 0x55, 0x0, 0x0,
 0xa0, 0x66, 0x75, 0x55, 0x55, 0x55, 0x0, 0x0}, v8_int16 = {0x62a0, 0x5575, 0x5555, 0x0, 0x66a0, 0x5575, 0x5555, 0x0}, v4_int32 = {0x557562a0, 0
x5555, 0x557566a0, 0x5555}, v2_int64 = {0x5555557562a0, 0x5555557566a0}, uint128 = 0x5555557566a000005555557562a0}
xmm1           {v4_float = {0x2a000000, 0x0, 0x0, 0x0}, v2_double = {0x0, 0x0}, v16_int8 = {0xa0, 0x62, 0x75, 0x55, 0x55, 0x55, 0x0, 0x0, 0x0, 0
x0, 0x0, 0x0, 0x0, 0x0, 0x0, 0x0, 0x0}, v8_int16 = {0x62a0, 0x5575, 0x5555, 0x0, 0x0, 0x0, 0x0, 0x0}, v4_int32 = {0x557562a0, 0x5555, 0x0, 0x0}, v2_i
nt64 = {0x5555557562a0, 0x0}, uint128 = 0x5555557562a0}
xmm2           {v4_float = {0x6a000000, 0x0, 0x0, 0x0}, v2_double = {0x0, 0x0}, v16_int8 = {0xa0, 0x66, 0x75, 0x55, 0x55, 0x55, 0x0, 0x0, 0x0, 0
x0, 0x0, 0x0, 0x0, 0x0, 0x0, 0x0, 0x0}, v8_int16 = {0x66a0, 0x5575, 0x5555, 0x0, 0x0, 0x0, 0x0, 0x0}, v4_int32 = {0x557566a0, 0x5555, 0x0, 0x0}, v2_i
nt64 = {0x5555557566a0, 0x0}, uint128 = 0x5555557566a0}
xmm3           {v4_float = {0xffffffff, 0x0, 0xffffffff, 0xffffffff}, v2_double = {0x0, 0x7fffffffffffffff}, v16_int8 = {0x0, 0x0, 0xff, 0xff, 0
x0, 0x0, 0x0, 0x0, 0x0, 0x0, 0xff, 0xff, 0xff, 0x0, 0xff, 0xff, 0xff}, v8_int16 = {0x0, 0xffff, 0x0, 0x0, 0xff00, 0xffff, 0xff00, 0xffff}, v4_int32 =
 {0xffff0000, 0x0, 0xffffff00, 0xffffff00}, v2_int64 = {0xffff0000, 0xffffff00ffffff00}, uint128 = 0xffffff00ffffff00000000000ffff0000}
xmm4           {v4_float = {0x0, 0x0, 0x0, 0x0}, v2_double = {0x0, 0x0}, v16_int8 = {0x0 <repeats 16 times>}, v8_int16 = {0x0, 0x0, 0x0, 0x0, 0x
0, 0x0, 0x0, 0x0}, v4_int32 = {0x0, 0x0, 0x0, 0x0}, v2_int64 = {0x0, 0x0}, uint128 = 0x0}
```

Automatic Display

Automatic display is helpful if the user wants to track the change in the value of an argument or expression. Once that expression is defined as an automatic display, the value of this expression will be printed on each step.

Step 27: Kill the running instance of the program

Command: kill

```
(gdb) kill
Kill the program being debugged? (y or n) y
[Inferior 1 (process 6667) killed]
```

Step 28: Start the program again and pass arguments to it

Command: start 4 5

```
(gdb) start 4 5
Temporary breakpoint 2 at 0x555555554789: file sample3.c, line 31.
Starting program: /root/sample3 4 5

Temporary breakpoint 2, main (argc=3, argv=0x7fffffffe5f8) at sample3.c:31
31              int a=0, b=0, result=0;
```

Step 29: Define the display expression/equation

Command: display a+b

```
(gdb) display a+b
1: a+b = 32767
```

Step 30: Check the current values of defined display expressions

Command: display a+b

```
(gdb) display
1: a+b = 32767
```

Step 31: List all defined automatic display

Command: info display

```
(gdb) info display
Auto-display expressions now in effect:
Num Enb Expression
1:   y  a+b
```

Step 32: Step through the execution using “next” or “n” command and observe the value of defined display expression

Command: n

```
(gdb) n
33              if (argc != 3) {
1: a+b = 0
(gdb) n
38              a = atoi(argv[1]);
1: a+b = 0
(gdb) n
39              b = atoi(argv[2]);
1: a+b = 4
(gdb) n
41              printf("Both numbers accepted for addition. \n");
1: a+b = 9
(gdb)
```

Step 33: Once the work is done, the display can be disabled.

Command: disable display 1

```
(gdb) disable display 1
(gdb) n
Both numbers accepted for addition.
43              result = sum_func(a,b);
```

Step 34: Once the work is done, the display can be disabled.

Command: disable display 1

```
(gdb) disable display 1
(gdb) n
Both numbers accepted for addition.
43              result = sum_func(a,b);
```

Step 35: Similarly, the delete operation is also available. Delete the defined automatic display expression and verify the same.

Commands:
info display
delete display 1
info display

```
(gdb) info display
Auto-display expressions now in effect:
Num Enb Expression
1:   n  a+b
(gdb) delete display 1
(gdb) info display
There are no auto-display expressions now.
(gdb)
```

Operating System Auxiliary Information

GDB can access operating system-specific information and show on some OS.

Command syntax: info os <infotype>

Step 36: The list of the possible values for infotype can be checked by defining no infotype.

Command: info os

```
(gdb) info os
Type        Description
cpus        Listing of all cpus/cores on the system
files       Listing of all file descriptors
modules     Listing of all loaded kernel modules
msg         Listing of all message queues
processes   Listing of all processes
procgroups  Listing of all process groups
semaphores  Listing of all semaphores
shm         Listing of all shared-memory regions
sockets     Listing of all internet-domain sockets
threads     Listing of all threads
```

Step 37: Display the list of all CPUs/cores.

Command: info os cpus

```
(gdb) info os cpus
processor  vendor_id  cpu family model      model name stepping   cpu MHz    cache size physical id siblings   core id    cpu cores  apicid
initial apicid fpu        fpu_exception cpuid level wp         flags      bugs       bogomips   TLB size   clflush size cache_alignment address
sizes power management
0          AuthenticAMD 6          6          QEMU Virtual CPU version 2.5+ 3          1996.313   512 KB     0          1          0          1
         0          0          yes        yes        13         yes        fpu de pse tsc msr pae mce cx8 apic sep mtrr pge mca cmov pat pse36 c
lflush mmx fxsr sse sse2 syscall nx lm nopl cpuid pni cx16 hypervisor lahf_lm svm 3dnowprefetch vmmcall fxsave_leak sysret_ss_attrs spectre_v1 s
pectre_v2 spec_store_bypass 3992.62    1024 4K pages 64         64         40 bits physical, 48 bits virtual
1          AuthenticAMD 6          6          QEMU Virtual CPU version 2.5+ 3          1996.313   512 KB     1          1          0          1
         1          1          yes        yes        13         yes        fpu de pse tsc msr pae mce cx8 apic sep mtrr pge mca cmov pat pse36 c
lflush mmx fxsr sse sse2 syscall nx lm nopl cpuid pni cx16 hypervisor lahf_lm svm 3dnowprefetch vmmcall fxsave_leak sysret_ss_attrs spectre_v1 s
pectre_v2 spec_store_bypass 4003.76    1024 4K pages 64         64         40 bits physical, 48 bits virtual
```

Step 38: Display the list of open file descriptors on the target.

Command: info os files

```
(gdb) info os files
pid         command    file descriptor name
1           systemd    0           /dev/null
1           systemd    1           /dev/null
1           systemd    2           /dev/null
1           systemd    3           /dev/kmsg
1           systemd    4           anon_inode:[eventpoll]
1           systemd    5           anon_inode:[signalfd]
1           systemd    6           anon_inode:inotify
1           systemd    7           /sys/fs/cgroup/unified
1           systemd    8           anon_inode:[timerfd]
1           systemd    9           socket:[11274]
1           systemd    10          anon_inode:[eventpoll]
1           systemd    11          /proc/1/mountinfo
1           systemd    12          anon_inode:inotify
```

Step 39: Display the list of all loaded kernel modules on the target.

Command: info os modules

```
(gdb) info os modules
name        size       num uses   dependencies status     address
ppdev       24576      0          -            Live       ffffffffc02df000
kvm_amd     94208      0          -            Live       ffffffffc02c7000
ccp         86016      1          kvm_amd,     Live       ffffffffc03cd000
kvm         626688     1          kvm_amd,     Live       ffffffffc022d000
irqbypass   16384      1          kvm,         Live       ffffffffc0201000
input_leds  16384      0          -            Live       ffffffffc01d2000
psmouse     151552     0          -            Live       ffffffffc0207000
serio_raw   20480      0          -            Live       ffffffffc01ec000
parport_pc  40960      0          -            Live       ffffffffc01f3000
floppy      81920      0          -            Live       ffffffffc01d7000
parport     53248      2          ppdev,parport_pc, Live       ffffffffc01c4000
sch_fq_codel 20480       2            -            Live       ffffffffc017a000
e1000       139264     0          -            Live       ffffffffc011b000
```

Step 40: Display the list of processes on the target.

Command: info os processes

```
(gdb) info os processes
pid        user       command    cores
1          root       /sbin/init 1
2          root       [kthreadd] 0
3          root       [rcu_gp]   0
4          root       [rcu_par_gp] 0
5          root       [kworker/0:0-eve] 0
6          root       [kworker/0:0H-kb] 0
8          root       [mm_percpu_wq] 0
9          root       [ksoftirqd/0] 0
10         root       [rcu_sched] 0
11         root       [migration/0] 0
12         root       [idle_inject/0] 0
14         root       [cpuhp/0]  0
15         root       [cpuhp/1]  1
16         root       [idle_inject/1] 1
```

Step 41: Display the list of process groups on the target.

Command: info os procgroups

```
(gdb) info os procgroups
pgid       leader command pid        command line
1          systemd    1          /sbin/init
194        systemd-journal 194        /lib/systemd/systemd-journald
201        systemd-udevd 201        /lib/systemd/systemd-udevd
222        systemd-network 222        /lib/systemd/systemd-networkd
224        systemd-logind 224        /lib/systemd/systemd-logind
225        networkd-dispat 225        /usr/bin/python3 /usr/bin/networkd-dispatcher
226        dbus-daemon 226        /usr/bin/dbus-daemon --system --address=systemd: --nofork --nopidfile
232        sshd       232        /usr/sbin/sshd -D
265        dhclient   265        dhclient ens3
266        agetty     266        /sbin/agetty -o -p -- \u --keep-baud 115200,38400,9600 ttyS0 vt220
6807       sshd       6807       sshd: root@pts/0
6825       bash       6825       bash -c clear;/bin/bash
6827       bash       6827       /bin/bash
6842       gdb        6842       gdb -q sample3
```

Step 42: Display the list of threads running on the target.

Command: info os threads

```
(gdb) info os threads
pid        command    tid        core
1          systemd    1          1
2          kthreadd   2          0
3          rcu_gp     3          0
4          rcu_par_gp 4          0
5          kworker/0:0-eve 5          0
6          kworker/0:0H-kb 6          0
8          mm_percpu_wq 8            0
9          ksoftirqd/0 9           0
10         rcu_sched  10         0
11         migration/0 11          0
12         idle_inject/0 12          0
14         cpuhp/0    14         0
15         cpuhp/1    15         1
16         idle_inject/1 16          1
17         migration/1 17          1
18         ksoftirqd/1 18          1
```

Step 43: Display the list of Internet-domain sockets on the target.

Command: info os sockets

```
(gdb) info os sockets
local address local port remote address remote port state      user       family     protocol
0.0.0.0    22         0.0.0.0    0          LISTEN     root       INET       STREAM
10.0.2.15  22         192.255.159.2 38304      ESTABLISHED root       INET       STREAM
0.0.0.0    68         0.0.0.0    0          CLOSE      root       INET       DGRAM
::         22         ::         0          LISTEN     root       INET6      STREAM
```

Value History

Values printed by the print command are saved in the GDB value history and can be checked..

Step 44: Start the GDB with sample3, set a breakpoint at line 21 and run the program.

Commands:
gdb -q sample3
b 21
run 5 6

```
root@localhost:~# gdb -q sample3
Reading symbols from sample3...
(gdb) b 21
Breakpoint 1 at 0x744: file sample3.c, line 21.
(gdb) run 5 6
Starting program: /root/sample3 5 6
Both numbers accepted for addition.

Breakpoint 1, sum_f (x=5, y=6) at sample3.c:21
21              return x+y;
```

Step 45: Print variables to create a value history.

Commands:
print x
print y
print x+y
print x+y+x
print x*y
print x-y

```
(gdb) print x
$1 = 5
(gdb) print y
$2 = 6
(gdb) print x+y
$3 = 11
(gdb) print x+y+x
$4 = 16
(gdb) print x*y
$5 = 30
(gdb) print x-y
$6 = -1
```

Step 46: List the value history.

Command: show values

```
(gdb) show values
$1 = 5
$2 = 6
$3 = 11
$4 = 16
$5 = 30
$6 = -1
```

Convenience Variables

The convenience variables can be used by the user within GDB to hold on to a value and refer to it later.

Step 47: Define a convenience variable.

Command: set $array = cs.arr

```
(gdb) set $array = cs.arr
```

Step 48: Print the freshly defined convenience variable.

Commands:
print $array
print *$array

```
(gdb) print $array
$7 = (int *) 0x555555755028 <cs+8>
(gdb) print *$array
$8 = 0
```

Step 49: Check the convenience variables defined by the user.

Command: show convenience

```
(gdb) show convenience
$array = (int *) 0x555555755028 <cs+8>
$bpnum = 1
$_gdb_minor = 1
$_gdb_major = 9
```

Step 50: There are multiple convenience variables already defined in GDB, so in order to not overwrite an already defined variable, init-if-undefined command. Define a variable and verify.

Commands:
init-if-undefined $var_x = x
show convenience

```
(gdb) init-if-undefined $var_x = x
(gdb) show convenience
$var_x = 5
```

Now if init-if-undefined command is used to define var_x and verify that the value is not overwritten.

Commands:
init-if-undefined $var_x = y
show convenience

```
(gdb) init-if-undefined $var_x = y
(gdb) show convenience
$var_x = 5
```

Produce Core File from Program

A core file or core dump is a file that records the memory image of a running process and its process status (register values etc.).

Step 51: Generate the core dump file for running program

Command: generate-core-file core-dump or **Command:** gcore core-dump

```
(gdb) generate-core-file core-dump
warning: target file /proc/6853/cmdline contained unexpected null characters
Saved corefile core-dump
```

Step 52: Check the present working directory.

Command: ls -l

```
root@localhost:~# ls -l
total 660
-rw-r--r-- 1 root root 656208 Apr 20 11:26 core-dump
-rwxr-xr-x 1 root root  11672 Apr 19 23:49 sample3
-rw-r--r-- 1 root root    738 Apr 19 23:38 sample3.c
```

Copy Between Memory and a File

GDB has the capability to dump, append, and restore to copy data between target memory and a file.

Step 53: Check the memory map of the program.

Command: info proc map

```
(gdb) info proc map
process 6894
Mapped address spaces:

          Start Addr           End Addr       Size     Offset objfile
      0x555555554000     0x555555555000     0x1000        0x0 /root/sample3
      0x555555754000     0x555555755000     0x1000        0x0 /root/sample3
      0x555555755000     0x555555756000     0x1000     0x1000 /root/sample3
      0x555555756000     0x555555777000    0x21000        0x0 [heap]
      0x7ffff7dd2000     0x7ffff7df7000    0x25000        0x0 /lib/x86_64-linux-gnu/libc-2.31.so
      0x7ffff7df7000     0x7ffff7f6f000   0x178000    0x25000 /lib/x86_64-linux-gnu/libc-2.31.so
      0x7ffff7f6f000     0x7ffff7fb9000    0x4a000   0x19d000 /lib/x86_64-linux-gnu/libc-2.31.so
      0x7ffff7fb9000     0x7ffff7fba000     0x1000   0x1e7000 /lib/x86_64-linux-gnu/libc-2.31.so
      0x7ffff7fba000     0x7ffff7fbd000     0x3000   0x1e7000 /lib/x86_64-linux-gnu/libc-2.31.so
      0x7ffff7fbd000     0x7ffff7fc0000     0x3000   0x1ea000 /lib/x86_64-linux-gnu/libc-2.31.so
      0x7ffff7fc0000     0x7ffff7fc6000     0x6000        0x0
      0x7ffff7fcb000     0x7ffff7fce000     0x3000        0x0 [vvar]
      0x7ffff7fce000     0x7ffff7fcf000     0x1000        0x0 [vdso]
      0x7ffff7fcf000     0x7ffff7fd0000     0x1000        0x0 /lib/x86_64-linux-gnu/ld-2.31.so
      0x7ffff7fd0000     0x7ffff7ff3000    0x23000     0x1000 /lib/x86_64-linux-gnu/ld-2.31.so
      0x7ffff7ff3000     0x7ffff7ffb000     0x8000    0x24000 /lib/x86_64-linux-gnu/ld-2.31.so
      0x7ffff7ffc000     0x7ffff7ffd000     0x1000    0x2c000 /lib/x86_64-linux-gnu/ld-2.31.so
      0x7ffff7ffd000     0x7ffff7ffe000     0x1000    0x2d000 /lib/x86_64-linux-gnu/ld-2.31.so
      0x7ffff7ffe000     0x7ffff7fff000     0x1000        0x0
      0x7ffffffde000     0x7ffffffff000    0x21000        0x0 [stack]
  0xffffffffff600000 0xffffffffff601000     0x1000        0x0 [vsyscall]
```

Step 54: Dump the heap memory to a file.

Command: dump memory memory.dump 0x555555756000 0x555555777000

```
(gdb) dump memory memory.dump 0x555555756000 0x555555777000
```

Step 55: Verify the file is created in the present working directory.

Command: ls -l

```
root@localhost:~# ls -l
total 148
-rw-r--r-- 1 root root 135168 Apr 20 11:44 memory.dump
-rwxr-xr-x 1 root root  11672 Apr 19 23:49 sample3
-rw-r--r-- 1 root root    738 Apr 19 23:38 sample3.c
```

References:

1. GDB Documentation (https://sourceware.org/gdb/current/onlinedocs/gdb)