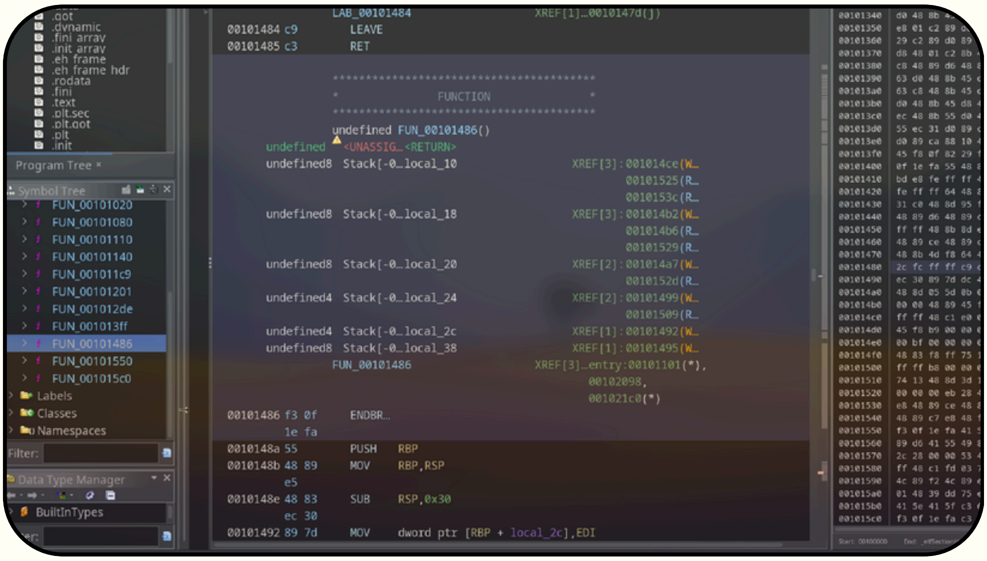
Importing “chal1” File To Ghidra

After importing the “chal1” file into Ghidra and it turns the machine code back into readable form i started to search for entry point and i found an interesting function called FUN_00101486 showing in Screenshot 1.



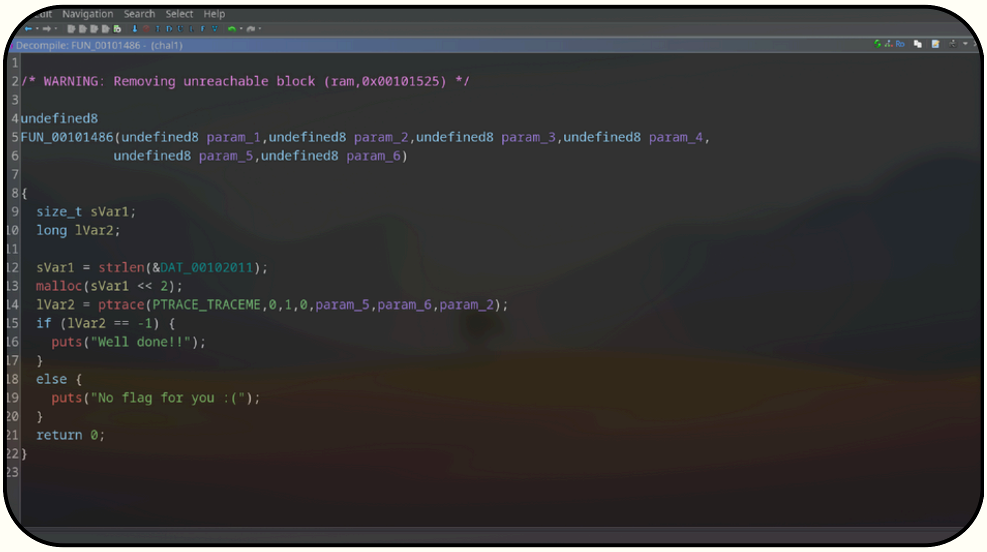
```
                         LAB_00101484                    XREF[1]…0010147d(j)
00101484 c9          LEAVE
00101485 c3          RET

                     **************************************
                     *               FUNCTION             *
                     **************************************
                     undefined FUN_00101486()
     undefined         <UNASSIG…<RETURN>
     undefined8   Stack[-0…local_10                XREF[3]:001014ce(W…
                                                           00101525(R…
                                                           0010153c(R…
     undefined8   Stack[-0…local_18                XREF[3]:001014b2(W…
                                                           001014b6(R…
                                                           00101529(R…
     undefined8   Stack[-0…local_20                XREF[2]:001014a7(W…
                                                           0010152d(R…
     undefined4   Stack[-0…local_24                XREF[2]:00101499(W…
                                                           00101509(R…
     undefined4   Stack[-0…local_2c                XREF[1]:00101492(W…
     undefined8   Stack[-0…local_38                XREF[1]:00101495(W…
                     FUN_00101486                  XREF[3]…entry:00101101(*),
                                                        00102098,
                                                        001021c0(*)
00101486 f3 0f       ENDBR…
         1e fa
0010148a 55          PUSH    RBP
0010148b 48 89       MOV     RBP,RSP
         e5
0010148e 48 83       SUB     RSP,0x30
         ec 30
00101492 89 7d       MOV     dword ptr [RBP + local_2c],EDI
```

Screenshot 1

Decompiled The Function

Using the Ghidra Decompiler, the assembly instructions were translated into C-like pseudocode for better readability as we can see in Screenshot 2.


Edit Navigation Search Select Help
Decompile: FUN_00101486 - (chal1)

```
/* WARNING: Removing unreachable block (ram,0x00101525) */

undefined8
FUN_00101486(undefined8 param_1,undefined8 param_2,undefined8 param_3,undefined8 param_4,
            undefined8 param_5,undefined8 param_6)

{
  size_t sVar1;
  long lVar2;

  sVar1 = strlen(&DAT_00102011);
  malloc(sVar1 << 2);
  lVar2 = ptrace(PTRACE_TRACEME,0,1,0,param_5,param_6,param_2);
  if (lVar2 == -1) {
    puts("Well done!!");
  }
  else {
    puts("No flag for you :(");
  }
  return 0;
}
```

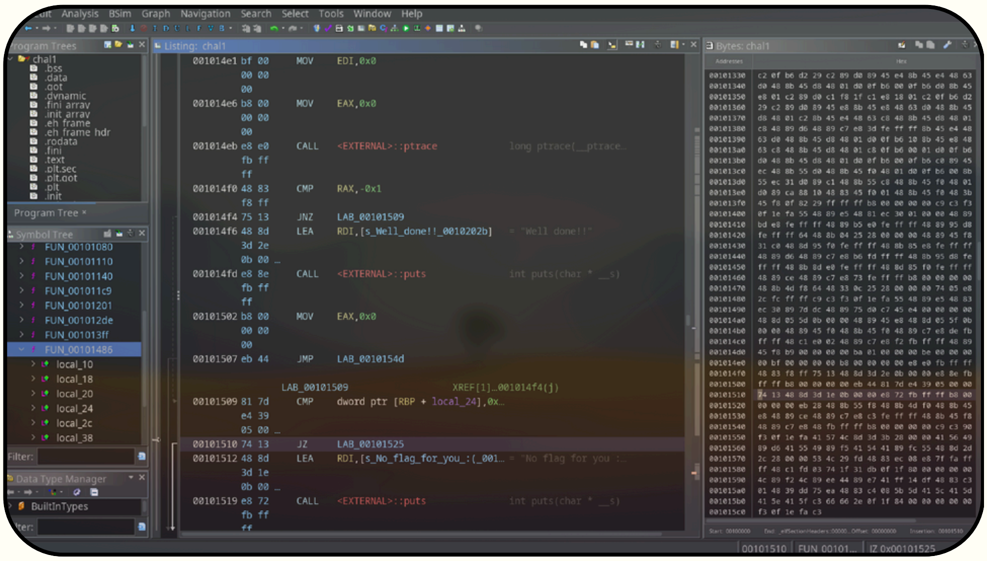
Screenshot 2

Inside the function, an if condition was observed that decides which message the program prints. The two possible outputs were:

- **"Well done!!"**
- **"No flag for you :("**

This indicated that the program was intentionally hiding a success condition.

so by going back to Screenshot 1 back to assembly code and Investigating the Conditional as showing in Screenshot 3



```
001014e1 bf 00      MOV     EDI,0x0
         00 00
         00
001014e6 b8 00      MOV     EAX,0x0
         00 00
         00
001014eb e8 e0      CALL    <EXTERNAL>::ptrace                  long ptrace(__ptrace…
         fb ff
         ff
001014f0 48 83      CMP     RAX,-0x1
         f8 ff
001014f4 75 13      JNZ     LAB_00101509
001014f6 48 8d      LEA     RDI,[s_Well_done!!_0010202b]       = "Well done!!"
         3d 2e
         0b 00 …
001014fd e8 8e      CALL    <EXTERNAL>::puts                    int puts(char * __s)
         fb ff
         ff
00101502 b8 00      MOV     EAX,0x0
         00 00
         00
00101507 eb 44      JMP     LAB_0010154d

             LAB_00101509                          XREF[1]…001014f4(j)
00101509 81 7d      CMP     dword ptr [RBP + local_24],0x…
         e4 39
         05 00 …
00101510 74 13      JZ      LAB_00101525
00101512 48 8d      LEA     RDI,[s_No_flag_for_you_:(_001… = "No flag for you :…
         3d 1e
         0b 00 …
00101519 e8 72      CALL    <EXTERNAL>::puts                    int puts(char * __s)
         fb ff
         ff
```

Screenshot 3

Upon examining the assembly code, the following logic was identified:

CMP RAX, -1

JNZ LAB_00101509

This comparison checks the return value of a system **call (ptrace)**. If the return value is **-1**, the program assumes it is being debugged and immediately prints **"Well done!!"** as a decoy.

If the check fails, execution continues to another comparison:

CMP [RBP + local_24], 0x539

JZ LAB_00101525

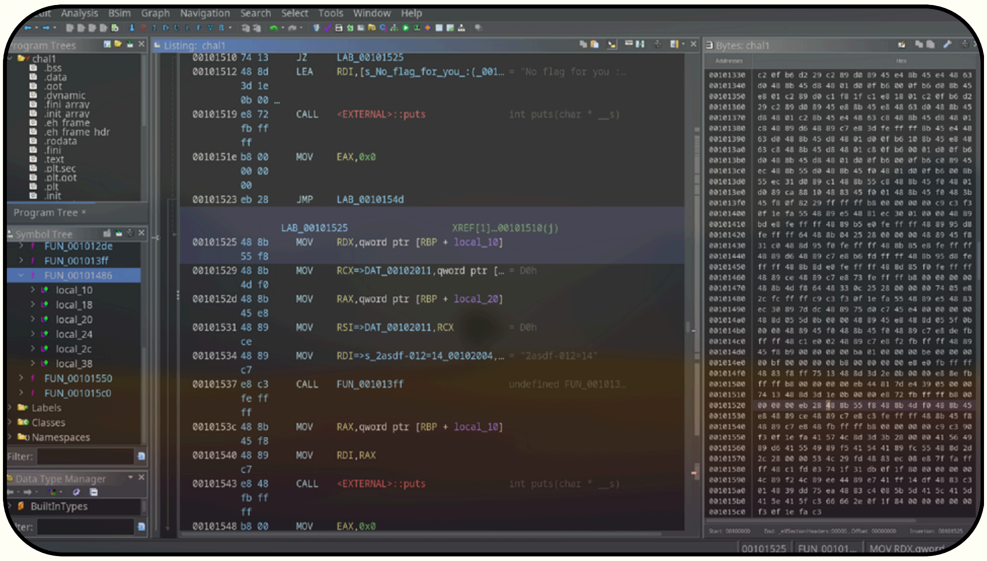
However, during analysis it was discovered that **local_24** is never modified anywhere in the program. This means the condition can never evaluate to true during normal execution.

As a result, the program always prints:

"No flag for you :("

Further inspection showed that the real flag is only printed if execution reaches a hidden code block located at:

LAB_00101525 as showing in Screenshot 4



```
00101510 74 13      JZ      LAB_00101525
00101512 48 8d      LEA     RDI,[s_No_flag_for_you_:(_001… = "No flag for you :…
         3d 1e
         0b 00 …
00101519 e8 72      CALL    <EXTERNAL>::puts                    int puts(char * __s)
         fb ff
         ff
0010151e b8 00      MOV     EAX,0x0
         00 00
         00
00101523 eb 28      JMP     LAB_0010154d

             LAB_00101525                          XREF[1]…00101510(j)
00101525 48 8b      MOV     RDX,qword ptr [RBP + local_10]
         55 f8
00101529 48 8b      MOV     RCX=>DAT_00102011,qword ptr [… = D0h
         4d f0
0010152d 48 8b      MOV     RAX,qword ptr [RBP + local_20]
         45 e8
00101531 48 89      MOV     RSI=>DAT_00102011,RCX              = D0h
         ce
00101534 48 89      MOV     RDI=>s_2asdf-012=14_00102004,… = "2asdf-012=14"
         c7
00101537 e8 c3      CALL    FUN_001013ff                        undefined FUN_001013…
         fe ff
         ff
0010153c 48 8b      MOV     RAX,qword ptr [RBP + local_10]
         45 f8
00101540 48 89      MOV     RDI,RAX
         c7
00101543 e8 48      CALL    <EXTERNAL>::puts                    int puts(char * __s)
         fb ff
         ff
00101548 b8 00      MOV     EAX,0x0
```

Screenshot 4

This block calls a function that decrypts and prints the actual flag string. However, due to the unreachable conditional check, this code path is never reached during normal execution.

Bypassing The Condition

So to access the real flag, the conditional jump instruction was patched.

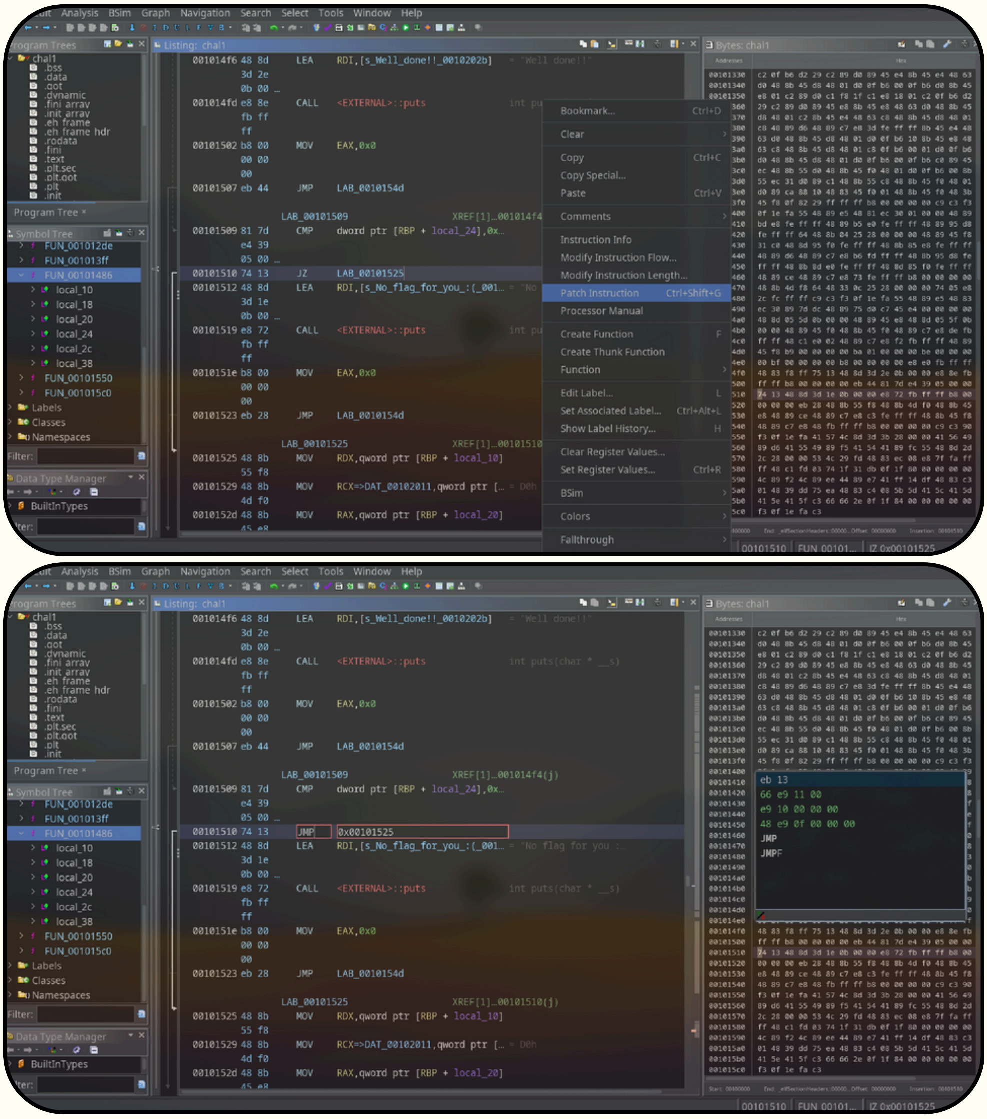
Original instruction:

JZ LAB_00101525

Patched instruction:

JMP LAB_00101525

This forces the program to always jump to the success branch, bypassing the failing condition. As we can see in the Screenshot 5



```
Listing: chal1
001014f6 48 8d       LEA     RDI,[s_Well_done!!_0010202b]          = "Well done!!"
         3d 2e
         0b 00 …
001014fd e8 8e       CALL    <EXTERNAL>::puts                      int pu
         fb ff
         ff
00101502 b8 00       MOV     EAX,0x0
         00 00
         00
00101507 eb 44       JMP     LAB_0010154d

                 LAB_00101509                     XREF[1]…001014f4
00101509 81 7d       CMP     dword ptr [RBP + local_24],0x…
         e4 39
         05 00 …
00101510 74 13       JZ      LAB_00101525
00101512 48 8d       LEA     RDI,[s_No_flag_for_you_:(_001…  = "No
         3d 1e
         0b 00 …
00101519 e8 72       CALL    <EXTERNAL>::puts                      int pu
         fb ff
         ff
0010151e b8 00       MOV     EAX,0x0
         00 00
         00
00101523 eb 28       JMP     LAB_0010154d

                 LAB_00101525                     XREF[1]…00101510
00101525 48 8b       MOV     RDX,qword ptr [RBP + local_10]
         55 f8
00101529 48 8b       MOV     RCX=>DAT_00102011,qword ptr [… = D0h
         4d f0
0010152d 48 8b       MOV     RAX,qword ptr [RBP + local_20]
         45 e8
```

```
Listing: chal1
001014f6 48 8d       LEA     RDI,[s_Well_done!!_0010202b]          = "Well done!!"
         3d 2e
         0b 00 …
001014fd e8 8e       CALL    <EXTERNAL>::puts                      int puts(char * __s)
         fb ff
         ff
00101502 b8 00       MOV     EAX,0x0
         00 00
         00
00101507 eb 44       JMP     LAB_0010154d

                 LAB_00101509                     XREF[1]…001014f4(j)
00101509 81 7d       CMP     dword ptr [RBP + local_24],0x…
         e4 39
         05 00 …
00101510 74 13       JMP     0x00101525
00101512 48 8d       LEA     RDI,[s_No_flag_for_you_:(_001…  = "No flag for you :…
         3d 1e
         0b 00 …
00101519 e8 72       CALL    <EXTERNAL>::puts                      int puts(char * __s)
         fb ff
         ff
0010151e b8 00       MOV     EAX,0x0
         00 00
         00
00101523 eb 28       JMP     LAB_0010154d

                 LAB_00101525                     XREF[1]…00101510(j)
00101525 48 8b       MOV     RDX,qword ptr [RBP + local_10]
         55 f8
00101529 48 8b       MOV     RCX=>DAT_00102011,qword ptr [… = D0h
         4d f0
0010152d 48 8b       MOV     RAX,qword ptr [RBP + local_20]
         45 e8
```

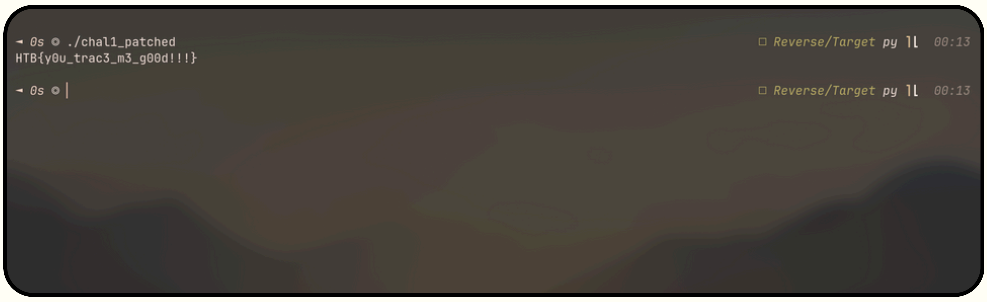
```
eb 13
66 e9 11 00
e9 10 00 00 00
48 e9 0f 00 00 00
JMP
JMPF
```

Screenshot 5

Final Result

After patching and executing the binary, the program printed the correct flag:

HTB{y0u_trac3_m3_g00d!!!} As showing in Screenshot 6



```
➜ 0s ○ ./chal1_patched                                    □ Reverse/Target py ⅼⅼ 00:13
HTB{y0u_trac3_m3_g00d!!!}

➜ 0s ○ |                                                  □ Reverse/Target py ⅼⅼ 00:13
```

Screenshot 6