Lab Report - Exercise 1

Gitesh Gund
(gund@rhrk.uni-kl.de) Group **3**

Ashitha Mudraje
(bud07nyc@rptu.de) Group **3**

Anas Ahmad
(aahmad@rhrk.uni-kl.de) Group **3**

Vinson Noronha
(noronha@rptu.de) Group **9**

March 23, 2024

1 Abstract

Short summary of key findings.

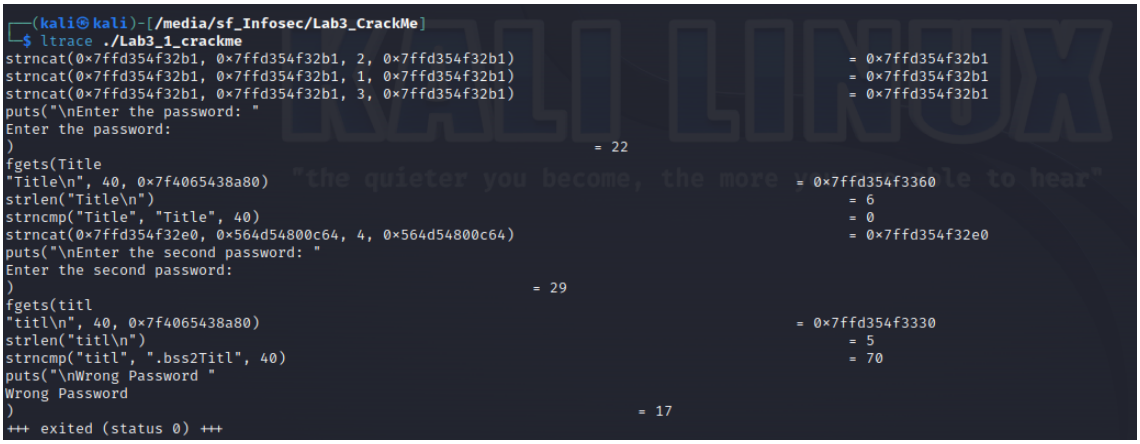
1. We found passwords of encrypted file using tools(strings, ltrace, gdb).
2. We examined including all seven subcalculations algorithm and reconstruct the steps involved while considering the changes made to remove the time syncing mechanism and add a different step.

2 Crack Me

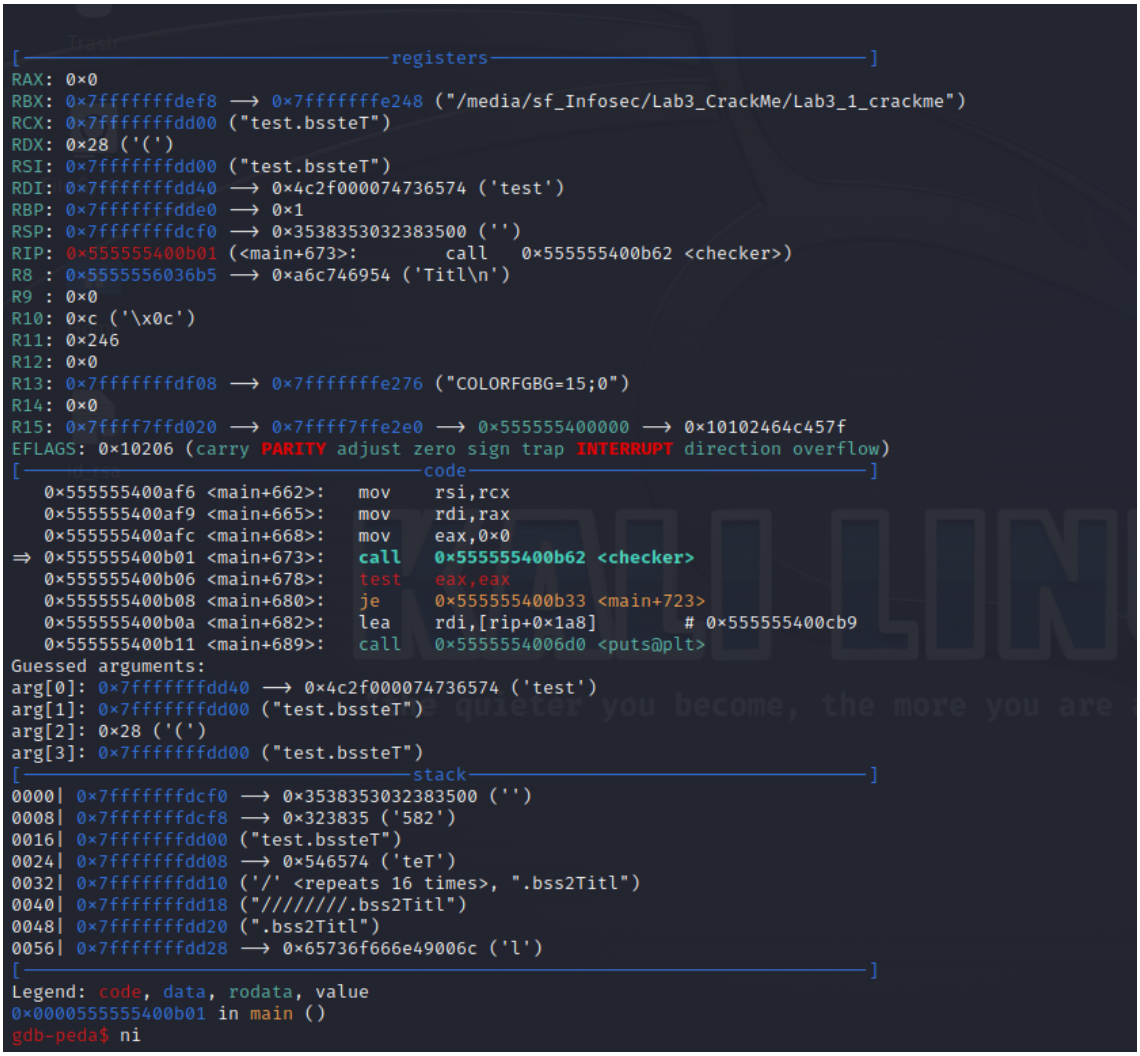
We identified passwords of encrypted file using debugging tools. The first password is found using "Strings" command and we observed that first password is Title.

```
┌──(kali㉿kali)-[/media/sf_Infosec/Lab3_CrackMe]
└─$ strings ./Lab3_1_crackme
/lib64/ld-linux-x86-64.so.2
libc.so.6
strncmp
puts
stdin
printf
fgets
strlen
strncat
__cxa_finalize
__libc_start_main
_ITM_deregisterTMCloneTable
__gmon_start__
_Jv_RegisterClasses
_ITM_registerTMCloneTable
GLIBC_2.2.5
.bssf
test
5820
AWAVA
AUATL
[]A\A]A^A_
Title
Enter the password:
Enter the second password:
Enter the third password:
Congratulations
Flag: %s
Wrong Password
;*3$"
GCC: (Debian 6.3.0-18+deb9u1) 6.3.0 20170516
crtstuff.c
__JCR_LIST__
deregister_tm_clones
__do_global_dtors_aux
completed.6972
__do_global_dtors_aux_fini_array_entry
frame_dummy
__frame_dummy_init_array_entry
auth.c
__FRAME_END__
__JCR_END__
__init_array_end
_DYNAMIC
__init_array_start
__GNU_EH_FRAME_HDR
_GLOBAL_OFFSET_TABLE_
```

We discovered second password using "ltrace" command and it was .bss2Titl



Lastly we found Third password using "gdb" command. We set a breakpoints in main and identified password as test.bssteT



Finally we entered all three password and found flag.

```
┌──(kali㉿kali)-[/media/sf_Infosec/Lab3_CrackMe]
└─$ ./Lab3_1_crackme

Enter the password:
Title

Enter the second password:
.bss2Titl

Enter the third password:
test.bssteT

Congratulations
Flag: 5820585582
```

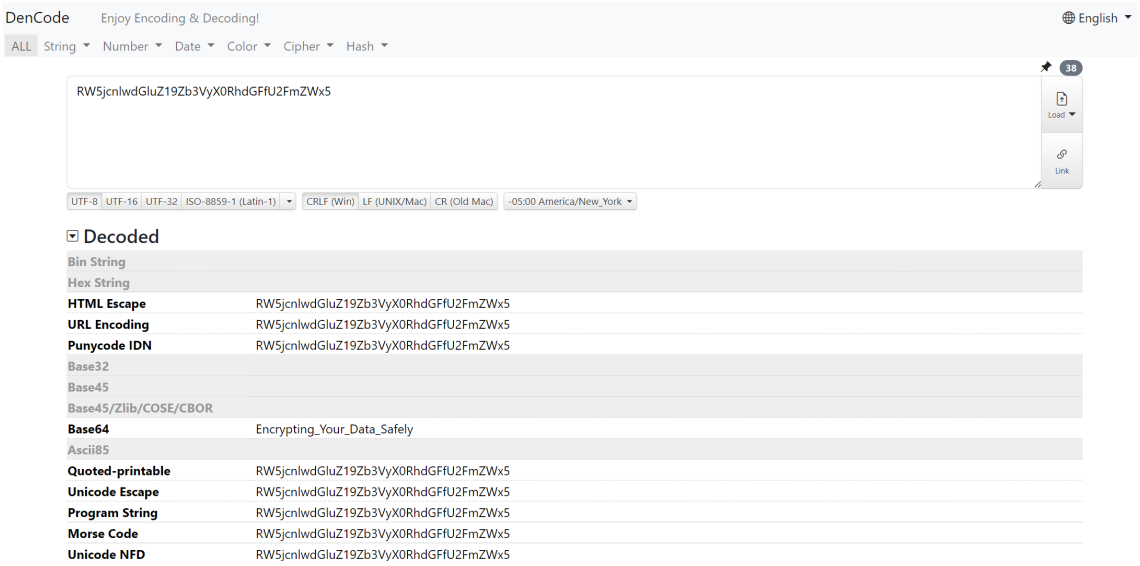
3 Reverse Engineering

For reverse engineering we were given a file **communicate secret**, this is the binary that we need to reverse engineer. First we made the file executable using the command "chmod $+x$ communicate-secret". We were also given two seperate encrypted files **secret1.txt** and **secret2.txt**.

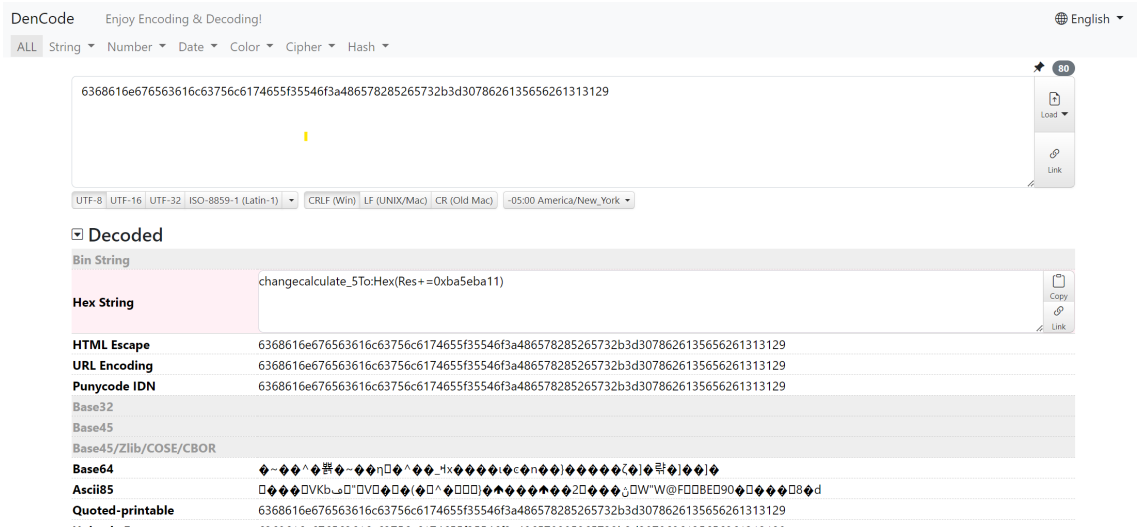
3.1 Decoding secret text strings

The text files provided were encoded using different encoders.

1. The secret1.txt file contains the text that should be used with the binary but this is encoded in an unknown format. We decoded this file using online decoder **https://dencode.com/en/**. The online tool shows that the encoding used on this string was base64.

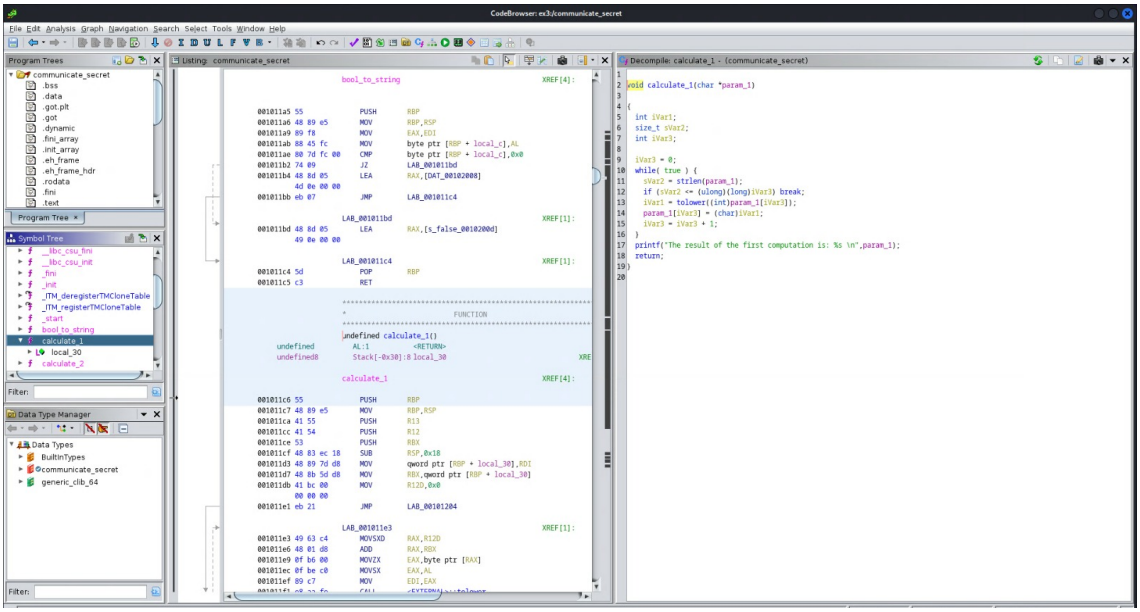


2. The secret2.txt file contains the changes that should be made in the binary to time synching mechanism. We decoded this file using the same online tool as used for the secret1 file. The online tool shows that this file was encoded as hex string.



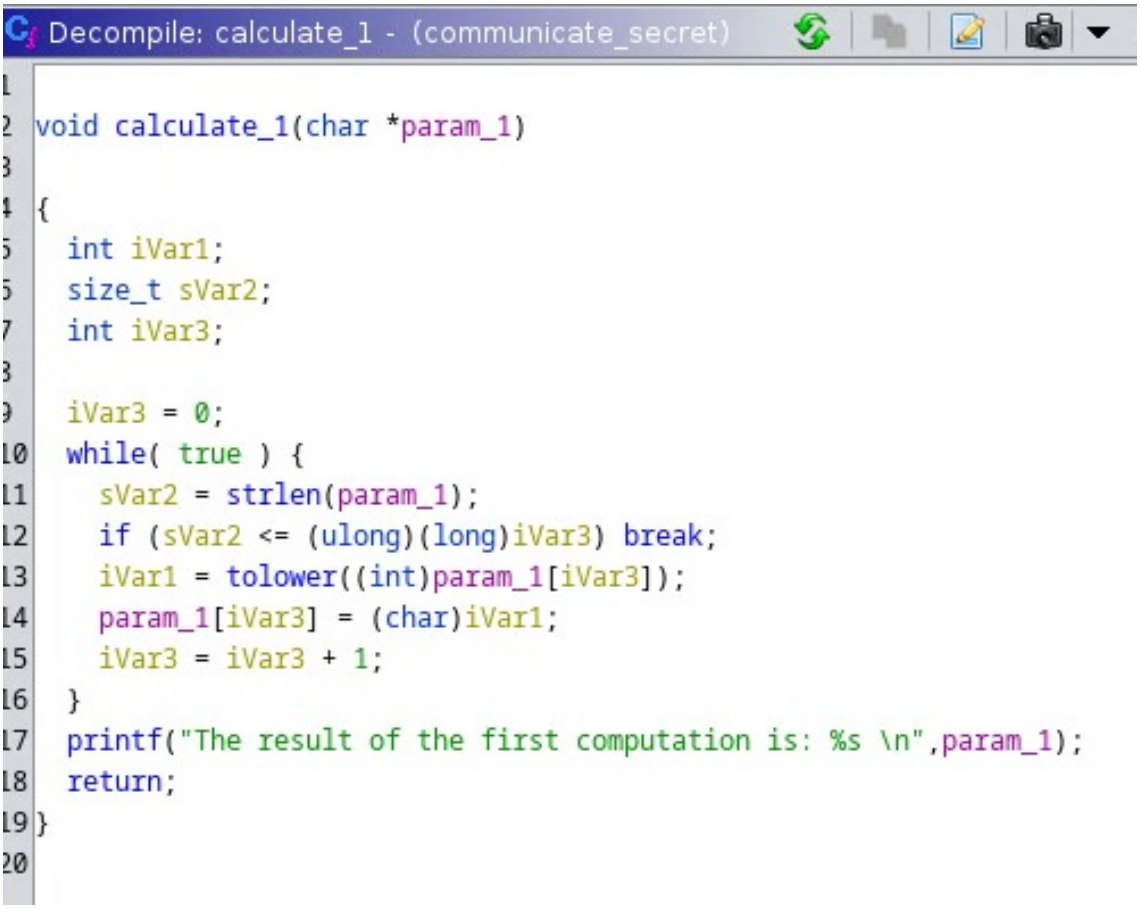
3.2 Reversing

To reverse engineer the communicate secret binary file we used a tool named "Ghidra". It is available to be used with Kali Linux. We installed it using the "apt install" command, opened a new project in it and imported our binary that we need to reverse engineer and examine. Ghidra gives a nice organized view of the binary, with all the functions in the symbol tree section on left side and their decompiled coode in the right panel of the binary code. This made our analysis much more easier.



There were seven calculate functions in the binary, we will explain all of them one by one:

1. **Function: calculate 1**



The first calculate function calculates the length of the string that we enter as an input to the program and convert each letter of the string to lower case using the tolower() function.

2. **Function: calculate 2**

Decompile: calculate_2 - (communicate_secret)

```
1
2  long calculate_2(char *param_1)
3
4  {
5    size_t sVar1;
6    long lVar2;
7    int iVar3;
8
9    lVar2 = 0;
10   iVar3 = 0;
11   while( true ) {
12     sVar1 = strlen(param_1);
13     if (sVar1 <= (ulong)(long)iVar3) break;
14     lVar2 = lVar2 + param_1[iVar3];
15     iVar3 = iVar3 + 1;
16   }
17   printf("The result of the second computation is: %lu \n",lVar2);
18   return lVar2;
19 }
20
```

The second calculate function uses a while loop with always true condition and adds the ASCII value of each character of the input string and returns the sum of the ASCII values.

3. **Function: calculate 3**

Decompile: calculate_3 - (communicate_secret)

```
1
2  long calculate_3(char *param_1,long param_2)
3
4  {
5    size_t sVar1;
6
7    sVar1 = strlen(param_1);
8    printf("The result of the third computation is: %lu \n",param_2 + sVar1);
9    return param_2 + sVar1;
10 }
11
```

The third calculate function is stroring the length of the input string using strlen() function, then adds that length to the output of second calculate function and returns the final value as output

4. **Function: calculate 4**

[Decompile: calculate_4] - (communicate_secret)

```
1
2  long calculate_4(char *param_1,long param_2)
3
4  {
5    char *__s2;
6    int iVar1;
7    size_t sVar2;
8    ulong uVar3;
9    undefined8 uVar4;
10   int iVar5;
11   int iVar6;
12   undefined8 uStack_90;
13   long local_88 [4];
14   long local_68;
15   char *local_60;
16   char *local_50;
17   size_t local_48;
18   int local_3c;
19
20   iVar6 = 0x32;
21   uStack_90 = 0x10131b;
22   local_68 = param_2;
23   local_60 = param_1;
24   sVar2 = strlen(param_1);
25   if (sVar2 == 1) {
26     param_2 = param_2 * 0x32;
27     uStack_90 = 0x101350;
28     printf("The result of the fourth computation is: false, %lu \n",param_2);
29   }
30   else {
31     uStack_90 = 0x101360;
32     local_48 = strlen(param_1);
33     local_88[2] = local_48 + 1;
34     local_88[3] = 0;
35     local_88[0] = local_48 + 1;
36     local_88[1] = 0;
37     uVar3 = (local_48 + 0x10) / 0x10;
38     local_50 = (char *)(local_88 + uVar3 * -2);
39     local_3c = 0;
40     (&uStack_90)[uVar3 * -2] = 0x1013c1;
41     sVar2 = strlen(param_1);
42     __s2 = local_50;
43     iVar5 = (int)sVar2;
44     while (iVar1 = local_3c, iVar5 = iVar5 + -1, -1 < iVar5) {
45       local_3c = local_3c + 1;
46       local_50[iVar5] = param_1[iVar1];
47     }
```

```
48    (&uStack_90)[uVar3 * -2] = 0x1013fc;
49    iVar5 = strcmp(param_1,__s2);
50    if (iVar5 == 0) {
51      iVar6 = 100;
52    }
53    param_2 = param_2 * iVar6;
54    (&uStack_90)[uVar3 * -2] = 0x101420;
55    uVar4 = bool_to_string(iVar6 != 0x32);
56    (&uStack_90)[uVar3 * -2] = 0x101437;
57    printf("The result of the fourth computation is: %s, %lu \n",uVar4,param_2);
58  }
59  return param_2;
60}
```

The fourth calculate function checks if the length of string is 1, if the length is 1 it multiplies the result of calculate3 with 50. Otherwise, it reverses the string and checks if it is palindrome then it returns true and multiplies the result of calculate3 with 100, else if it is not a palindrome then it returns false and multiplies the result of calculate3 with 50.

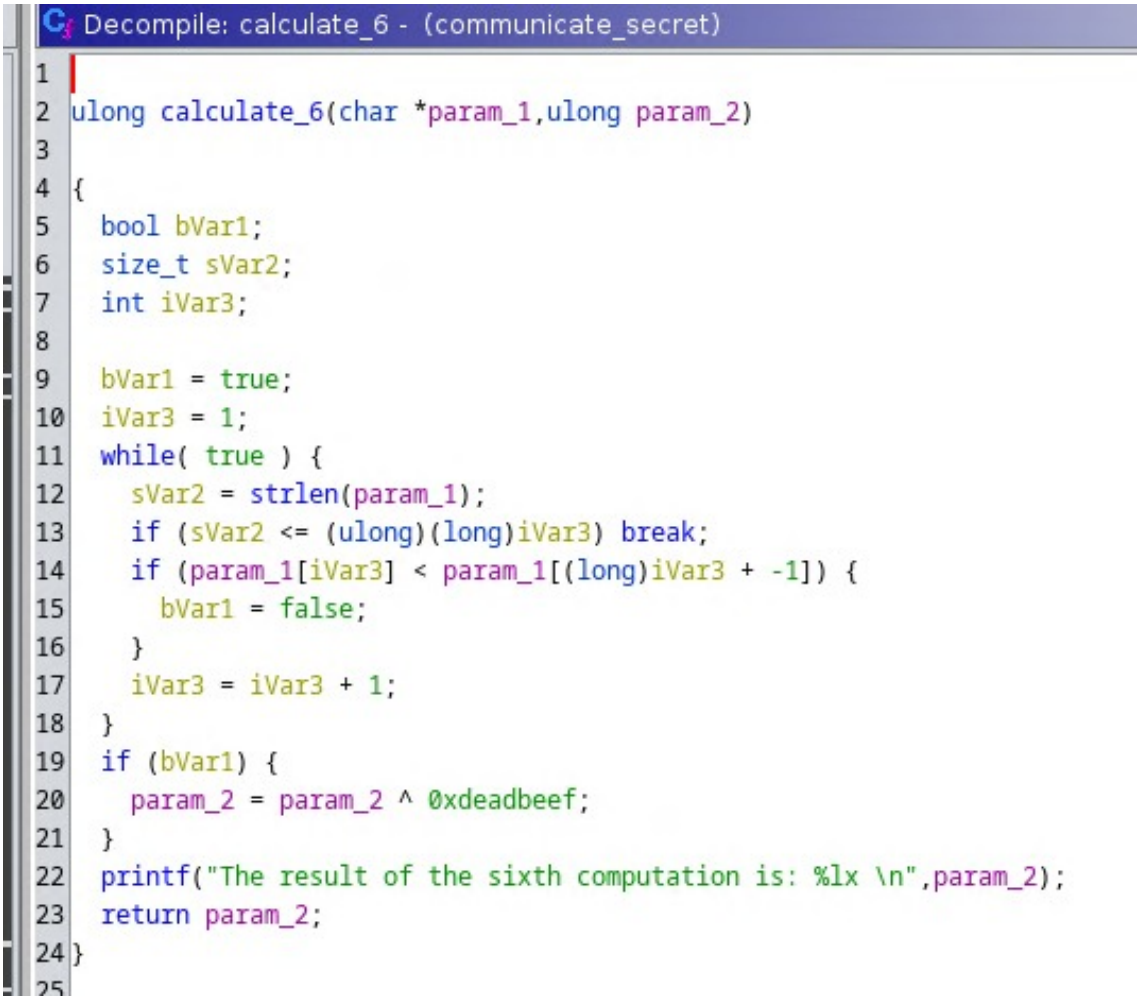
5. **Function: calculate 5**

Decompile: calculate_5 - (communicate_secret)

```
1
2  long calculate_5(undefined8 param_1,long param_2)
3
4  {
5    time_t tVar1;
6
7    tVar1 = time((time_t *)0x0);
8    printf("The result of the fifth computation is: %lx \n",param_2 + tVar1);
9    return param_2 + tVar1;
10 }
11
```

The fifth calculate function takes the current time from system converts it to hex value using type casting and then adds the hex value of time to the output of previous function i.e calculate4.

6. **Function: calculate 6**



The sixth calculate function check if the string character are in the increasing order, if the answer is true, then it XOR the output of 5th calculate function with a hardcoded value 0xdeadbeef. Otherwise, it returns the same result as calculate5.

7. **Function: calculate last**

Decompile: calculate_last - (communicate_secret)

```
1
2  long calculate_last(char *param_1,long param_2)
3
4  {
5    uint uVar1;
6    char *pcVar2;
7
8    pcVar2 = strchr(param_1,0x61);
9    uVar1 = (uint)(pcVar2 != (char *)0x0);
10   pcVar2 = strchr(param_1,0x65);
11   if (pcVar2 != (char *)0x0) {
12     uVar1 = 1;
13   }
14   pcVar2 = strchr(param_1,0x69);
15   if (pcVar2 != (char *)0x0) {
16     uVar1 = 1;
17   }
18   pcVar2 = strchr(param_1,0x6f);
19   if (pcVar2 != (char *)0x0) {
20     uVar1 = 1;
21   }
22   pcVar2 = strchr(param_1,0x75);
23   if (pcVar2 != (char *)0x0) {
24     uVar1 = 1;
25   }
26   puts(
27       "The result of the last computation is: ???\nThis one you\'ll have to solve without debugging
       outputs :D "
28       );
29   return param_2 + (int)uVar1;
30 }
31
```

The last calculate function checks whether any vowel "a,e,i,o,u" is present in the input string, if there is any vowel then it increments one in the output of the sixth calculate function and returns the incremented value.

3.3 Reconstructing Algorithm

After understanding the binary and decoding the secret2, we made a program in python that includes all the seven sub calculations. We made the changes in the 5th calculate function by replacing the time syncing mechanism with the value **0xba5eba11** in the return value of the calculate 5. This is the python code that we created.

```
1  def reversed_algo(param_1):
2  str_length = len(param_1)
3  # calculate_1
4  if str_length <= 0: return
5  param_1= param_1.lower()
6  print("The result of the first computation is:",param_1)
7
8  #calculate_2
9  param_2 = 0
10 for i in range(str_length):
11     param_2+=ord(param_1[i])
12 print("The result of the second computation is:",param_2);
13
14 #calculate_3:
15 param_3 = str_length + param_2
16 print("The result of the third computation is:", param_3)
17
18 #calculate_4:
19 param_4 = 0
20 ivar = 50
21 isPalindrome = False
```

```
if str_length !=1 and param_1 == param_1[::-1]:
    ivar =100
    isPalindrome = True
param_4 = param_3 * ivar
print("The result of the fourth computation is:", isPalindrome, param_4)

#calculate_5:
param_5 = hex(param_4 + int('0xba5eba11', 16))
print("The result of the fifth computation is:", param_5)

#calculate_6:
param_6 = param_5
chars = list(param_1)
if chars == sorted(param_1):
    param_6 = hex(int(param_5, 16) ^ int('0xdeadbeef', 16))
print("The result of the sixth computation is:", param_6)

#calculate_7:
vowels = {'a', 'e', 'i', 'o', 'u'}
uVar1 = int(any(char in vowels for char in param_1))
param_7 = hex(int(param_6,16) + uVar1)
print("The result of the last computation is:", param_7)

# Example usage
str_ip= input("Enter String: ")
if not str_ip:
    print("Please give a string as an input!")
reversed_algo(str_ip)
```

By making the changes in our python code and giving $Encrypting_Your_Data_Safely$ as input which is decoded string from secret1 we get the desired output and the second flag which is the output of seventh calculate function.

```
┌──(jetsunburst㉿LAP-GIGU)-[~]
└─$ python ./ex3.py
Enter String: Encrypting_Your_Data_Safely
The result of the first computation is: encrypting_your_data_safely
The result of the second computation is: 2893
The result of the third computation is: 2920
The result of the fourth computation is: False 146000
The result of the fifth computation is: 0xba60f461
The result of the sixth computation is: 0xba60f461
The result of the last computation is: 0xba60f462
```