Overview

Phila Dlamini
Labib A. Afia
Bomb number: 65
Help and collaboration:

- The following resources were used to complete this assignment:
    - The GNU Debugger (GDB). We used this to step through assembly instructions and test our hypotheses about what the code was doing
    - The AMD64 cheat sheet from Brown. We used this resource together with the AMD64 Architecture Programmer's Manual to look up instructions we did not understand.
    - The AMD64 Architecture Programmer's Manual, Volume 3. This was another useful resource in looking up instructions

Hours spent on assignment:

- 25 hours

Defuse

The following lines were passed into the bomb to defuse it. Each line corresponds to a phase. Note that the string "Gregory_Anton," which appears after the number 2 on line 4 below, unlocks the secret phase. After unlocking the secret phase, we then passed the number 36 into standard input to defuse it

```
If I could only get inside that brain of yours
1 2 6 24 120 720
0 b 788
2 Gregory_Anton
aaaaae
660
```

Description

Phase 1

- This phase expects the string “If I could only get inside that brain of yours” as input. If any other string is passed in, the bomb explodes
- The helper function `strings_not_equal` is used to test for string equality

Phase 2

- This phase expects a sequence of six numbers. If less than 6 numbers are passed in, the bomb explodes
- The first number must be 1, and all numbers that follow after must be that number’s position times the previous number. That is, the second number should be 2 $\times$ 1 = 2, the third number 3 $\times$ 2 = 6, the fourth 4 $\times$ 6 = 24, and so on
- The resulting sequence is thus $1\ 2\ 6\ 24\ 120\ 720$

Phase 3

- This phase uses switch statements, as evidenced by the jump table seen in the assembly. We speculate that an enum type with seven possible values is used in the switch statement, since if a value bigger than 7 is passed in, the bomb explodes
    - This phase expects three inputs. The first must be a number between 0 and 7, and this number decides which of the switch cases to run. Ultimately, this value decides what the other two inputs should be. We chose to pass in 0 as the first argument
    - The second value has to be a character. Since we’d passed in 0 as the first value, the expected character was a ‘b’ (we figured this out by looking at the assembly)
    - The last value had to be another number. In our case, this last value had to be 788. We again figured this out by examining the assembly
- The correct input therefore was $0\ b\ 788$

Phase 4

- This phase expects a number. The number had to be greater than 0, or else the bomb would explode
- The phase then calculates 7 raised to the power of the number passed in. We saw that the result had to be 49, and so the correct input was 2
- Notice the second string, “Gregory_Anton,” that comes after the number 2. This string is key to unlocking the secret phase, as described above.

Code

Phase 5:

In this phase, we have an array that contains the following numbers.
[2, 10, 6, 1, 12, 16, 9, 3, 4, 7, 14, 5, 11, 8, 15, 13]

The phase expects a string of length 6 as input. If a string of length not 6 is passed in, explode _bomb() is called. After verifying that the string has the correct length, the phase then iterates through each character of the string, retrieves the least 4 significant bits of the character's ASCII value, and then uses that value as an index in the array. A sum variable accumulates the total sum of the values index.

The accumulated sum needs to be 36 to defuse the bomb.

Our C code for this phase therefore looks like the following

```c
#include <stdio.h>
#include <stdlib.h>

static int arr[] = {2, 10, 6, 1, 12, 16, 9, 3, 4, 7, 14, 5, 11, 8, 15, 13};

void phase_defused() {
   printf("phase5 defused\n");
}

void explode_bomb() {
   printf("BOMB!\n");
}

int string_length(char *input)
{
    int length = 0;
    while(input[length++] != '\0');
    return length - 1;
}

void phase5 (char* input) {
   int sum = 0;
```

```
    if(string_length(input) != 6) {
        explode_bomb();
    }

    for(int i = 0; i < 6; i++) {
        char mask = 0xf;
        int index = *(input+i) & mask;
        sum += arr[index];
    }

    if(sum == 66) {
        phase_defused();
    } else {
        explode_bomb();
    }

}

int main() {
    phase5("aaaaae");
    return 0;
}
```

Phase 6:

In this phase, we start with the following linked list of numbers:
157 → 204 → 660 → 494 → 718 → 288 → 653 → 466 → 554 → NULL

The phase then reads in the input value passed in and creates a node (node 0) that contains this value. The correct input value that defuses this phase is 660 (we explain why below). The updated list, therefore, looks like the following:
660 → 157 → 204 → 660 → 494 → 718 → 288 → 653 → 466 → 554 → NULL

After this, the function `fun6()` is called. `fun6()` performs insertion sort on the linked list and sorts it in decreasing order, and returns the head of the sorted list. The linked returned is thus the following:
718 → 660 → 660 → 653 → 554 → 494 → 466 → 288→ 204 → 157→ NULL

From stepping through the assembly, we see that the phase defuses if the value of the second node after the first (that is 660) must equal the value passed in. We decided then that the correct input was 660.

```
#include <stdio.h>
#include <stdlib.h>

typedef struct node* node;
struct node {
   long num;
   node next;
};

/* Create linked list */
static struct node ninth   =    {554, NULL};
static struct node eighth  =    {466, &ninth};
static struct node seventh =    {653, &eighth};
static struct node sixth   =    {288, &seventh};
static struct node fifth   =    {718, &sixth};
static struct node fourth  =    {494, &fifth};
static struct node third   =    {660, &fourth};
static struct node second  =    {204, &third};
static struct node first   =    {157, &second};


node fun6(node head) {
   node sortedHead = NULL;
   node unsortedHead = head;

   while(unsortedHead != NULL) {
       if(sortedHead == NULL) {
           sortedHead = head;
           node placer = unsortedHead->next;
           sortedHead->next = NULL;
           unsortedHead = placer;
       } else {
           node toInsert = unsortedHead;
           unsortedHead = unsortedHead->next;
           node curr = sortedHead;
           node prev = NULL;
           int count = 0;
           while(curr != NULL && curr->num > toInsert->num) {
```

```
                prev = curr;
                curr = curr->next;
                count++;
            }
            if(count == 0) {
                //push front
                toInsert->next = sortedHead;
                sortedHead = toInsert;
            } else {
                //prev point to inserted
                //inserted point to what used to be after prev
                node after = prev->next;
                prev->next = toInsert;
                toInsert->next = after;
            }
        }
    }
    return sortedHead;
}

void phase_defused() {
    printf("phase6 defused\n");
}

void explode_bomb() {
    printf("BOOM!\n");
}

void phase6() {
    static struct node zeroth = {660, &first};
    node newHead = fun6(&zeroth);
    if (newHead->next->num == 660) {
        phase_defused();
    } else {
        explode_bomb();
    }
}

int main() {
    phase6();
    return 0;
}
```