Colby Crutcher

Lab 2

Secure Coding

1. (5 points) What type of file is this, and what kind of security does it have?

- An exe file, and just has read permissions for regular users.

2. (5 points) What are the files that make up this binary, and which one contains main?

- cscd437main.c, cybr437lab2.c, cybr437.h.

cscd437main.c contains teh main function.

3. (5 points) What are the type(s) and name(s) of parameter(s) being passed to main?

- Types - int, and char pointer
- Names - int argc, and char**argv

4. (5 points) Set a breakpoint on each function and display the breakpoints.

```
(gdb) info breakpoints
Num     Type           Disp Enb Address            What
1       breakpoint     keep y   0x00000000000011fc in main at cscd437lab2main.c:5
2       breakpoint     keep y   0x0000000000001275 in fillArray at cybr437lab2.c:6
3       breakpoint     keep y   0x0000000000001457 in printArray at cybr437lab2.c:60
4       breakpoint     keep y   0x0000000000001509 in cleanUp at cybr437lab2.c:76
(gdb)
```

5. (5 points) Begin running the program. How many arguments are passed to main?

- argc = 1 was passed through main, and that is it.

```
(gdb) run
Starting program: /home/ccrutcher/sfiles/main
[Thread debugging using libthread_db enabled]
Using host libthread_db library "/lib/x86_64-linux-gnu/libthread_db.so.1".

Breakpoint 1, main (argc=1, argv=0x7fffffffe388) at cscd437lab2main.c:5
5       {
(gdb)
```

6. (5 points) Step twice and display the contents of the constant and two variables in main.

- int size = 4, and the int pointer myArray is null.

```
(gdb) next
6               int size = 4;
(gdb) next
7               int * myArray = NULL;
```

7. (5 points) Step until you enter the first function called in main. What is the type and name of the constant? Where is the constant declared?

- The constant is an int, name “MAX”, and has a value of 9. It is declared globally in cybr437lab2.c

```
(gdb) step

Breakpoint 2, fillArray (size=0x7fffffffe24c) at cybr437lab2.c:6
6       {
(gdb) list
1       #include "cybr437lab2.h"
2
3       const int MAX = 9;
4
5       int * fillArray(int *size)
6       {
```

8. (5 points) Explain the difference between printing var and var[x]?

- Printing var outputs the memory address, and printing var[x] will print the actual value at index x of the array.

9. (5 points) Instead of continuing this function, return to main and print the current line.

- I used finish to get out of fillArray, annd rant he function until it returned to main. Then I printed the current line using ‘list’.

```
(gdb) finish
Run till exit from #0  fillArray (size=0x7fffffffe24c) at cybr437lab2.c:6
Please enter up to 9 positive (>0) integers. To quit early enter -99

enter pos value (-99 to quit) 4
enter pos value (-99 to quit) 3
enter pos value (-99 to quit) 2
enter pos value (-99 to quit) 4
enter pos value (-99 to quit) 2
enter pos value (-99 to quit) 5
enter pos value (-99 to quit) 6
enter pos value (-99 to quit) 7
enter pos value (-99 to quit) 8
9
0x0000555555555226 in main (argc=1, argv=0x7fffffffe388) at cscd437lab2main.c:9
9               myArray = fillArray(&size);
Value returned is $3 = (int *) 0x5555555592a0
(gdb) list
4       int main(int argc, char**argv)
5       {
6               int size = 4;
7               int * myArray = NULL;
8
9               myArray = fillArray(&size);
10              printArray(size, myArray);
11              size = cleanUp(&myArray);
12
13      }
(gdb)
```

10. (5 points) Step into the second function called from main. What is the name and starting value of the counting variable?

- The function I stepped into is printArray, and the counting variable is named x, and starts at 0.

```
(gdb) step

Breakpoint 3, printArray (size=9, myArray=0x5555555592a0) at cybr437lab2.c:60
60          if(size != 0)
(gdb) list
55
56
57      void printArray(int size, int * myArray)
58      {
59          int x;
60          if(size != 0)
61          {
62              printf("[ ");
63              for(x = 0; x < size - 1; x++)
64                  printf("%d, ", myArray[x]);
(gdb)
```

11. (5 points) Use the disassemble command in GDB to display the assembly code for the function. What does the output show, and how does it correlate with the C source code?

- It shows the dump and you can still see some addresses, jumps (about the extent of Assembly I know about) and the printArray function.

```
(gdb) disassemble
Dump of assembler code for function printArray:
   0x0000555555555444 <+0>:     endbr64
   0x0000555555555448 <+4>:     push   %rbp
   0x0000555555555449 <+5>:     mov    %rsp,%rbp
   0x000055555555544c <+8>:     sub    $0x20,%rsp
   0x0000555555555450 <+12>:    mov    %edi,-0x14(%rbp)
   0x0000555555555453 <+15>:    mov    %rsi,-0x20(%rbp)
=> 0x0000555555555457 <+19>:    cmpl   $0x0,-0x14(%rbp)
   0x000055555555545b <+23>:    je     0x5555555554e7 <printArray+163>
   0x0000555555555461 <+29>:    lea    0xc43(%rip),%rax        # 0x5555555560ab
   0x0000555555555468 <+36>:    mov    %rax,%rdi
   0x000055555555546b <+39>:    mov    $0x0,%eax
   0x0000555555555470 <+44>:    call   0x5555555550d0 <printf@plt>
   0x0000555555555475 <+49>:    movl   $0x0,-0x4(%rbp)
   0x000055555555547c <+56>:    jmp    0x5555555554ae <printArray+106>
   0x000055555555547e <+58>:    mov    -0x4(%rbp),%eax
   0x0000555555555481 <+61>:    cltq
   0x0000555555555483 <+63>:    lea    0x0(,%rax,4),%rdx
   0x000055555555548b <+71>:    mov    -0x20(%rbp),%rax
--Type <RET> for more, q to quit, c to continue without paging--
```

12. (5 points) Delete the breakpoint for the first function called in main and disable the breakpoint for the last function called.

-
```
(gdb) disable 5
(gdb) info breakpoints
Num     Type           Disp Enb Address            What
1       breakpoint     keep y   0x00005555555551fc in main at cscd437lab2main.c:5
        breakpoint already hit 1 time
2       breakpoint     keep y   0x0000555555555275 in fillArray at cybr437lab2.c:6
        breakpoint already hit 1 time
5       breakpoint     keep n   0x0000555555555509 in cleanUp at cybr437lab2.c:76
(gdb)
```

13. (5 points) Without starting over, AKA your current location in a function that is not main, print the memory location of the first variable passed to main.

- I was in cleanup, so I had to switch back to mains stack fram, and print the address of argc.

```
(gdb) backtrace
#0  cleanUp (myArray=0x7fffffffe250) at cybr437lab2.c:76
#1  0x0000555555555247 in main (argc=1, argv=0x7fffffffe388) at cscd437lab2main.c:11
(gdb) frame 1
#1  0x0000555555555247 in main (argc=1, argv=0x7fffffffe388) at cscd437lab2main.c:11
11              size = cleanUp(&myArray);
(gdb) print &argc
$4 = (int *) 0x7fffffffe23c
(gdb)
```

14. (5 points) Print the information on the current running threads. How many threads are running?

- There is 1 thread running.

```
(gdb) info threads
  Id   Target Id                                  Frame
* 1    Thread 0x7ffff7fb4740 (LWP 37542) "main" cleanUp (myArray=0x7fffffffe250) at cybr437lab2.c:76
(gdb)
```

15. (5 points) Enable the breakpoint for the third function called by main. What is the type, name, and memory location of the variable passed to it?

- The third function is cleanUp(&myArray). It is an int pointer. While in cleanup I printed the value using 'print myArray'.

```
(gdb) print myArray
$5 = (int *) 0x5555555592a0
(gdb) print *myArray
$6 = 4
(gdb) print &myArray
$7 = (int **) 0x7fffffffe250
(gdb)
```