UNIVERSITI TEKNIKAL MALAYSIA MELAKA

FACULTY OF INFORMATION AND COMMUNICATION TECHNOLOGY

COURSE: BITZ

BITS 3453 MALWARE ANALYSIS & DIGITAL INVESTIGATION

ASSIGNMENT 1

GROUP KeepAlive

LECTURER: TS. DR. MOHD ZAKI BIN MAS'UD

Name	Metric No
Muhammad Izham Bin Norhamadi	B032020039
Ahmad Sha Herizam Bin Tahir	B032020009
Affendy Elyas bin Azhari Sharidan	B032020024
Muhammad Rifqi Bin Ramlan	B032020028

TABLE OF CONTENTS

INTRODUCTION

In this modern world, all information that flows in and out of the internet is having risk of being infected by malware or malicious viruses. This includes any type of file ranging from executable file to picture since these viruses can reside silently inside it. This is not only common in windows medium but also concerning Linux too. That's why analyzation of these suspicious type of file need to be pay attention to and every technician and programmer need to have set amount of analyze skills to make sure they can detect any skeptical piece of codes during the process of reverse-engineering. This is because any malicious code can be as perfect as possible to be hard to detect by anyone without the analysis skills needed.

Type of analysis can be divided into two types which are static analysis and dynamic analysis. Up until now, there's still an ongoing debate on which type is the best one. Truthfully, these two types of analysis have their own pros and cons as well as their own specific function as well. Static analysis is analysis of source code automatically without the execution of the application. Meanwhile, dynamic analysis is evaluation of program of a source code in real time. Even though the definition of these analysis is different, the main goals of these analysis are the same which are to identify or break down the source code to detect any malicious piece of code. Obviously, to perform any of this analysis, some software is needed such as Ghidra, IDA pro or EDB debugger. These is really recommended to use this as it is as easy as ABC to use and it's user-friendly.

In this report, we will break down two puzzle codes provided by our lecturer which are PE analysis and ELF analysis. By using the known reverse-engineering software such as Ghidra and others to solve the given puzzles. The information of the tools used, and the procedure used will be recorded below.

PE ANALYSIS

1) Tools used for EXE analysis

- Ghidra

  Ghidra for windows - A free and open-source reverse engineering (RE) tools developed by NSA's Research Directorate. The sources of tools were published GitHub. Its strength comes from decompiling program to C language useful for static malware analysis.

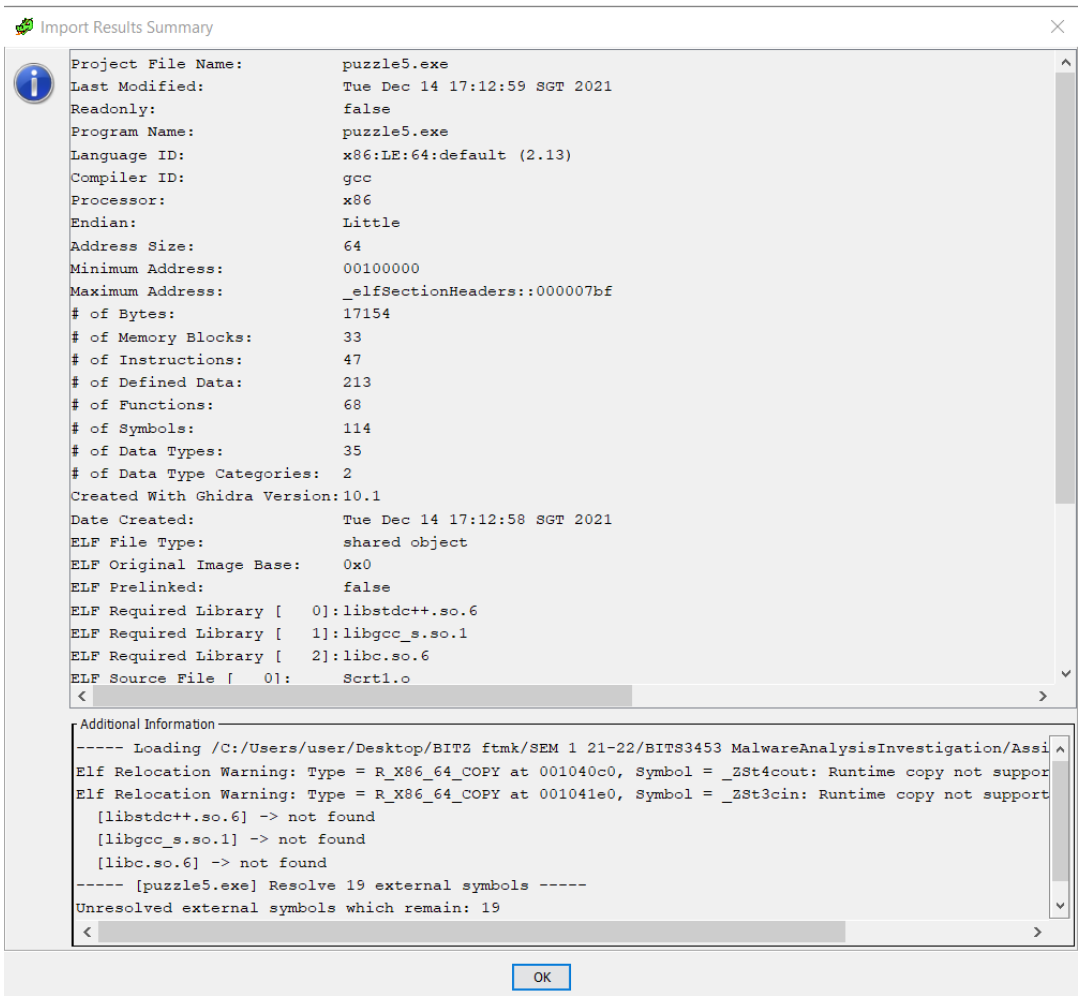
- WinDBG

  A debugger for Microsoft Windows operating system computer distributed by Microsoft.

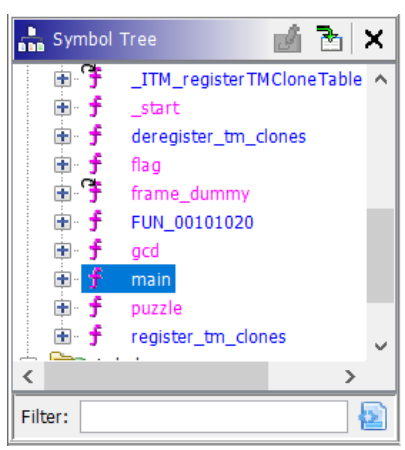
2) Step by step instructions

- Static Analysis

1. Run the Ghidra and create a new project. Import the puzzle5.exe into Ghidra and start analyzing the file. After importing the puzzle5.exe into Ghidra, it prompted us with Import Result summary which showed us some details of the imported file such as the Project File Name, Compiler ID, Last Modified date and some more.



2. Search for Main function, use the symbol tree located at the left side of the panel to ease your search. Click on it and it will direct you to the Main function in assembly language and C code.

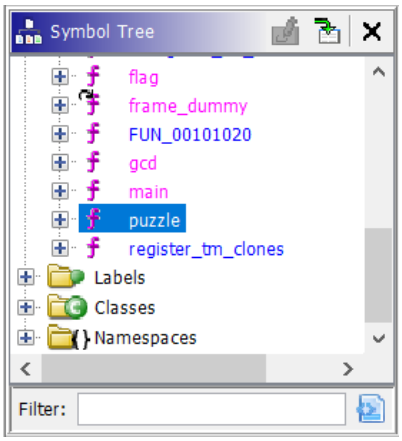


```
                    **************************************************************
                    *                                                            *
                    *  FUNCTION                                                  *
                    **************************************************************
                    undefined main()
     undefined        AL:1           <RETURN>
     undefined4       Stack[-0xc]:4  local_c                    XREF[2]:     00101a3e(*),
                                                                               00101aa3(R)
     undefined4       Stack[-0x10]:4 local_10                   XREF[2]:     00101a64(*),
                                                                               00101aa0(R)
     undefined4       Stack[-0x14]:4 local_14                   XREF[2]:     00101a8a(*),
                                                                               00101a9d(R)
                    main                                  XREF[4]:     Entry Point(*),
                                                                       _start:0010114d(*), 0010220c,
                                                                       001024ac(*)
00101a10 55          PUSH       RBP
```

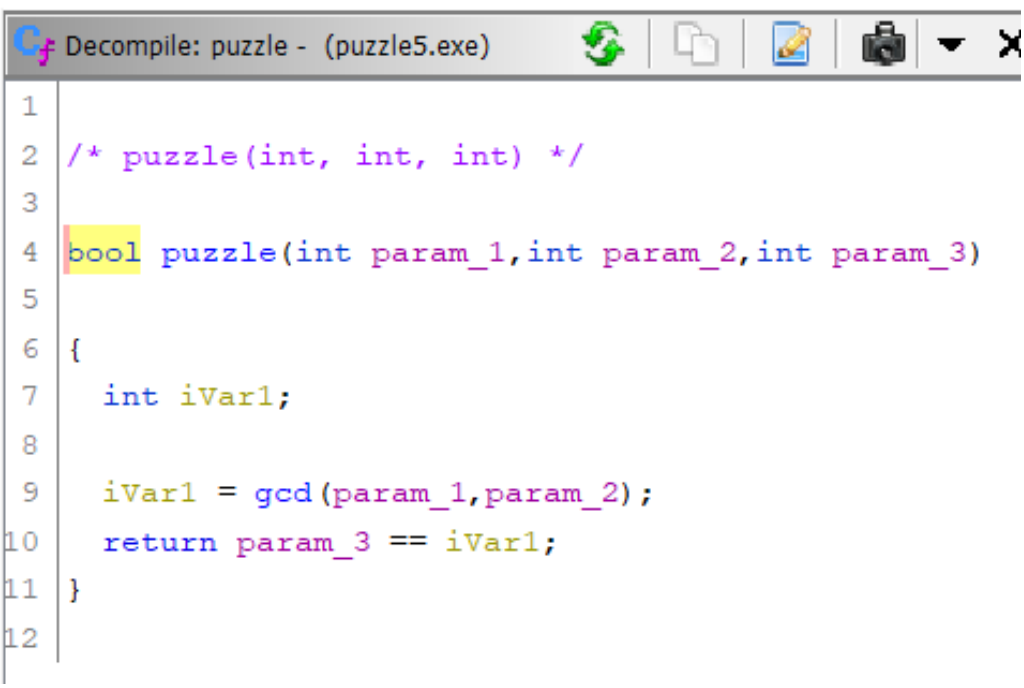
Decompile: main - (puzzle5.exe)

```
1
2  undefined8 main(void)
3
4  {
5    char cVar1;
6    int local_14;
7    int local_10;
8    int local_c;
9
10   std::operator<<((basic_ostream *)std::cout,"This is a puzzle program\n");
11   std::operator<<((basic_ostream *)std::cout,"Please enter secret number 1 : ");
12   std::basic_istream<char,std::char_traits<char>>::operator>>
13             ((basic_istream<char,std::char_traits<char>> *)std::cin,&local_c);
14   std::operator<<((basic_ostream *)std::cout,"Please enter secret number 2 : ");
15   std::basic_istream<char,std::char_traits<char>>::operator>>
16             ((basic_istream<char,std::char_traits<char>> *)std::cin,&local_10);
17   std::operator<<((basic_ostream *)std::cout,"Please enter secret number 3 : ");
18   std::basic_istream<char,std::char_traits<char>>::operator>>
19             ((basic_istream<char,std::char_traits<char>> *)std::cin,&local_14);
20   cVar1 = puzzle(local_c,local_10,local_14);
21   if (cVar1 == '\x01') {
22     std::operator<<((basic_ostream *)std::cout,"Congratulation you guest the correct number !!!!!...
       n"
23                    );
24     flag();
25   }
26   else {
27     std::operator<<((basic_ostream *)std::cout,"Please try again !!!!!\n");
28   }
29   return 0;
30 }
```

3. From the Main function we can see that it calls for a puzzle function. Go to the puzzle function, use the symbol tree again to ease your search and click on it. It will then prompt you with puzzle function in assembly language and C code.

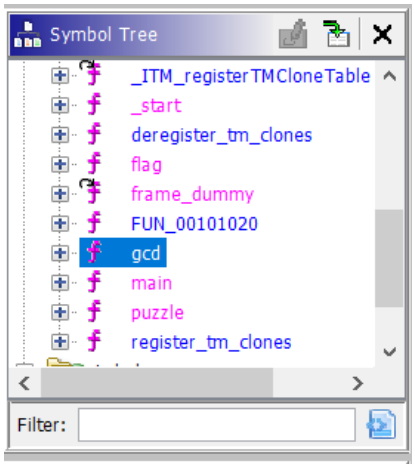



```
                    **************************************************************...
                    * puzzle(int, int, int)                                        ...
                    **************************************************************...
                    undefined __stdcall puzzle(int param_1, int param_2, in...
      undefined        AL:1          <RETURN>
      int              EDI:4         param_1
      int              ESI:4         param_2
      int              EDX:4         param_3
      undefined4       Stack[-0xc]:4 local_c                          XREF[2]:     0010124c(W),
                                                                                    00101258(R)
      undefined4       Stack[-0x10]:4local_10                         XREF[2]:     0010124f(W),
                                                                                    00101255(R)
      undefined4       Stack[-0x14]:4local_14                         XREF[2]:     00101252(W),
                                                                                    00101264(R)
                    _Z6puzzleiii                              XREF[4]:     Entry Point(*),
                    puzzle                                                 main:00101aaa(c), 001021fc,
                                                                           00102440(*)
00101244 55          PUSH       RBP
```




```
/* puzzle(int, int, int) */

bool puzzle(int param_1,int param_2,int param_3)

{
  int iVar1;

  iVar1 = gcd(param_1,param_2);
  return param_3 == iVar1;
}
```

4. From the puzzle function, we can see that it uses the gcd which is the operation for finding the greatest common divisor. It basically finding the largest integer from two integers that can exactly divide both numbers without a remainder.




```
                    **************************************************************...
                    * gcd(int, int)                                                ...
                    **************************************************************...
                    undefined __stdcall gcd(int param_1, int param_2)
     undefined          AL:1           <RETURN>
     int                EDI:4          param_1
     int                ESI:4          param_2
     undefined4         Stack[-0xc]:4  local_c                          XREF[3]:     0010121d(W),
                                                                                     00101229(R),
                                                                                     0010122e(R)
     undefined4         Stack[-0x10]:4 local_10                         XREF[4]:     00101220(W),
                                                                                     00101223(R),
                                                                                     00101232(R),
                                                                                     00101235(R)
                    _Z3gcdii                                       XREF[5]:     Entry Point(*), 0010123c(c),
                    gcd                                                         puzzle:0010125f(c), 001021f4,
                                                                                00102420(*)
00101215 55            PUSH       RBP
```

Decompile: gcd - (puzzle5.exe)

```
/* gcd(int, int) */

ulong gcd(int param_1,int param_2)

{
  ulong uVar1;

  if (param_2 == 0) {
    uVar1 = (ulong)(uint)param_1;
  }
  else {
    uVar1 = gcd(param_2,param_1 % param_2);
  }
  return uVar1;
}
```

5. From the above, we understand that this puzzle wants us to enter 3 numbers and the third numbers will be compared with the gcd of first and second numbers. The puzzle is solved if the 3rd numbers are the same as the gcd of the 1st and 2nd numbers.

- Dynamic Analysis

1. Open EDB debugger and open the puzzle5.exe program, press start once, this is the main function

```
rax  0000561b:a2986a10  55                    push rbp
     0000561b:a2986a11  48 89 e5              mov rbp, rsp
     0000561b:a2986a14  48 83 ec 10           sub rsp, 0x10
     0000561b:a2986a18  48 8d 35 83 06 00 00  lea rsi, [rel 0x561ba29870a2]     ASCII "This is a puzzle program\n"
     0000561b:a2986a1f  48 8d 3d 9a 26 00 00  lea rdi, [rel 0x561ba29890c0]
     0000561b:a2986a26  e8 65 f6 ff ff        call puzzle5.exe!std::basic_ostream<char, std:…
     0000561b:a2986a2b  48 8d 35 8e 06 00 00  lea rsi, [rel 0x561ba29870c0]     ASCII "Please enter secret number 1 : "
     0000561b:a2986a32  48 8d 3d 87 26 00 00  lea rdi, [rel 0x561ba29890c0]
     0000561b:a2986a39  e8 52 f6 ff ff        call puzzle5.exe!std::basic_ostream<char, std:…
     0000561b:a2986a3e  48 8d 45 fc           lea rax, [rbp-4]
     0000561b:a2986a42  48 89 c6              mov rsi, rax
     0000561b:a2986a45  48 8d 3d 94 27 00 00  lea rdi, [rel 0x561ba29891e0]
     0000561b:a2986a4c  e8 ff f5 ff ff        call puzzle5.exe!std::istream::operator>>(int&…
     0000561b:a2986a51  48 8d 35 88 06 00 00  lea rsi, [rel 0x561ba29870e0]     ASCII "Please enter secret number 2 : "
     0000561b:a2986a58  48 8d 3d 61 26 00 00  lea rdi, [rel 0x561ba29890c0]
     0000561b:a2986a5f  e8 2c f6 ff ff        call puzzle5.exe!std::basic_ostream<char, std:…
     0000561b:a2986a64  48 8d 45 f8           lea rax, [rbp-8]
     0000561b:a2986a68  48 89 c6              mov rsi, rax
     0000561b:a2986a6b  48 8d 3d 6e 27 00 00  lea rdi, [rel 0x561ba29891e0]
     0000561b:a2986a72  e8 d9 f5 ff ff        call puzzle5.exe!std::istream::operator>>(int&…
     0000561b:a2986a77  48 8d 35 82 06 00 00  lea rsi, [rel 0x561ba2987100]     ASCII "Please enter secret number 3 : "
     0000561b:a2986a7e  48 8d 3d 3b 26 00 00  lea rdi, [rel 0x561ba29890c0]
     0000561b:a2986a85  e8 06 f6 ff ff        call puzzle5.exe!std::basic_ostream<char, std:…
     0000561b:a2986a8a  48 8d 45 f4           lea rax, [rbp-0xc]
     0000561b:a2986a8e  48 89 c6              mov rsi, rax
     0000561b:a2986a91  48 8d 3d 48 27 00 00  lea rdi, [rel 0x561ba29891e0]
```

2. Put breakpoints on puzzle function and flag function

```
0000561b:a2986aaa  e8 95 f7 ff ff     call puzzle5.exe!puzzle(int, int, int)
0000561b:a2986aaf  0f b6 c0           movzx eax, al
0000561b:a2986ab2  83 f8 01           cmp eax, 1
0000561b:a2986ab5  0f 94 c0           sete al
0000561b:a2986ab8  84 c0              test al, al
0000561b:a2986aba  74 1a              je 0x561ba2986ad6
0000561b:a2986abc  48 8d 35 5d 06 00 00  lea rsi, [rel 0x561ba2987120]   ASCII "Congratulation you guest the correct number !!!!!\n"
0000561b:a2986ac3  48 8d 3d f6 25 00 00  lea rdi, [rel 0x561ba29890c0]
0000561b:a2986aca  e8 c1 f5 ff ff     call puzzle5.exe!std::basic_ostream<char, std:…
0000561b:a2986acf  e8 a8 f7 ff ff     call puzzle5.exe!flag()
0000561b:a2986ad4  eb 13              jmp 0x561ba2986ae9
0000561b:a2986ad6  48 8d 35 76 06 00 00  lea rsi, [rel 0x561ba2987153]   ASCII "Please try again !!!!!\n"
0000561b:a2986add  48 8d 3d dc 25 00 00  lea rdi, [rel 0x561ba29890c0]
0000561b:a2986ae4  e8 a7 f5 ff ff     call puzzle5.exe!std::basic_ostream<char, std:…
0000561b:a2986ae9  b8 00 00 00 00     mov eax, 0
0000561b:a2986aee  c9                 leave
0000561b:a2986aef  c3                 ret
0000561b:a2986af0  55                 push rbp
```

3. Continue the program as usual and input any number

```
edb output
This is a puzzle program
Please enter secret number 1 : 1
Please enter secret number 2 : 2
Please enter secret number 3 : 3
```

4. Step into puzzle function

File View Debug Plugins Options Help

```
0000561b:a2986a8a  48 8d 45 f4                lea rax, [rbp-0xc]
0000561b:a2986a8e  48 89 c6                   mov rsi, rax
0000561b:a2986a91  48 8d 3d 48 27 00 00       lea rdi, [rel 0x561ba29891e0]
0000561b:a2986a98  e8 b3 f5 ff ff             call puzzle5.exe!std::istream::operator>>(int&…
0000561b:a2986a9d  8b 55 f4                   mov edx, [rbp-0xc]
0000561b:a2986aa0  8b 4d f8                   mov ecx, [rbp-8]
0000561b:a2986aa3  8b 45 fc                   mov eax, [rbp-4]
0000561b:a2986aa6  89 ce                      mov esi, ecx
0000561b:a2986aa8  89 c7                      mov edi, eax
0000561b:a2986aaa  e8 95 f7 ff ff             call puzzle5.exe!puzzle(int, int, int)
0000561b:a2986aaf  0f b6 c0                   movzx eax, al
0000561b:a2986ab2  83 f8 01                   cmp eax, 1
0000561b:a2986ab5  0f 94 c0                   sete al
0000561b:a2986ab8  84 c0                      test al, al
0000561b:a2986aba  74 1a                      je 0x561ba2986ad6
0000561b:a2986abc  48 8d 35 5d 06 00 00       lea rsi, [rel 0x561ba2987120]
0000561b:a2986ac3  48 8d 3d f6 25 00 00       lea rdi, [rel 0x561ba29890c0]
0000561b:a2986aca  e8 c1 f5 ff ff             call puzzle5.exe!std::basic_ostream<char, std:…
0000561b:a2986acf  e8 a8 f7 ff ff             call puzzle5.exe!flag()
0000561b:a2986ad4  eb 13                      jmp 0x561ba2986ae9
```

5. This is inside puzzle function.

```
0000561b:a2986244 55                    push rbp
0000561b:a2986245 48 89 e5              mov rbp, rsp
0000561b:a2986248 48 83 ec 10           sub rsp, 0x10
0000561b:a298624c 89 7d fc              mov [rbp-4], edi
0000561b:a298624f 89 75 f8              mov [rbp-8], esi
0000561b:a2986252 89 55 f4              mov [rbp-0xc], edx
0000561b:a2986255 8b 55 f8              mov edx, [rbp-8]
0000561b:a2986258 8b 45 fc              mov eax, [rbp-4]
0000561b:a298625b 89 d6                 mov esi, edx
0000561b:a298625d 89 c7                 mov edi, eax
0000561b:a298625f e8 b1 ff ff ff        call puzzle5.exe!gcd(int, int)
0000561b:a2986264 39 45 f4              cmp [rbp-0xc], eax
0000561b:a2986267 0f 94 c0              sete al
0000561b:a298626a 84 c0                 test al, al
0000561b:a298626c 74 07                 je 0x561ba2986275
0000561b:a298626e b8 01 00 00 00        mov eax, 1
0000561b:a2986273 eb 05                 jmp 0x561ba298627a
0000561b:a2986275 b8 00 00 00 00        mov eax, 0
0000561b:a298627a c9                    leave
0000561b:a298627b c3                    ret
0000561b:a298627c 55                    push rbp
0000561b:a298627d 48 89 e5              mov rbp, rsp
0000561b:a2986280 41 55                 push r13
0000561b:a2986282 41 54                 push r12
```

6. Continue to step into instructions one by one until arrive at cmp instruction

```
0000561b:a2986264 39 45 f4              cmp [rbp-0xc], eax
0000561b:a2986267 0f 94 c0              sete al
0000561b:a298626a 84 c0                 test al, al
0000561b:a298626c 74 07                 je 0x561ba2986275
0000561b:a298626e b8 01 00 00 00        mov eax, 1
0000561b:a2986273 eb 05                 jmp 0x561ba298627a
0000561b:a2986275 b8 00 00 00 00        mov eax, 0
0000561b:a298627a c9                    leave
0000561b:a298627b c3                    ret
```

7. We can see the program is trying to make a comparison between the value rbp-0xc and eax

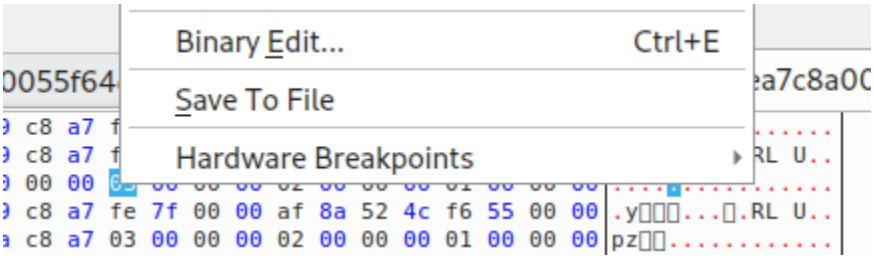
```
dword ptr [rbp - 0xc] = [0x00007ffcf2da80b4] = 0x00000003
eax = 0x00000001
```

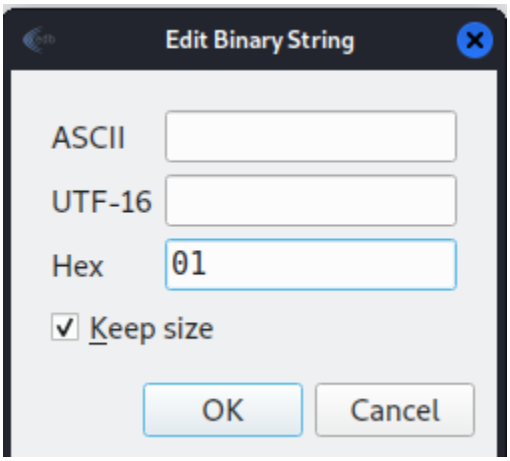
8. Follow the RBP register to registry dump

```
Registers
RBX 0000000000000000
RSP 00007ffea7c87950
RBP 00007ffea7c87960
RSI 0000000000000000
RDI 0000000000000001
R8  000000000000000a
R9  000055f64c52b1f0
R10 00007ffa44c4b5f3    ASCII "_ZNKSt
R11 0000000000000246
R12 000055f64c528130
R13 0000000000000000
R14 0000000000000000
R15 0000000000000000

RIP 000055f64c528264    <puzzle5.exe!p

C 0   ES 0000
P 1   CS 0033
A 0   SS 002b
Z 1   DS 0000
S 0   FS 0000  (00007ffa448ca140)
T 0   GS 0000  (0000000000000000)
D 0
```

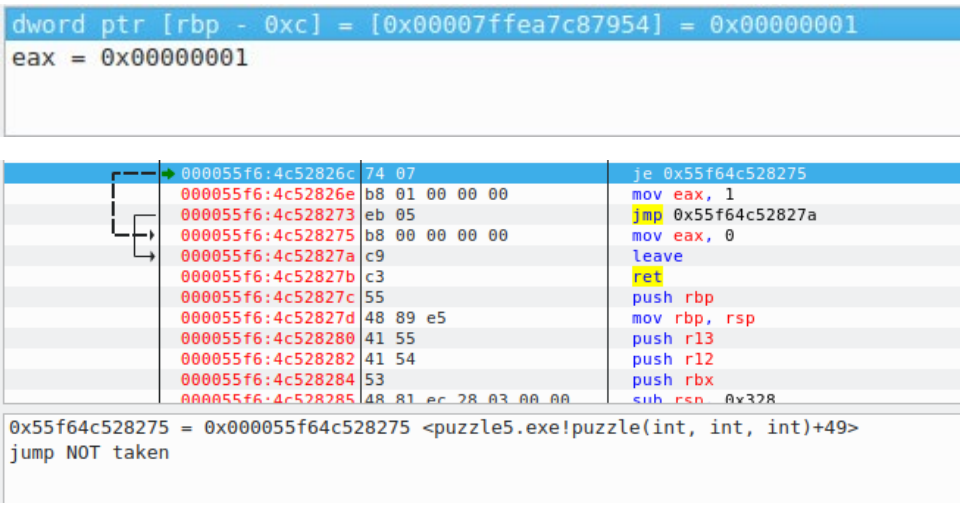
```
5f64c528000-0x000055f64c529000   0x00007ffea7c69000-0x00007ffea7c8a000

00007ffe:a7c87930 44 79 c8 a7 fe 7f 00 00 02 00 00 00 01 00 00 00 Dy□□□...........
00007ffe:a7c87940 60 79 c8 a7 fe 7f 00 00 64 82 52 4c f6 55 00 00 `y□□□...d.RL U..
00007ffe:a7c87950 00 00 00 00 03 00 00 00 02 00 00 00 01 00 00 00 ................
00007ffe:a7c87960 80 79 c8 a7 fe 7f 00 00 af 8a 52 4c f6 55 00 00 .y□□□...□.RL U..
00007ffe:a7c87970 70 7a c8 a7 03 00 00 00 02 00 00 00 01 00 00 00 pz□□............
00007ffe:a7c87980 e0 8e 52 4c f6 55 00 00 4a 7e a3 44 fa 7f 00 00 ¢.RL U..J~□D ...
00007ffe:a7c87990 78 7a c8 a7 fe 7f 00 00 bf 11 00 00 01 00 00 00 xz□□□...□.......
00007ffe:a7c879a0 10 8a 52 4c f6 55 00 00 f0 20 d6 44 fa 7f 00 00 ..RL U..  □D ...
00007ffe:a7c879b0 00 00 00 00 00 00 00 00 7b 03 0b 01 9d 1f 86 2d ........{.♪....-
00007ffe:a7c879c0 30 81 52 4c f6 55 00 00 00 00 00 00 00 00 00 00 0.RL U..........
00007ffe:a7c879d0 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 ................
00007ffe:a7c879e0 7b 03 eb ef a8 c8 97 79 7b 03 cd e0 7e 0e 9e 79 {.□□□□.y{.□¢~..y
00007ffe:a7c879f0 00 00 00 00 fa 7f 00 00 00 00 00 00 00 00 00 00 .... ...........
```

9. We can edit the value of the hex to make them equal, which is in this case hex 01





10. Now the instruction compares two same values.



11. eax receives the value 1 and returns to main function

```
000055f6:4c52826e  b8 01 00 00 00   mov eax, 1
000055f6:4c528273  eb 05            jmp 0x55f64c52827a
000055f6:4c528275  b8 00 00 00 00   mov eax, 0
000055f6:4c52827a  c9               leave
000055f6:4c52827b  c3               ret
000055f6:4c52827c  55               push rbp
000055f6:4c52827d  48 89 e5         mov rbp, rsp
000055f6:4c528280  41 55            push r13
000055f6:4c528282  41 54            push r12
000055f6:4c528284  53               push rbx
000055f6:4c528285  48 81 ec 28 03 00 00   sub rsp, 0x328

eax = 0x00000001
```

12. The program will call the flag function and the flag will be printed.

```
000055f6:4c528acf  e8 a8 f7 ff ff   call puzzle5.exe!flag()
000055f6:4c528ad4  eb 13            jmp 0x55f64c528ae9
000055f6:4c528ad6  48 8d 35 76 06 00 00   lea rsi, [rel 0x55f64c529153]      ASCII "Please try again !!!!!\n"
000055f6:4c528add  48 8d 3d dc 25 00 00   lea rdi, [rel 0x55f64c52b0c0]
000055f6:4c528ae4  e8 a7 f5 ff ff   call puzzle5.exe!std::basic_ostream<char, std:…
000055f6:4c528ae9  b8 00 00 00 00   mov eax, 0
000055f6:4c528aee  c9               leave
000055f6:4c528aef  c3               ret
000055f6:4c528af0  55               push rbp
000055f6:4c528af1  48 89 e5         mov rbp, rsp
000055f6:4c528af4  48 83 ec 10      sub rsp, 0x10
000055f6:4c528af8  89 7d fc         mov [rbp-4], edi
000055f6:4c528afb  89 75 f8         mov [rbp-8], esi
000055f6:4c528afe  83 7d fc 01      cmp dword [rbp-4], 1
000055f6:4c528b02  75 32            jne 0x55f64c528b36
000055f6:4c528b04  81 7d f8 ff ff 00 00   cmp dword [rbp-8], 0xffff
000055f6:4c528b0b  75 29            jne 0x55f64c528b36
000055f6:4c528b0d  48 8d 3d e5 27 00 00   lea rdi, [rel 0x55f64c52b2f9]
000055f6:4c528b14  e8 d7 f5 ff ff   call puzzle5.exe!std::ios_base::Init::Init()@p…
000055f6:4c528b19  48 8d 15 78 25 00 00   lea rdx, [rel 0x55f64c52b098]
000055f6:4c528b20  48 8d 35 d2 27 00 00   lea rsi, [rel 0x55f64c52b2f9]
000055f6:4c528b27  48 8b 05 ca 24 00 00   mov rax, [rel 0x55f64c52aff8]
000055f6:4c528b2e  48 89 c7         mov rdi, rax
000055f6:4c528b31  e8 4a f5 ff ff   call puzzle5.exe!__cxa_atexit@plt
000055f6:4c528b36  90               nop
000055f6:4c528b37  c9               leave
000055f6:4c528b38  c3               ret
```

edb output

```
This is a puzzle program
Please enter secret number 1 : 1
Please enter secret number 2 : 2
Please enter secret number 3 : 3
This is a puzzle program
Please enter secret number 1 : 1
Please enter secret number 2 : 2
Please enter secret number 3 : 3
Congratulation you guest the correct number !!!!!
flag_bits3453_yougotme
```

ELF ANALYSIS

1) Tools used for ELF Analysis:

- Ghidra

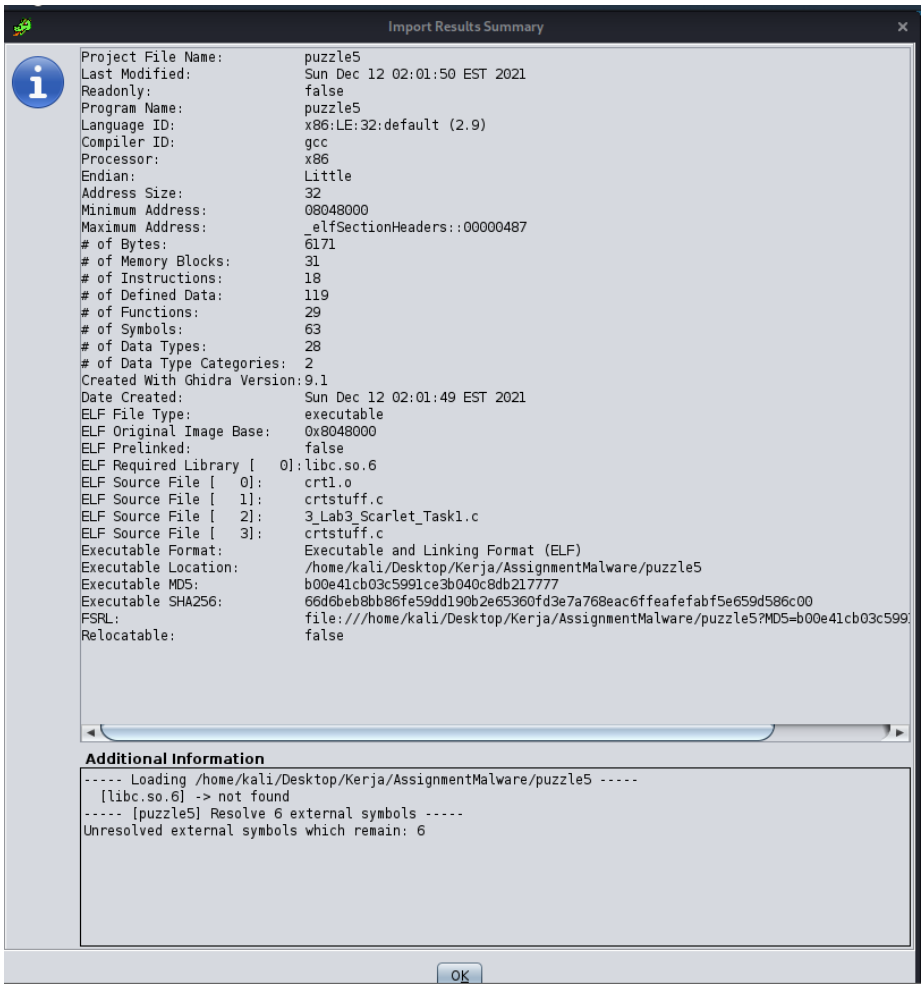
  Ghidra is a free and open-source reverse engineering tool by NSA's Research Directorate. The sources of tools were published on GitHub. Ghidra strength comes from decompiling program to C language.

- EDB Debugger

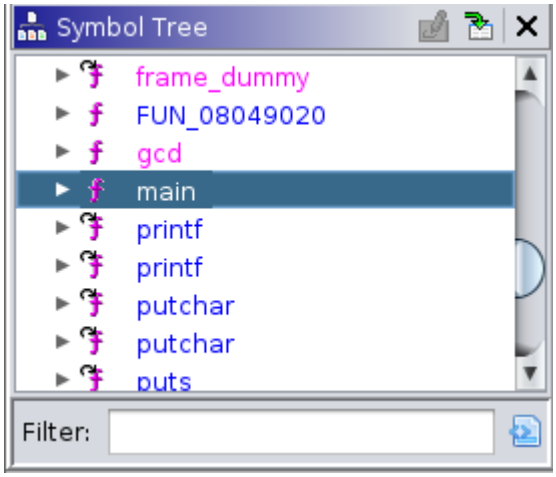
  EDB is a graphical cross platform x86/x86-64 debugger.

2) Static Analysis:

1. After importing the puzzle file into Ghidra, Ghidra will display the summary information of the file in a detailed manner. This includes the name of the file, the last time the file is modified, the address and many others. This information can be handy to make sure we choose the right file to be analyze with.



2. Find the main function of the source code of the puzzle. It should have keyword ‘main’ in it. We can detect the main function search at the Symbol Tree.



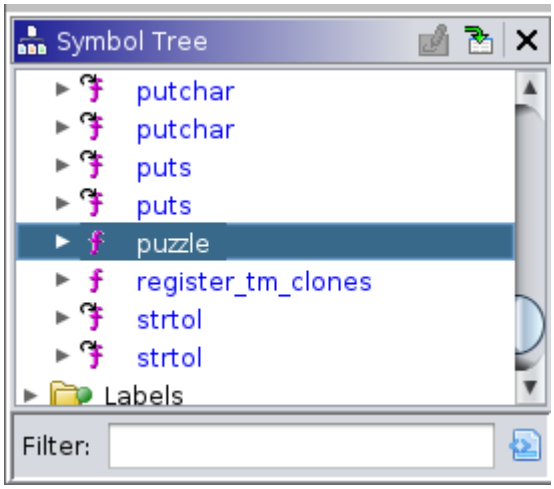
```
                    **************************************************************
                    *                          FUNCTION                          *
                    **************************************************************
                    undefined main(undefined1 param_1)
     undefined          AL:1           <RETURN>
     undefined1         Stack[0x4]:1   param_1                          XREF[1]:     080491a2(*)
     undefined4         Stack[0x0]:4   local_res0                       XREF[1]:     080491a9(R)
     undefined1         Stack[-0x10]:1 local_10                         XREF[1]:     08049294(*)
     undefined4         Stack[-0x14]:4 local_14                         XREF[2]:     080491e6(*),
                                                                                     0804924f(R)
     undefined4         Stack[-0x18]:4 local_18                         XREF[2]:     0804920e(*),
                                                                                     0804924c(R)
     undefined4         Stack[-0x1c]:4 local_1c                         XREF[2]:     08049236(*),
                                                                                     08049249(R)
                    main                                        XREF[5]:     Entry Point(*),
                                                                             _start:080490b6(*),
                                                                             _start:080490bc(*), 0804a1a0,
                                                                             0804a264(*)
080491a2 8d 4c 24 04     LEA        ECX=>param_1,[ESP + 0x4]
```

3. After decompile the source code into C language, we can see the main function clearly understandable and much more easy to read.

Decompile: main - (puzzle5)

```
/* WARNING: Function: __x86.get_pc_thunk.bx replaced with injection: g

undefined4 main(void)

{
  char cVar1;
  undefined4 local_1c;
  undefined4 local_18;
  undefined4 local_14;
  undefined *local_10;

  local_10 = &stack0x00000004;
  puts("This is a puzzle program");
  printf("Please enter secret number 1: ");
  __isoc99_scanf(&DAT_0804a0db,&local_14);
  printf("Please enter secret number 2: ");
  __isoc99_scanf(&DAT_0804a0db,&local_18);
  printf("Please enter secret number 3: ");
  __isoc99_scanf(&DAT_0804a0db,&local_1c);
  cVar1 = puzzle(local_14,local_18,local_1c);
  if (cVar1 == '\0') {
    puts("Please try again !!!!!");
  }
  else {
    puts("Congratulation you guest the correct number !!!!!");
    flag();
  }
  return 0;
}
```

4. From the Main function, it redirects us to puzzle function source code. We can also search up the puzzle function from the Symbol Tree. The puzzle function acts as backbone of the source code and the pattern of the puzzle can be found in this puzzle function.




```
                    ************************************************************
                    *                          FUNCTION                        *
                    ************************************************************
                    undefined puzzle(undefined4 param_1, undefined4 param_2,...
     undefined          AL:1           <RETURN>
     undefined4         Stack[0x4]:4   param_1                          XREF[1]:     080492e9(R)
     undefined4         Stack[0x8]:4   param_2                          XREF[1]:     080492e6(R)
     undefined4         Stack[0xc]:4   param_3                          XREF[1]:     080492f4(R)
                    puzzle                                        XREF[4]:     Entry Point(*), main:08049258(c),
                                                                               0804a1b0, 0804a2b8(*)
080492d3 55             PUSH        EBP
```

5. After decompiling the puzzle function, the C language version of this function is displayed which is more understandable. From the C language, it crystal clear display the operation happens inside the puzzle function which is GCD which is stands for Great Common Divisor. The pattern reveals that the puzzle requires us two input three numbers and this function will find GCD of first and second number then compare the result with the third number. If the number result of GCD same as the third number, the puzzle will return true and the puzzle is solved.

Decompile: puzzle - (puzzle5)

```
uint puzzle(undefined4 param_1,undefined4 param_2,int param_3)

{
  int iVar1;

  __x86.get_pc_thunk.ax();
  iVar1 = gcd(param_1,param_2);
  return (uint)(param_3 == iVar1);
}
```

6. For example, if we input three numbers which is 4, 8 and 4 respectively, the puzzle will find the GCD of 4 and 8. The answer is 4. This answer than compared to the third number which is 4 as well and automatically it will result in correct numbers as the pattern described in the puzzle function of the source code.

```
┌──(kali㉿kali)-[~/Desktop/Kerja/AssignmentMalware]
└─$ ./puzzle5
This is a puzzle program
Please enter secret number 1: 4
Please enter secret number 2: 8
Please enter secret number 3: 4
Congratulation you guest the correct number !!!!!

 your flag is =flag_bits3453_yougotme
```

3) Dynamic Analysis:

1. Open EDB debugger and open the puzzle5 program, press start once

```
0804:91db e8 50 fe ff ff      call puzzle5!printf@plt
0804:91e0 83 c4 10            add esp, 0x10
0804:91e3 83 ec 08            sub esp, 8
0804:91e6 8d 45 f4            lea eax, [ebp-0xc]
0804:91e9 50                  push eax
0804:91ea 8d 83 db e0 ff ff   lea eax, [ebx-0x1f25]
0804:91f0 50                  push eax
0804:91f1 e8 7a fe ff ff      call puzzle5!__isoc99_scanf@plt
0804:91f6 83 c4 10            add esp, 0x10
0804:91f9 83 ec 0c            sub esp, 0xc
0804:91fc 8d 83 e0 e0 ff ff   lea eax, [ebx-0x1f20]
0804:9202 50                  push eax
0804:9203 e8 28 fe ff ff      call puzzle5!printf@plt
0804:9208 83 c4 10            add esp, 0x10
0804:920b 83 ec 08            sub esp, 8
0804:920e 8d 45 f0            lea eax, [ebp-0x10]
0804:9211 50                  push eax
0804:9212 8d 83 db e0 ff ff   lea eax, [ebx-0x1f25]
0804:9218 50                  push eax
0804:9219 e8 52 fe ff ff      call puzzle5!__isoc99_scanf@plt
0804:921e 83 c4 10            add esp, 0x10
0804:9221 83 ec 0c            sub esp, 0xc
0804:9224 8d 83 00 e1 ff ff   lea eax, [ebx-0x1f00]
0804:922a 50                  push eax
0804:922b e8 00 fe ff ff      call puzzle5!printf@plt
0804:9230 83 c4 10            add esp, 0x10
```

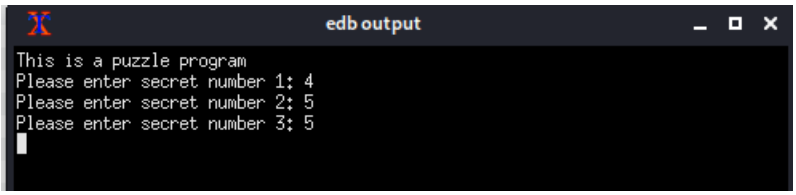
Instructions in main function, containing 3 scan calls

2. Toggle breakpoints on points of interests, such as puzzle function and flag function.

```
● 0804:9258 e8 76 00 00 00      call puzzle5!puzzle
  0804:925d 83 c4 10            add esp, 0x10
● 0804:9260 84 c0               test al, al
  0804:9262 74 19               je 0x804927d
  0804:9264 83 ec 0c            sub esp, 0xc
  0804:9267 8d 83 20 e1 ff ff   lea eax, [ebx-0x1ee0]
  0804:926d 50                  push eax
  0804:926e e8 cd fd ff ff      call puzzle5!puts@plt
  0804:9273 83 c4 10            add esp, 0x10
● 0804:9276 e8 8c 00 00 00      call puzzle5!flag
```

Breakpoints toggled for functions

3. Enter number input in terminal to proceed with the program.



4. Step into puzzle function. Continue stepping into instructions one by one and keep note of registry values, mainly ebp, esp, and eax.

```
0804:92c9 e8 d0 ff ff ff    call puzzle5!gcd
0804:92ce 83 c4 10          add esp, 0x10
0804:92d1 c9                leave
0804:92d2 c3                ret
0804:92d3 55                push ebp
0804:92d4 89 e5             mov ebp, esp
0804:92d6 83 ec 08          sub esp, 8
0804:92d9 e8 ba 00 00 00    call puzzle5!__x86.get_pc_thunk.ax
0804:92de 05 22 2d 00 00    add eax, 0x2d22
0804:92e3 83 ec 08          sub esp, 8
0804:92e6 ff 75 0c          push dword [ebp+0xc]
0804:92e9 ff 75 08          push dword [ebp+8]
0804:92ec e8 ad ff ff ff    call puzzle5!gcd
0804:92f1 83 c4 10          add esp, 0x10
0804:92f4 39 45 10          cmp [ebp+0x10], eax
0804:92f7 75 07             jne 0x8049300
0804:92f9 b8 01 00 00 00    mov eax, 1
0804:92fe eb 05             jmp 0x8049305
0804:9300 b8 00 00 00 00    mov eax, 0
0804:9305 c9                leave
```

5. Note that the program calls gcd function then made a comparison between two values. gcd function takes two values and calculate the biggest divisor of the two.

```
0804:92ec e8 ad ff ff ff    call puzzle5!gcd
0804:92f1 83 c4 10          add esp, 0x10
0804:92f4 39 45 10          cmp [ebp+0x10], eax
0804:92f7 75 07             jne 0x8049300
0804:92f9 b8 01 00 00 00    mov eax, 1
0804:92fe eb 05             jmp 0x8049305
0804:9300 b8 00 00 00 00    mov eax, 0
0804:9305 c9                leave
```

The program calling gcd function

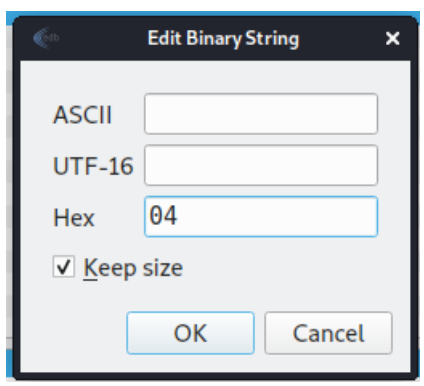
6. Here, the program is trying to compare the value of ebp+0x10 (our biggest divisor) and eax (the third value), if they are the same, return 1 and if not, return 0.

```
0804:92f4 39 45 10          cmp [ebp+0x10], eax
0804:92f7 75 07             jne 0x8049300
0804:92f9 b8 01 00 00 00    mov eax, 1
0804:92fe eb 05             jmp 0x8049305
0804:9300 b8 00 00 00 00    mov eax, 0
0804:9305 c9                leave
0804:9306 c3                ret
0804:9307 55                push ebp
0804:9308 89 e5             mov ebp, esp
0804:930a 57                push edi
0804:930b 56                push esi
0804:930c 53                push ebx
0804:930d 83 ec 6c          sub esp, 0x6c
0804:9310 e8 cb fd ff ff    call puzzle5!__x86.get_pc_thunk.bx
0804:9315 81 c3 eb 2c 00 00 add ebx, 0x2ceb

dword ptr [ebp + 0x10] = [0xffb060d8] = 0x00000005
eax = 0x00000004
possible jump from 0x080492ea
```

7. We can follow the ebp+0x10 value on the registry dump and see the value here

```
0x08049000-0x0804a000   0xffae6000-0xffb07000
ffb0:60c8 f8 60 b0 ff 5d 92 04 08 04 00 00 00 04 00 00 00
ffb0:60d8 05 00 00 00 b9 91 04 08 01 00 00 00 05 00 00 00
ffb0:60e8 04 00 00 00 04 00 00 00 10 61 b0 ff 00 00 00 00
ffb0:60f8 00 00 00 00 d6 9f db f7 00 50 f8 f7 00 50 f8 f7
ffb0:6108 00 00 00 00 d6 9f db f7 01 00 00 00 b4 61 b0 ff
```

8. Edit the binary string to 04 to match with eax



```
0804:92f4 39 45 10          cmp [ebp+0x10], eax
0804:92f7 75 07             jne 0x8049300
0804:92f9 b8 01 00 00 00    mov eax, 1
0804:92fe eb 05             jmp 0x8049305
0804:9300 b8 00 00 00 00    mov eax, 0
0804:9305 c9                leave
0804:9306 c3                ret
0804:9307 55                push ebp
0804:9308 89 e5             mov ebp, esp
0804:930a 57                push edi
0804:930b 56                push esi
0804:930c 53                push ebx
0804:930d 83 ec 6c          sub esp, 0x6c
0804:9310 e8 cb fd ff ff    call puzzle5!__x86.get_pc_thunk.bx
0804:9315 81 c3 eb 2c 00 00 add ebx, 0x2ceb

dword ptr [ebp + 0x10] = [0xffb060d8] = 0x00000004
eax = 0x00000004
possible jump from 0x080492ea
```

ebp+0x10 and eax now has the same dword value

9. Here the program will not jump, eax will receive the value 1 and returned to main.

```
0804:92f4 39 45 10          cmp [ebp+0x10], eax
0804:92f7 75 07             jne 0x8049300
0804:92f9 b8 01 00 00 00    mov eax, 1
0804:92fe eb 05             jmp 0x8049305
0804:9300 b8 00 00 00 00    mov eax, 0
0804:9305 c9                leave
0804:9306 c3                ret
0804:9307 55                push ebp
0804:9308 89 e5             mov ebp, esp
0804:930a 57                push edi
0804:930b 56                push esi
0804:930c 53                push ebx
0804:930d 83 ec 6c          sub esp, 0x6c
0804:9310 e8 cb fd ff ff    call puzzle5!__x86.get_pc_thunk.
0804:9315 81 c3 eb 2c 00 00 add ebx, 0x2ceb

0x8049300 = 0x08049300 <puzzle5!puzzle+45>
jump NOT taken
```

10. The problem will then call the flag function and print out the flag.

```
[esp]  0804:9276 e8 8c 00 00 00      call puzzle5!flag
       0804:927b eb 12               jmp 0x804928f
       0804:927d 83 ec 0c            sub esp, 0xc
       0804:9280 8d 83 52 e1 ff ff   lea eax, [ebx-0x1eae]
       0804:9286 50                  push eax
       0804:9287 e8 b4 fd ff ff      call puzzle5!puts@plt
       0804:928c 83 c4 10            add esp, 0x10
       0804:928f b8 00 00 00 00      mov eax, 0
       0804:9294 8d 65 f8            lea esp, [ebp-8]
       0804:9297 59                  pop ecx
       0804:9298 5b                  pop ebx
       0804:9299 5d                  pop ebp
       0804:929a 8d 61 fc            lea esp, [ecx-4]
       0804:929d c3                  ret
       0804:929e 55                  push ebp
       0804:929f 89 e5               mov ebp, esp
       0804:92a1 83 ec 08            sub esp, 8
       0804:92a4 e8 ef 00 00 00      call puzzle5!__x86.get_pc_thunk.ax
       0804:92a9 05 57 2d 00 00      add eax, 0x2d57
       0804:92ae 83 7d 0c 00         cmp dword [ebp+0xc], 0
       0804:92b2 75 05               jne 0x80492b9
       0804:92b4 8b 45 08            mov eax, [ebp+8]
       0804:92b7 eb 18               jmp 0x80492d1
       0804:92b9 8b 45 08            mov eax, [ebp+8]
       0804:92bc 99                  cdq
       0804:92bd f7 7d 0c            idiv dword [ebp+0xc]
       0804:92c0 89 d0               mov eax, edx
```

```
This is a puzzle program
Please enter secret number 1: 4
Please enter secret number 2: 4
Please enter secret number 3: 5
Congratulation you guest the correct number !!!!!

 your flag is =flag_bits3453_yougotme
```

The flag function printed out the flag

CONCLUSION

In closing of curtain of this discussion, analysis skills are extremely important. It does not matter which type of analysis we excel in whether it is static analysis or dynamic analysis. Both can be used to analyze any source code to detect any malicious code proven from this discussion report. From this report, we can see both analysis types will result in successful solutions for both puzzles. It doesn't matter what kind of puzzle we need to solve, if we have a basic skill of analyzing and excel in either one of the types of analysis, we can solve any type of puzzle provided. Not only that, but basic programming skills are also needed in order to understand the source code plus with the decompiled version of the code. This clearly states that analyzing a file is harder than it seems. It requires a lot of technical skills such as programming, analyzing skills, problem solving skills and many more. This is requiring more if we are in a more professional business world which involves big companies as one small mistake can result in a big failure of the company. Therefore, every layer of IT individuals needs to pay attention to the world of analyzation, and it's recommended for each one of these individuals to have basic skills of analyzation in order to produce a well-verse of IT society for the future.