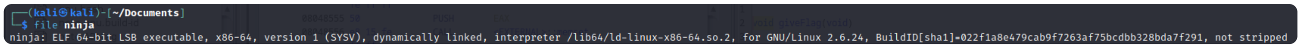
Challenge Description:

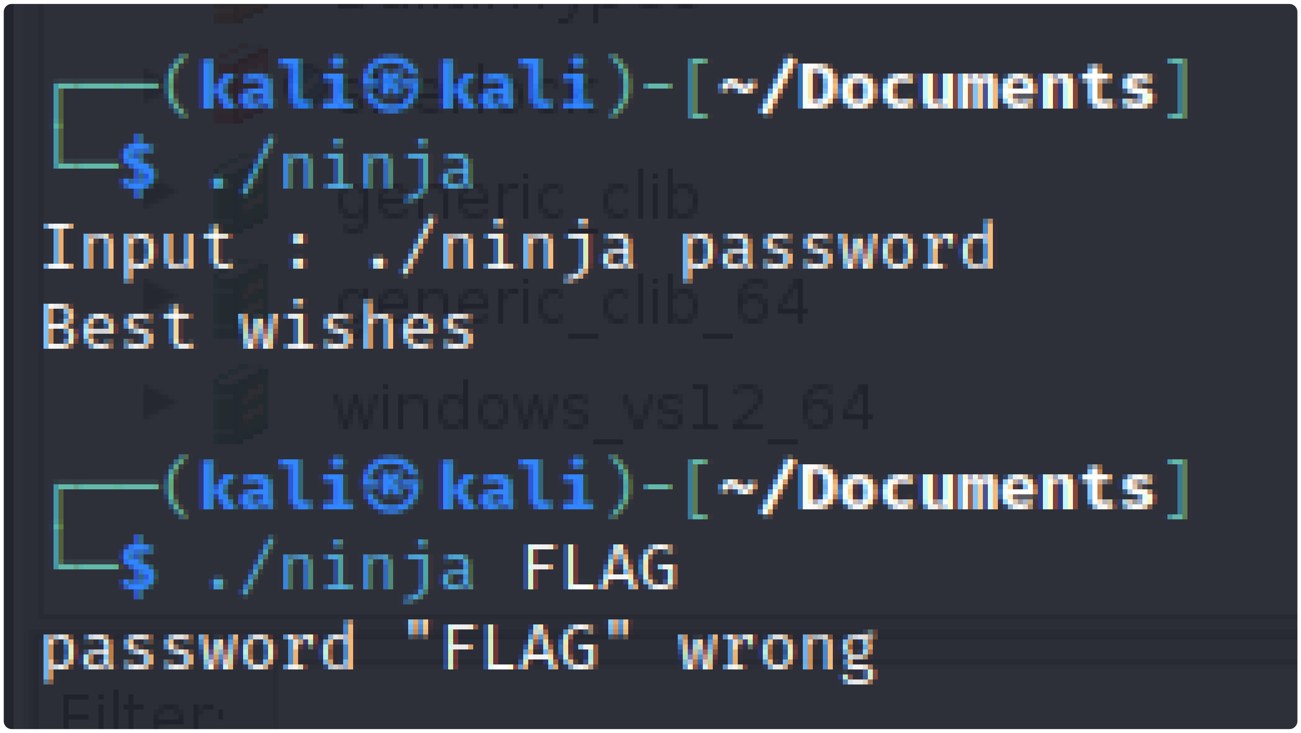
Navigate through a complex digital terrain, employing reverse engineering techniques to unveil a hidden secret buried within its labyrinthine depths.

Analysing the File :



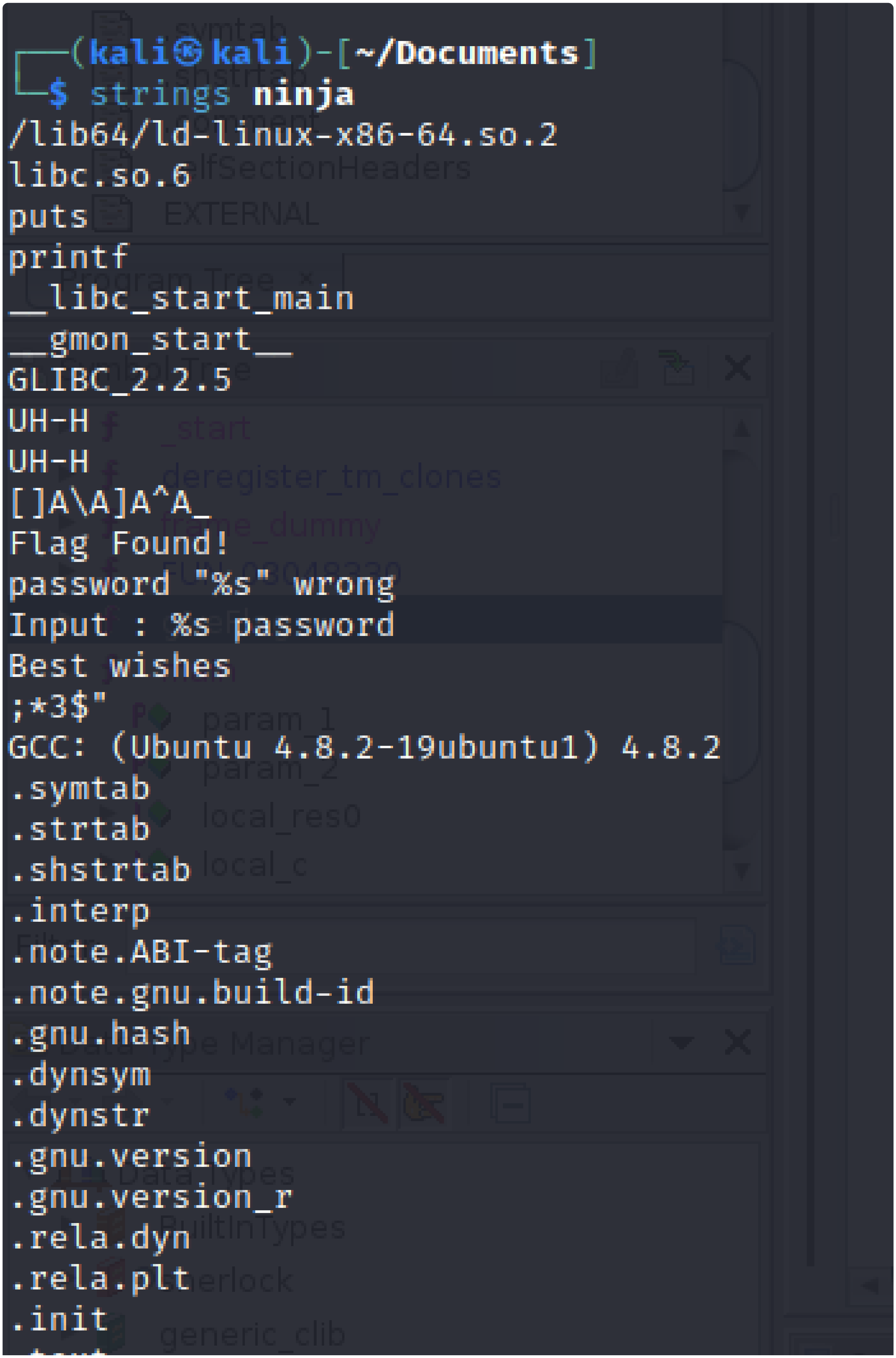
```
┌──(kali㉿kali)-[~/Documents]
└─$ file ninja
ninja: ELF 64-bit LSB executable, x86-64, version 1 (SYSV), dynamically linked, interpreter /lib64/ld-linux-x86-64.so.2, for GNU/Linux 2.6.24, BuildID[sha1]=022f1a8e479cab9f7263af75bcdbb328bda7f291, not stripped
```

ELF stands for Executable and Linkable Format. It is a common file format for executable files, object code, shared libraries, and core dumps. ELF files are used on Linux and other Unix-based systems.
The ELF format is versatile and can be executed on various processor types. It supports big-endian, little-endian, 32-bit, and 64-bit architectures systems and different CPUs.
The ELF format has several capabilities, including dynamic linking, dynamic loading, imposing run-time control on a program, and an improved method for creating shared libraries.The ELF format is the standard binary format on operating systems such as Linux.



```
┌──(kali㉿kali)-[~/Documents]
└─$ ./ninja
Input : ./ninja password
Best wishes

┌──(kali㉿kali)-[~/Documents]
└─$ ./ninja FLAG
password "FLAG" wrong
```

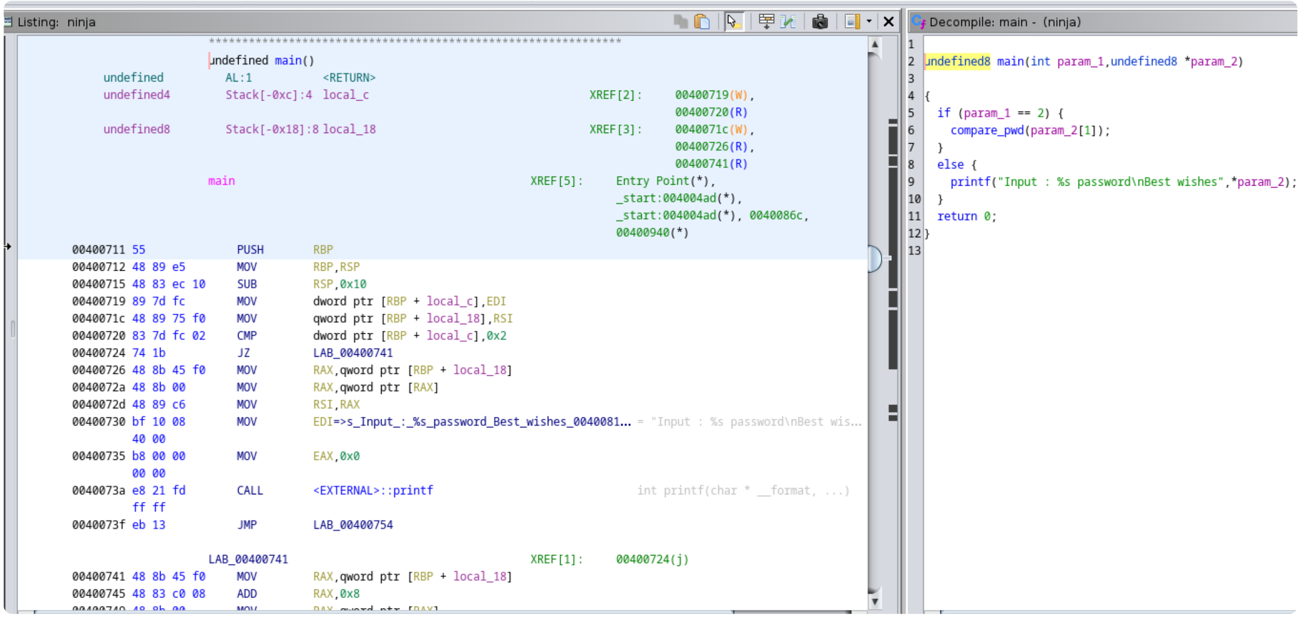
Strings:



```
┌──(kali㉿kali)-[~/Documents]
└─$ strings ninja
/lib64/ld-linux-x86-64.so.2
libc.so.6
puts
printf
__libc_start_main
__gmon_start__
GLIBC_2.2.5
UH-H
UH-H
[]A\A]A^A_
Flag Found!
password "%s" wrong
Input : %s password
Best wishes
;*3$"
GCC: (Ubuntu 4.8.2-19ubuntu1) 4.8.2
.symtab
.strtab
.shstrtab
.interp
.note.ABI-tag
.note.gnu.build-id
.gnu.hash
.dynsym
.dynstr
.gnu.version
.gnu.version_r
.rela.dyn
.rela.plt
.init
.text
```

Looking into Main Function

After analysing in Ghidra

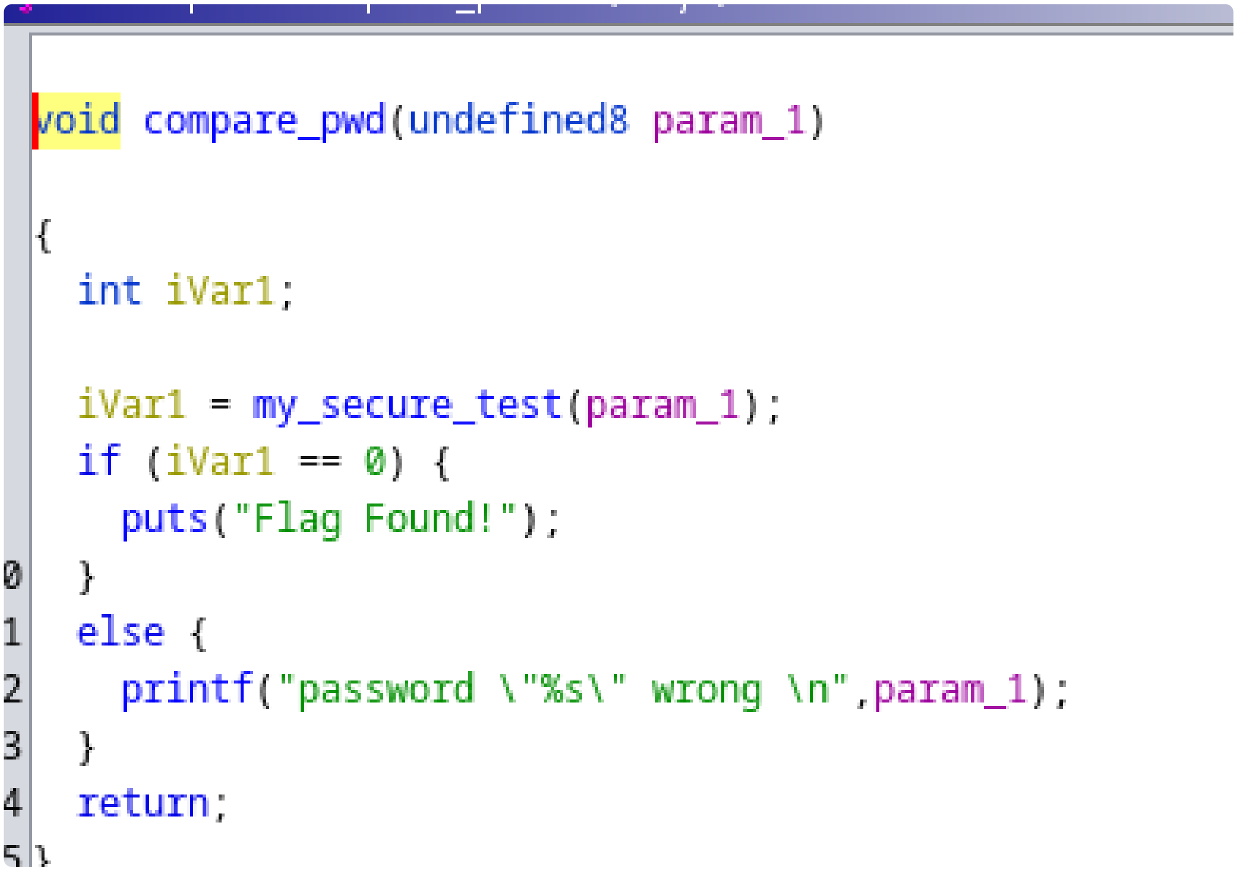


```
                    ****************************************************************
                    undefined main()
     undefined         AL:1           <RETURN>
     undefined4        Stack[-0xc]:4  local_c                          XREF[2]:     00400719(W),
                                                                                    00400720(R)
     undefined8        Stack[-0x18]:8 local_18                         XREF[3]:     0040071c(W),
                                                                                    00400726(R),
                                                                                    00400741(R)
                    main                                         XREF[5]:     Entry Point(*),
                                                                              _start:004004ad(*),
                                                                              _start:004004ad(*), 0040086c,
                                                                              00400940(*)
00400711 55              PUSH       RBP
00400712 48 89 e5        MOV        RBP,RSP
00400715 48 83 ec 10     SUB        RSP,0x10
00400719 89 7d fc        MOV        dword ptr [RBP + local_c],EDI
0040071c 48 89 75 f0     MOV        qword ptr [RBP + local_18],RSI
00400720 83 7d fc 02     CMP        dword ptr [RBP + local_c],0x2
00400724 74 1b           JZ         LAB_00400741
00400726 48 8b 45 f0     MOV        RAX,qword ptr [RBP + local_18]
0040072a 48 8b 00        MOV        RAX,qword ptr [RAX]
0040072d 48 89 c6        MOV        RSI,RAX
00400730 bf 10 08        MOV        EDI=>s_Input_:_%s_password_Best_wishes_0040081... = "Input : %s password\nBest wis...
         40 00
00400735 b8 00 00        MOV        EAX,0x0
         00 00
0040073a e8 21 fd        CALL       <EXTERNAL>::printf                          int printf(char * __format, ...)
         ff ff
0040073f eb 13           JMP        LAB_00400754

                    LAB_00400741                                 XREF[1]:     00400724(j)
00400741 48 8b 45 f0     MOV        RAX,qword ptr [RBP + local_18]
00400745 48 83 c0 08     ADD        RAX,0x8
00400749 48 8b 00        MOV        RAX,qword ptr [RAX]
```

```c

undefined8 main(int param_1,undefined8 *param_2)

{
  if (param_1 == 2) {
    compare_pwd(param_2[1]);
  }
  else {
    printf("Input : %s password\nBest wishes",*param_2);
  }
  return 0;
}

```

The `main` function takes two parameters: `param_1`, which represents the number of command-line arguments, and `param_2`, which is an array of command-line argument values.It checks if the program is invoked with exactly two arguments (`param_1 == 2`).

- If so, it calls the function `compare_pwd` with the second argument (`param_2[1]`). Presumably, `compare_pwd` is a function responsible for comparing the provided password with some predetermined value.
- If not, it prints a message indicating the correct usage of the program, which includes the program name followed by "password", and some additional text ("Best wishes").

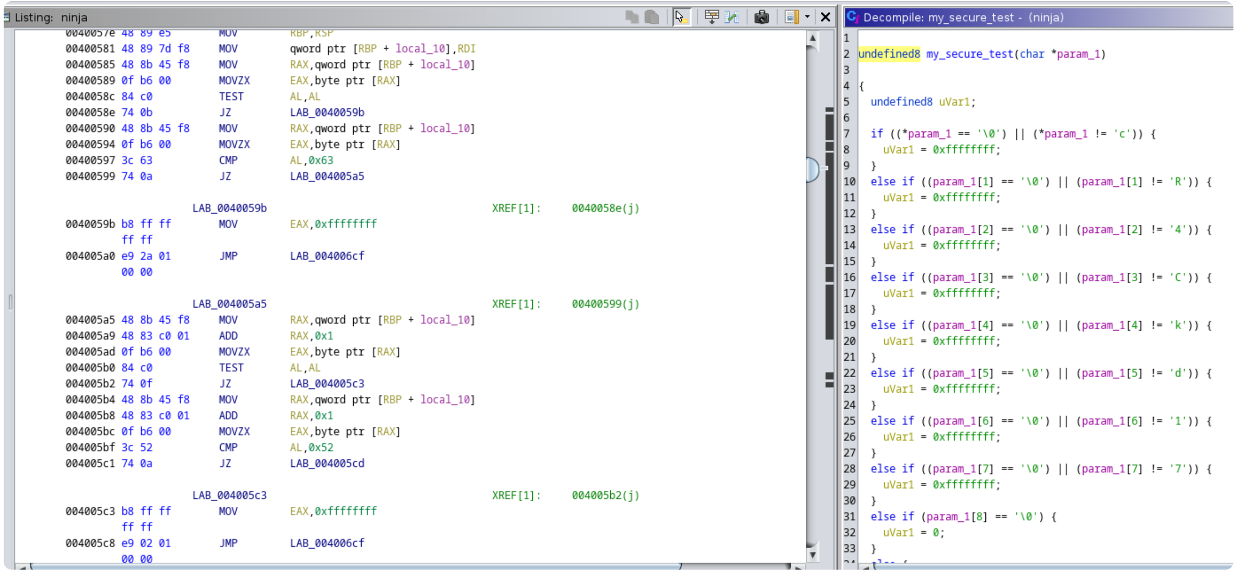


```
void compare_pwd(undefined8 param_1)

{
  int iVar1;

  iVar1 = my_secure_test(param_1);
  if (iVar1 == 0) {
    puts("Flag Found!");
  }
  else {
    printf("password \"%s\" wrong \n",param_1);
  }
  return;
}
```

It calls a function named `my_secure_test` with the provided password (`param_1`) as its argument.It stores the return value of `my_secure_test` in the variable `iVar1`.

It checks if the value stored in `iVar1` is equal to `0`.

- If it is, it prints "Flag Found!", indicating that the password is correct.
- If it's not, it prints a message indicating that the password is wrong, including the incorrect password itself.

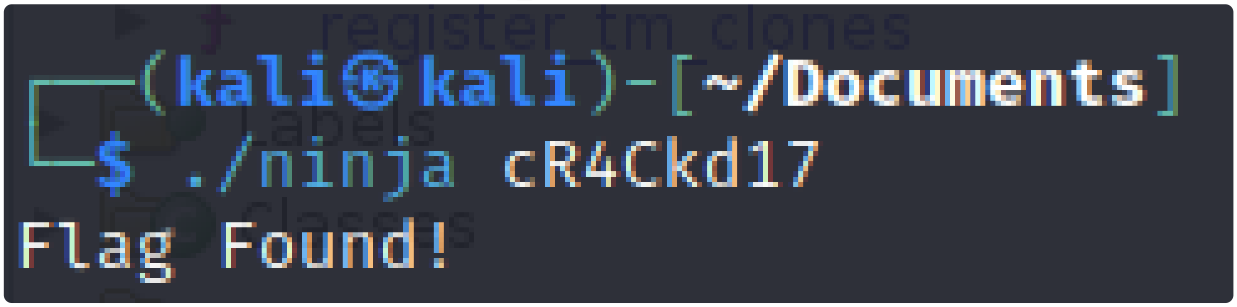


```
        0040057e 48 89 e5        MOV        RBP,RSP
        00400581 48 89 7d f8     MOV        qword ptr [RBP + local_10],RDI
        00400585 48 8b 45 f8     MOV        RAX,qword ptr [RBP + local_10]
        00400589 0f b6 00        MOVZX      EAX,byte ptr [RAX]
        0040058c 84 c0           TEST       AL,AL
        0040058e 74 0b           JZ         LAB_0040059b
        00400590 48 8b 45 f8     MOV        RAX,qword ptr [RBP + local_10]
        00400594 0f b6 00        MOVZX      EAX,byte ptr [RAX]
        00400597 3c 63           CMP        AL,0x63
        00400599 74 0a           JZ         LAB_004005a5

                             LAB_0040059b                                    XREF[1]:     0040058e(j)
        0040059b b8 ff ff        MOV        EAX,0xffffffff
                 ff ff
        004005a0 e9 2a 01        JMP        LAB_004006cf
                 00 00

                             LAB_004005a5                                    XREF[1]:     00400599(j)
        004005a5 48 8b 45 f8     MOV        RAX,qword ptr [RBP + local_10]
        004005a9 48 83 c0 01     ADD        RAX,0x1
        004005ad 0f b6 00        MOVZX      EAX,byte ptr [RAX]
        004005b0 84 c0           TEST       AL,AL
        004005b2 74 0f           JZ         LAB_004005c3
        004005b4 48 8b 45 f8     MOV        RAX,qword ptr [RBP + local_10]
        004005b8 48 83 c0 01     ADD        RAX,0x1
        004005bc 0f b6 00        MOVZX      EAX,byte ptr [RAX]
        004005bf 3c 52           CMP        AL,0x52
        004005c1 74 0a           JZ         LAB_004005cd

                             LAB_004005c3                                    XREF[1]:     004005b2(j)
        004005c3 b8 ff ff        MOV        EAX,0xffffffff
                 ff ff
        004005c8 e9 02 01        JMP        LAB_004006cf
                 00 00
```

```

undefined8 my_secure_test(char *param_1)

{
  undefined8 uVar1;

  if ((*param_1 == '\0') || (*param_1 != 'c')) {
    uVar1 = 0xffffffff;
  }
  else if ((param_1[1] == '\0') || (param_1[1] != 'R')) {
    uVar1 = 0xffffffff;
  }
  else if ((param_1[2] == '\0') || (param_1[2] != '4')) {
    uVar1 = 0xffffffff;
  }
  else if ((param_1[3] == '\0') || (param_1[3] != 'C')) {
    uVar1 = 0xffffffff;
  }
  else if ((param_1[4] == '\0') || (param_1[4] != 'k')) {
    uVar1 = 0xffffffff;
  }
  else if ((param_1[5] == '\0') || (param_1[5] != 'd')) {
    uVar1 = 0xffffffff;
  }
  else if ((param_1[6] == '\0') || (param_1[6] != '1')) {
    uVar1 = 0xffffffff;
  }
  else if ((param_1[7] == '\0') || (param_1[7] != '7')) {
    uVar1 = 0xffffffff;
  }
  else if (param_1[8] == '\0') {
    uVar1 = 0;
  }
  else {
```

It checks each character of the provided password against specific expected characters.

If any character doesn't match the expected character or if the password is not exactly 8 characters long, it returns `-1` (represented as `0xffffffff` in hexadecimal) to indicate failure.If all characters match the expected characters and the password is exactly 8 characters long, it returns `0` to indicate success.



```
┌──(kali㉿kali)-[~/Documents]
└─$ ./ninja cR4Ckd17
Flag Found!
```

Flag Format NOVA{ }

NOVA{cR4Ckd17}

Flag

NOVA{cR4Ckd17}