Sheldon Phase 1- Writeup

Objective of this report is to introduce the used approach to find the passphrase for the phase_1 of the Sheldon_1 binary file.

Firstly, launched the sheldon1 file through GDB to analyze the assembly code [Figure 1]. After running the 'info functions' command was able to retrieve all the functions related to the sheldon1 file. By examining the structure, I have realized that the program has a main function and respective functions related to each stage namely phase_1, phase_2, phase_3, phase_4, phase_5 and phase_6.

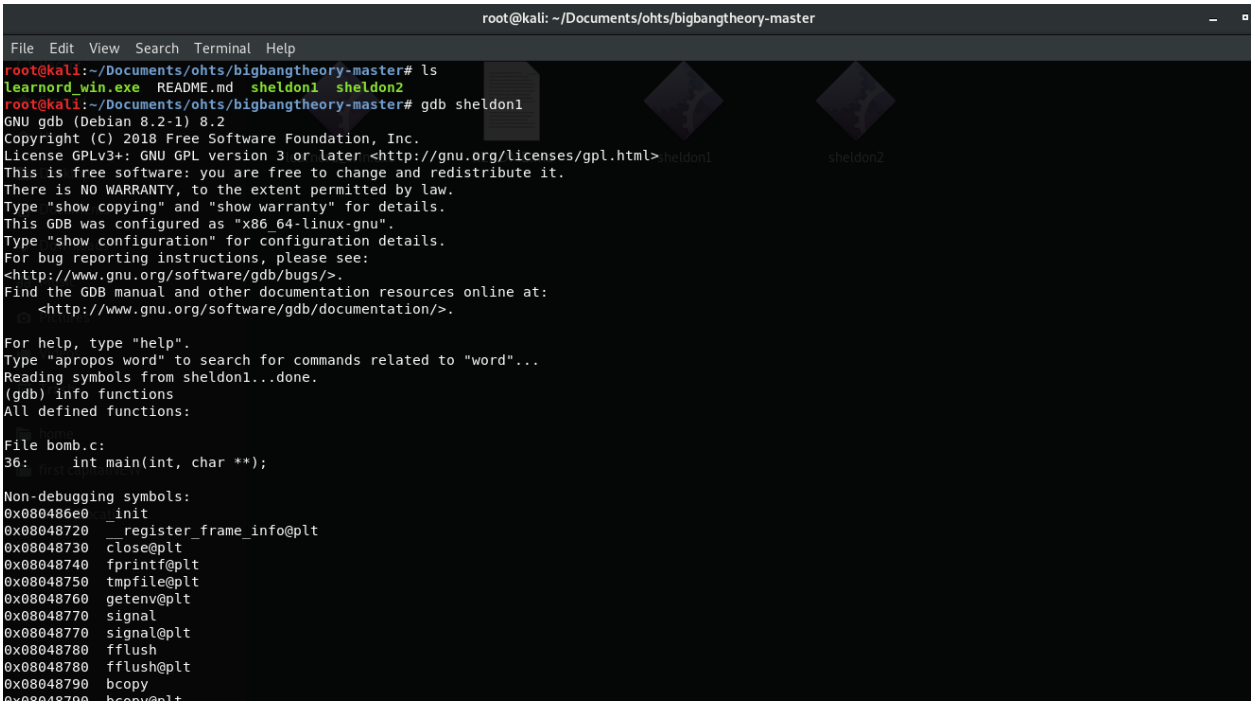

Figure 1: Launch of GDB

Then decided to disassemble the main function [Figure 2] to examine the behavior of the program. Also, by executing the program it was easy to identify the accepting mechanism of the passphrases. After running the binary file, the password for the phase_1 was needed to continue with the program. Since we must start from the beginning to feed passphrases, why not start disassembling with phase_1?

```
root@kali: ~/Documents/ohts/bigbangtheory-master
File  Edit  View  Search  Terminal  Help
--Type <RET> for more, q to quit, c to continue without paging--
   0x08048a15 <+101>:   push   %eax
   0x08048a16 <+102>:   push   $0x804963f
   0x08048a1b <+107>:   call   0x8048810 <printf@plt>
   0x08048a20 <+112>:   add    $0xfffffff4,%esp
   0x08048a23 <+115>:   push   $0x8
   0x08048a25 <+117>:   call   0x8048850 <exit@plt>
   0x08048a2a <+122>:   lea    0x0(%esi),%esi
   0x08048a30 <+128>:   call   0x8049160 <initialize_bomb>
   0x08048a35 <+133>:   add    $0xfffffff4,%esp
   0x08048a38 <+136>:   push   $0x8049660
   0x08048a3d <+141>:   call   0x8048810 <printf@plt>
   0x08048a42 <+146>:   add    $0xfffffff4,%esp
   0x08048a45 <+149>:   push   $0x80496a0
   0x08048a4a <+154>:   call   0x8048810 <printf@plt>
   0x08048a4f <+159>:   add    $0x20,%esp
   0x08048a52 <+162>:   call   0x80491fc <read_line>
   0x08048a57 <+167>:   add    $0xfffffff4,%esp
   0x08048a5a <+170>:   push   %eax
   0x08048a5b <+171>:   call   0x8048b20 <phase_1>
   0x08048a60 <+176>:   call   0x804952c <phase_defused>
   0x08048a65 <+181>:   add    $0xfffffff4,%esp
   0x08048a68 <+184>:   push   $0x80496e0
   0x08048a6d <+189>:   call   0x8048810 <printf@plt>
   0x08048a72 <+194>:   add    $0x20,%esp
   0x08048a75 <+197>:   call   0x80491fc <read_line>
   0x08048a7a <+202>:   add    $0xfffffff4,%esp
   0x08048a7d <+205>:   push   %eax
   0x08048a7e <+206>:   call   0x8048b48 <phase_2>
   0x08048a83 <+211>:   call   0x804952c <phase_defused>
   0x08048a88 <+216>:   add    $0xfffffff4,%esp
   0x08048a8b <+219>:   push   $0x8049720
   0x08048a90 <+224>:   call   0x8048810 <printf@plt>
   0x08048a95 <+229>:   add    $0x20,%esp
   0x08048a98 <+232>:   call   0x80491fc <read_line>
   0x08048a9d <+237>:   add    $0xfffffff4,%esp
   0x08048aa0 <+240>:   push   %eax
--Type <RET> for more, q to quit, c to continue without paging--
```

Figure 2: Disassembled Main Function

After disassembling the phase_1 function [Figure 3], it was noted that a value is being pushed to the stack before calling the 'strings_not_equal' function. That looks like something to be suspicious about. So, let's see what's inside of this particular memory location. And there it is, we found a string in the 0X80497C0 address.

```
root@kali: ~/Documents/ohts/bigbangtheory-master
File  Edit  View  Search  Terminal  Help
   0x08048b16 <+358>:   mov    -0x18(%ebp),%ebx
   0x08048b19 <+361>:   mov    %ebp,%esp
   0x08048b1b <+363>:   pop    %ebp
   0x08048b1c <+364>:   ret
End of assembler dump.
(gdb) disass phase_1
Dump of assembler code for function phase_1:
   0x08048b20 <+0>:     push   %ebp
   0x08048b21 <+1>:     mov    %esp,%ebp
   0x08048b23 <+3>:     sub    $0x8,%esp
   0x08048b26 <+6>:     mov    0x8(%ebp),%eax
   0x08048b29 <+9>:     add    $0xfffffff8,%esp
   0x08048b2c <+12>:    push   $0x80497c0
   0x08048b31 <+17>:    push   %eax
   0x08048b32 <+18>:    call   0x8049030 <strings_not_equal>
   0x08048b37 <+23>:    add    $0x10,%esp
   0x08048b3a <+26>:    test   %eax,%eax
   0x08048b3c <+28>:    je     0x8048b43 <phase_1+35>
   0x08048b3e <+30>:    call   0x80494fc <explode_bomb>
   0x08048b43 <+35>:    mov    %ebp,%esp
   0x08048b45 <+37>:    pop    %ebp
   0x08048b46 <+38>:    ret
End of assembler dump.
(gdb) x/s 0x08048b3a
0x8048b3a <phase_1+26>: "\205\300t\005\350\271\t"
(gdb) x/10c 0x08048b3a
0x8048b3a <phase_1+26>: -123 '\205'     -64 '\300'      116 't' 5 '\005'        -24 '\350'      -71 '\271'      9 '\t'  0 '\000'
0x8048b42 <phase_1+34>: 0 '\000'        -119 '\211'
(gdb) x/s 0x08048b32
0x8048b32 <phase_1+18>: "\350\371\004"
(gdb) x/s 0x08048b31
0x8048b31 <phase_1+17>: "P\350\371\004"
(gdb) x/s 0x08048b2c
0x8048b2c <phase_1+12>: "h\300\227\004\bP\350\371\004"
(gdb) x/s 0x80497c0
0x80497c0:      "Public speaking is very easy."
(gdb)
```

Figure 3: Disassembled Main Function and Memory Analyze

Now it’s time to test whether the bomb is friendly or not with the found passphrase [Figure 4].

Figure 4: Testing the Passphrase

Here we go! By using the ‘Public speaking is very easy.’ string we were able to diffuse the phase_1.