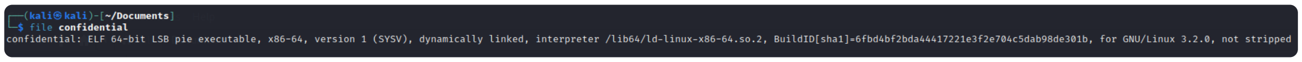
Challenge Title: "Cipher Quest"

Challenge Description:

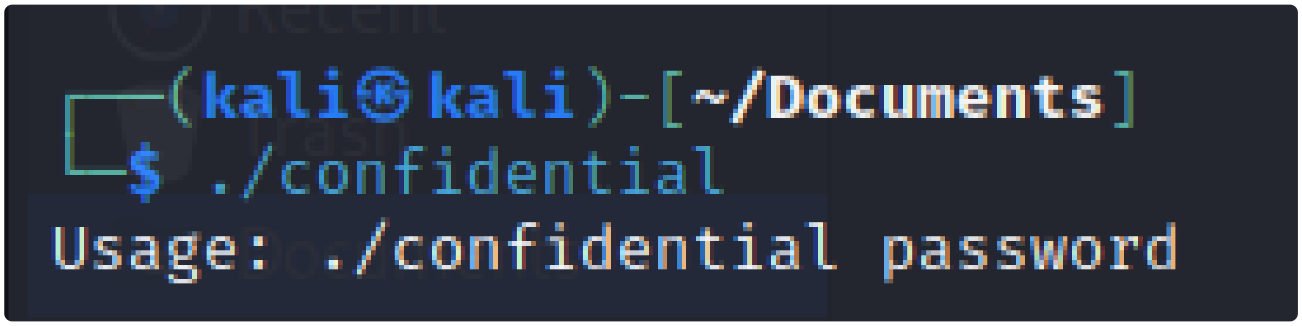
Alex,a cybersecurity analyst, received an email with an attachment named confidential. Upon opening, it revealed an attachment .Your task is to reverse engineer the attachment and uncover the hidden password. Join Alex in this "Cipher Quest" to decode the attachment and reveal the secret password.

Analysing the File:

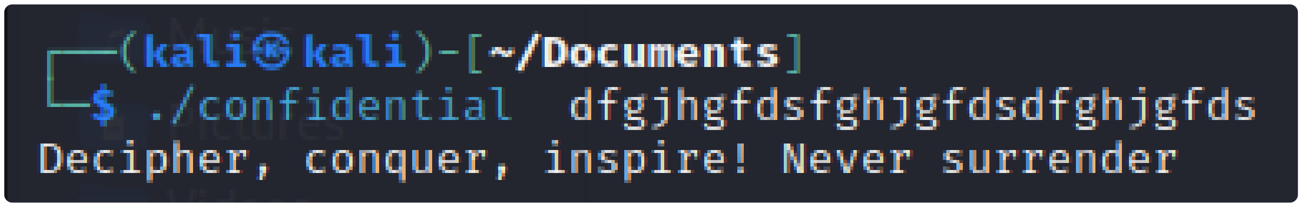


```
┌──(kali㉿kali)-[~/Documents]
└─$ file confidential
confidential: ELF 64-bit LSB pie executable, x86-64, version 1 (SYSV), dynamically linked, interpreter /lib64/ld-linux-x86-64.so.2, BuildID[sha1]=6fbd4bf2bda44417221e3f2e704c5dab98de301b, for GNU/Linux 3.2.0, not stripped
```

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.

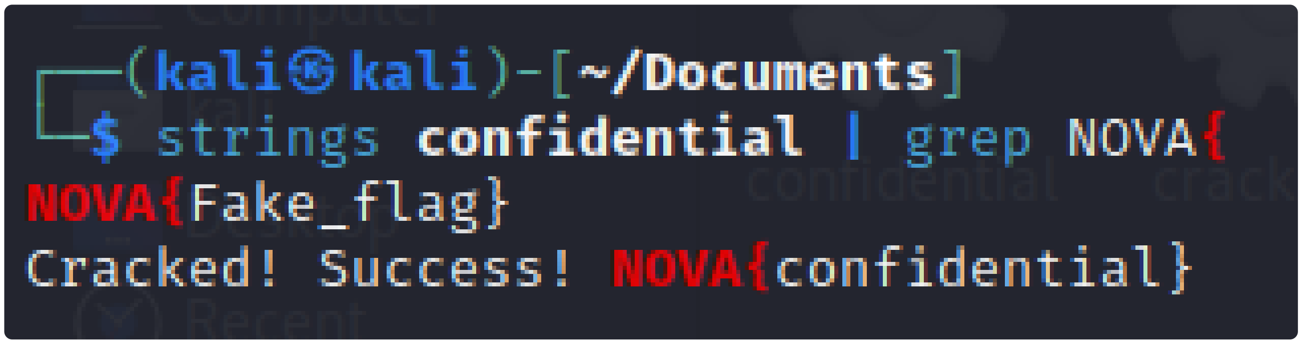


```
┌──(kali㉿kali)-[~/Documents]
└─$ ./confidential
Usage: ./confidential password
```



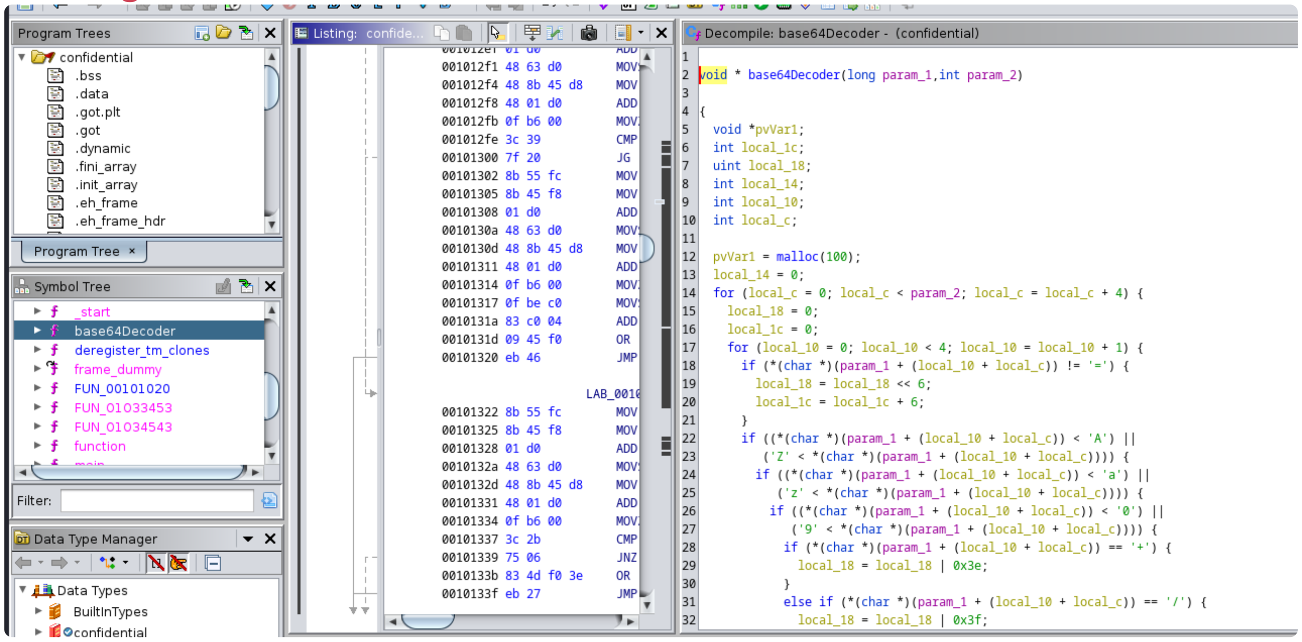
```
┌──(kali㉿kali)-[~/Documents]
└─$ ./confidential  dfgjhgfdsfghjgfdsdfghjgfds
Decipher, conquer, inspire! Never surrender
```

Strings:



```
┌──(kali㉿kali)-[~/Documents]
└─$ strings confidential | grep NOVA{
NOVA{Fake_flag}
Cracked! Success! NOVA{confidential}
```

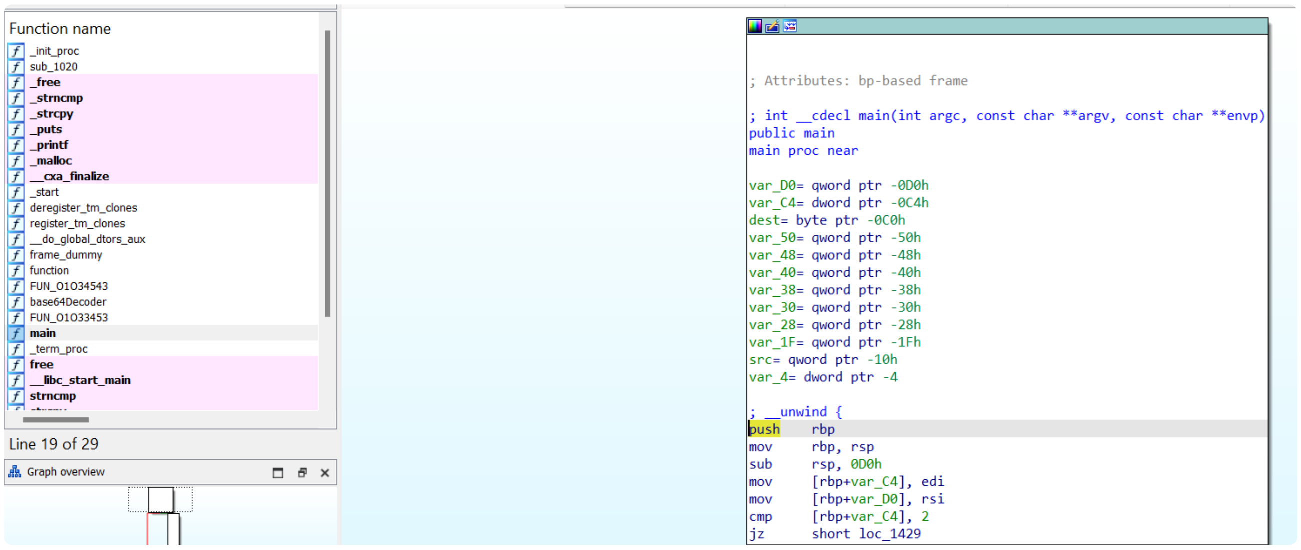
Looking into Main Function



```
001012f1 48 63 d0       MOV
001012f4 48 8b 45 d8    MOV
001012f8 48 01 d0       ADD
001012fb 0f b6 00       MOV
001012fe 3c 39          CMP
00101300 7f 20          JG
00101302 8b 55 fc       MOV
00101305 8b 45 f8       MOV
00101308 01 d0          ADD
0010130a 48 63 d0       MOV
0010130d 48 8b 45 d8    MOV
00101311 48 01 d0       ADD
00101314 0f b6 00       MOV
00101317 0f be c0       MOV
0010131a 83 c0 04       ADD
0010131d 09 45 f0       OR
00101320 eb 46          JMP

                        LAB_0010
00101322 8b 55 fc       MOV
00101325 8b 45 f8       MOV
00101328 01 d0          ADD
0010132a 48 63 d0       MOV
0010132d 48 8b 45 d8    MOV
00101331 48 01 d0       ADD
00101334 0f b6 00       MOV
00101337 3c 2b          CMP
00101339 75 06          JNZ
0010133b 83 4d f0 3e    OR
0010133f eb 27          JMP
```

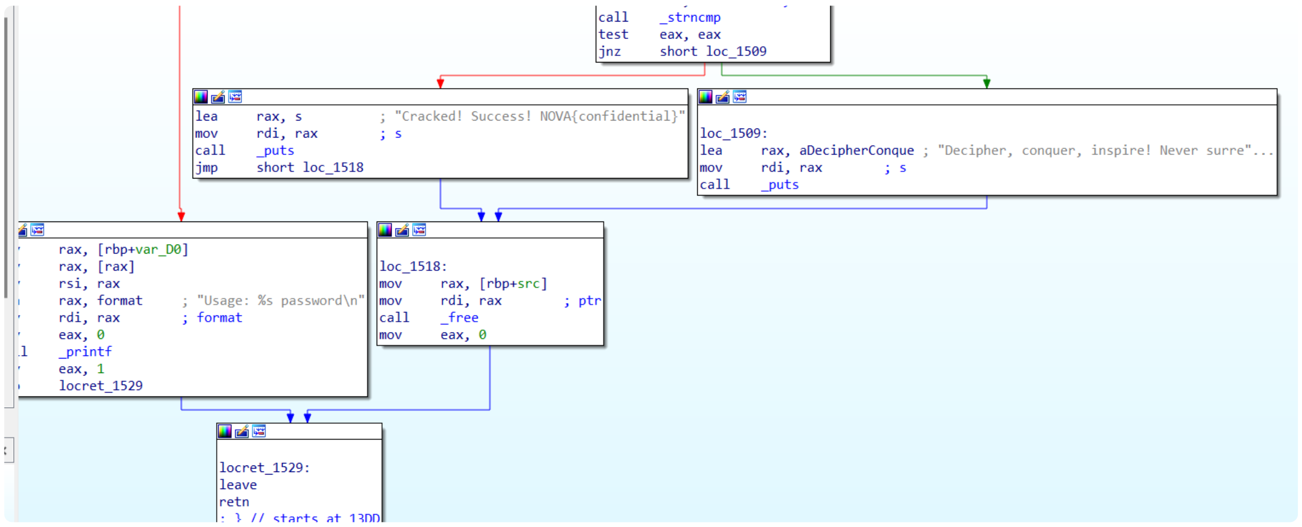
```c

void * base64Decoder(long param_1,int param_2)

{
  void *pvVar1;
  int local_1c;
  uint local_18;
  int local_14;
  int local_10;
  int local_c;

  pvVar1 = malloc(100);
  local_14 = 0;
  for (local_c = 0; local_c < param_2; local_c = local_c + 4) {
    local_18 = 0;
    local_1c = 0;
    for (local_10 = 0; local_10 < 4; local_10 = local_10 + 1) {
      if (*(char *)(param_1 + (local_10 + local_c)) != '=') {
        local_18 = local_18 << 6;
        local_1c = local_1c + 6;
      }
      if ((*(char *)(param_1 + (local_10 + local_c)) < 'A') ||
         ('Z' < *(char *)(param_1 + (local_10 + local_c)))) {
        if ((*(char *)(param_1 + (local_10 + local_c)) < 'a') ||
           ('z' < *(char *)(param_1 + (local_10 + local_c)))) {
          if ((*(char *)(param_1 + (local_10 + local_c)) < '0') ||
             ('9' < *(char *)(param_1 + (local_10 + local_c)))) {
            if (*(char *)(param_1 + (local_10 + local_c)) == '+') {
              local_18 = local_18 | 0x3e;
            }
            else if (*(char *)(param_1 + (local_10 + local_c)) == '/') {
              local_18 = local_18 | 0x3f;
```

File opened in IDA and analysed



```
; Attributes: bp-based frame

; int __cdecl main(int argc, const char **argv, const char **envp)
public main
main proc near

var_D0= qword ptr -0D0h
var_C4= dword ptr -0C4h
dest= byte ptr -0C0h
var_50= qword ptr -50h
var_48= qword ptr -48h
var_40= qword ptr -40h
var_38= qword ptr -38h
var_30= qword ptr -30h
var_28= qword ptr -28h
var_1F= qword ptr -1Fh
src= qword ptr -10h
var_4= dword ptr -4

; __unwind {
push    rbp
mov     rbp, rsp
sub     rsp, 0D0h
mov     [rbp+var_C4], edi
mov     [rbp+var_D0], rsi
cmp     [rbp+var_C4], 2
jz      short loc_1429
```

Looking into main function and program flow



```
call    _strncmp
test    eax, eax
jnz     short loc_1509
```

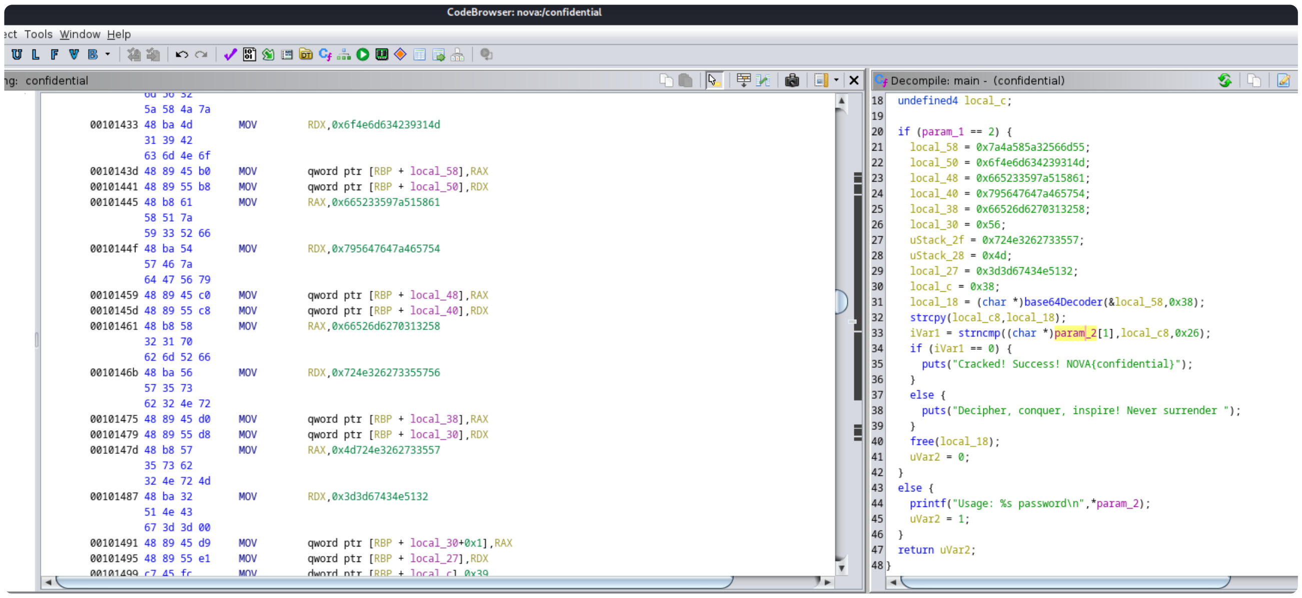
```
lea     rax, s          ; "Cracked! Success! NOVA{confidential}"
mov     rdi, rax        ; s
call    _puts
jmp     short loc_1518
```

```
loc_1509:
lea     rax, aDecipherConque ; "Decipher, conquer, inspire! Never surre"...
mov     rdi, rax        ; s
call    _puts
```

```
        rax, [rbp+var_D0]
        rax, [rax]
        rsi, rax
        rax, format     ; "Usage: %s password\n"
        rdi, rax        ; format
        eax, 0
        _printf
        eax, 1
        locret_1529
```

```
loc_1518:
mov     rax, [rbp+src]
mov     rdi, rax        ; ptr
call    _free
mov     eax, 0
```

```
locret_1529:
leave
retn
; } // starts at 13DD
```

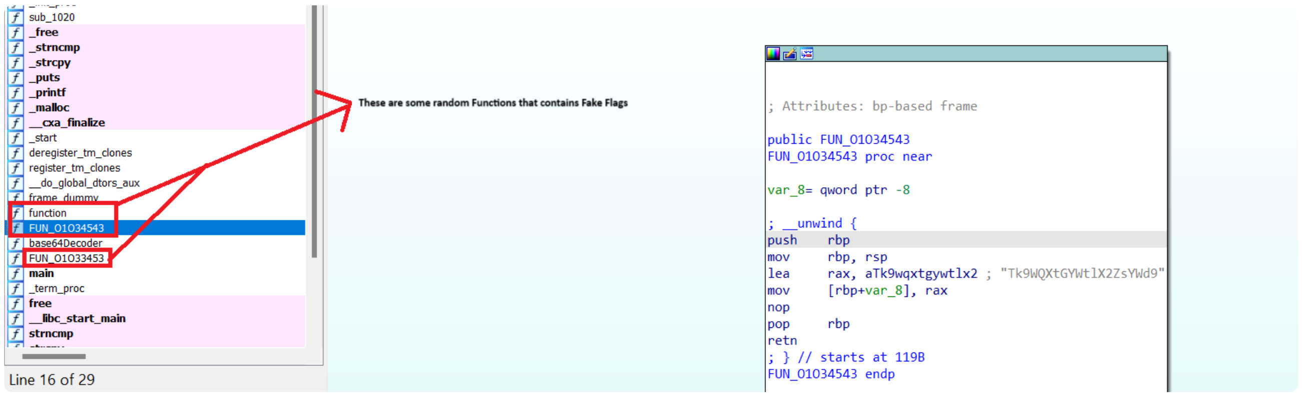


CodeBrowser: nova:/confidential

```
                     5a 58 4a 7a
        00101433 48 ba 4d       MOV     RDX,0x6f4e6d634239314d
                 31 39 42
                 63 6d 4e 6f
        0010143d 48 89 45 b0    MOV     qword ptr [RBP + local_58],RAX
        00101441 48 89 55 b8    MOV     qword ptr [RBP + local_50],RDX
        00101445 48 b8 61       MOV     RAX,0x665233597a515861
                 58 51 7a
                 59 33 52 66
        0010144f 48 ba 54       MOV     RDX,0x795647647a465754
                 57 46 7a
                 64 47 56 79
        00101459 48 89 45 c0    MOV     qword ptr [RBP + local_48],RAX
        0010145d 48 89 55 c8    MOV     qword ptr [RBP + local_40],RDX
        00101461 48 b8 58       MOV     RAX,0x66526d6270313258
                 32 31 70
                 62 6d 52 66
        0010146b 48 ba 56       MOV     RDX,0x724e326273355756
                 57 35 73
                 62 32 4e 72
        00101475 48 89 45 d0    MOV     qword ptr [RBP + local_38],RAX
        00101479 48 89 55 d8    MOV     qword ptr [RBP + local_30],RDX
        0010147d 48 b8 57       MOV     RAX,0x4d724e3262733557
                 35 73 62
                 32 4e 72 4d
        00101487 48 ba 32       MOV     RDX,0x3d3d67434e5132
                 51 4e 43
                 67 3d 3d 00
        00101491 48 89 45 d9    MOV     qword ptr [RBP + local_30+0x1],RAX
        00101495 48 89 55 e1    MOV     qword ptr [RBP + local_27],RDX
```

Decompile: main - (confidential)

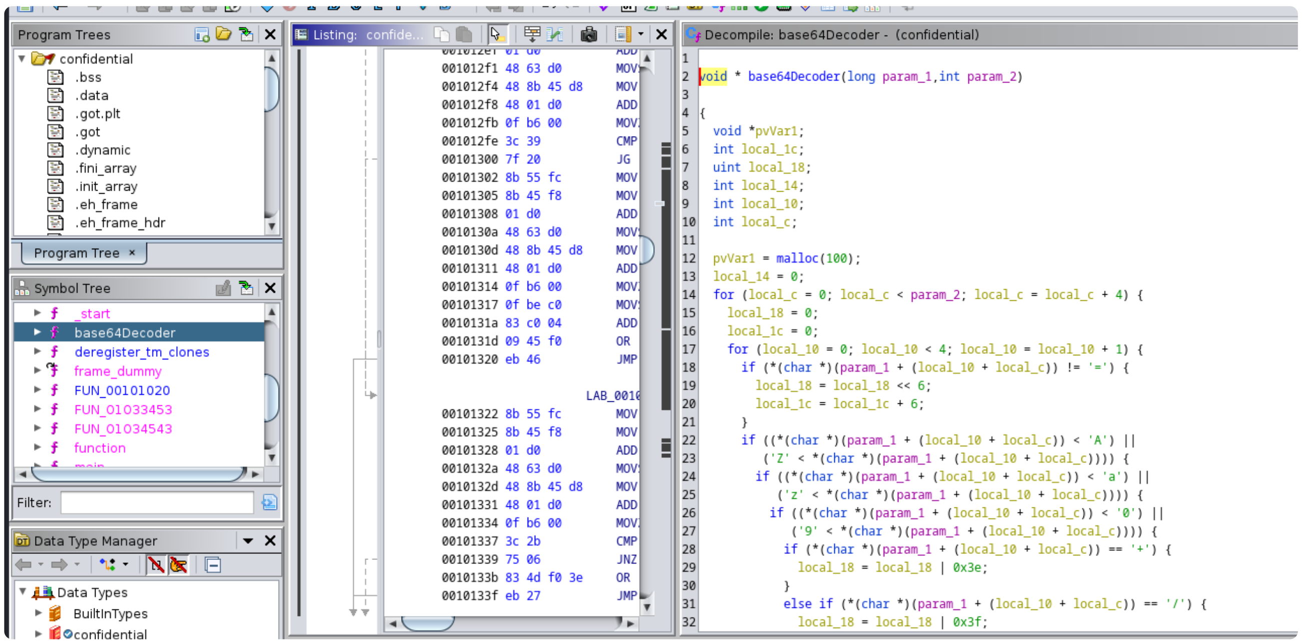
```c
  undefined4 local_c;

  if (param_1 == 2) {
    local_58 = 0x7a4a585a32566d55;
    local_50 = 0x6f4e6d634239314d;
    local_48 = 0x665233597a515861;
    local_40 = 0x795647647a465754;
    local_38 = 0x66526d6270313258;
    local_30 = 0x56;
    uStack_2f = 0x724e3262733557;
    uStack_28 = 0x4d;
    local_27 = 0x3d3d67434e5132;
    local_c = 0x38;
    local_18 = (char *)base64Decoder(&local_58,0x38);
    strcpy(local_c8,local_18);
    iVar1 = strncmp((char *)param_2[1],local_c8,0x26);
    if (iVar1 == 0) {
      puts("Cracked! Success! NOVA{confidential}");
    }
    else {
      puts("Decipher, conquer, inspire! Never surrender ");
    }
    free(local_18);
    uVar2 = 0;
  }
  else {
    printf("Usage: %s password\n",*param_2);
    uVar2 = 1;
  }
  return uVar2;
}
```



These are some random Functions that contains Fake Flags

```
; Attributes: bp-based frame

public FUN_01034543
FUN_01034543 proc near

var_8= qword ptr -8

; __unwind {
push    rbp
mov     rbp, rsp
lea     rax, aTk9wqxtgywtlx2 ; "Tk9WQXtGYWtlX2ZsYWd9"
mov     [rbp+var_8], rax
nop
pop     rbp
retn
; } // starts at 119B
FUN_01034543 endp
```

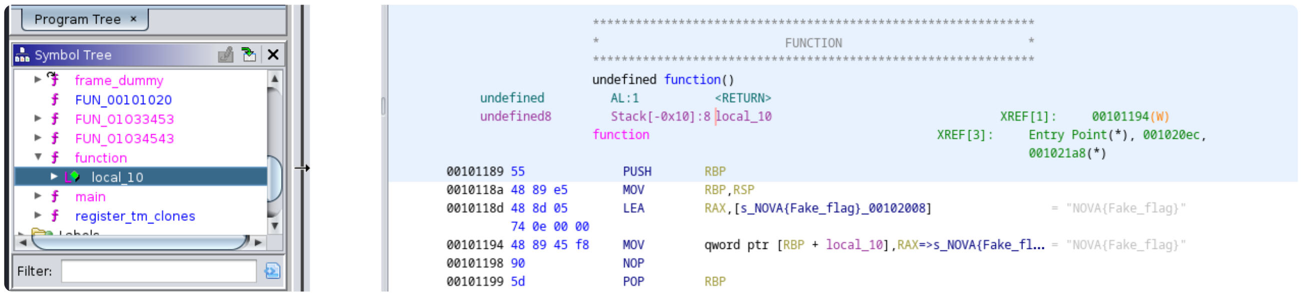
These are the fucntions that contains Fake flags. You know if you tried decoding the base-64 strings.



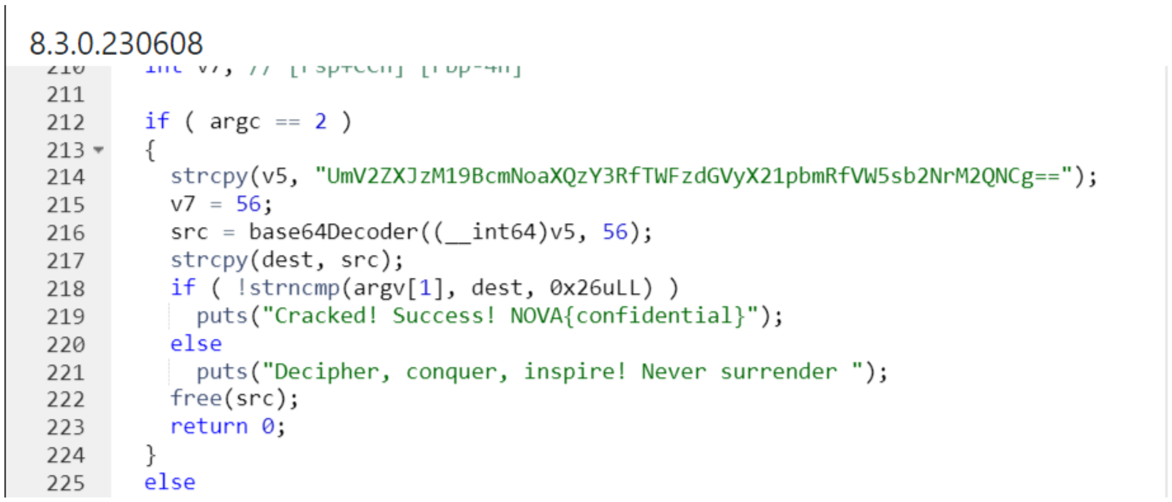
```
001012f1 48 63 d0       MOV
001012f4 48 8b 45 d8    MOV
001012f8 48 01 d0       ADD
001012fb 0f b6 00       MOV
001012fe 3c 39          CMP
00101300 7f 20          JG
00101302 8b 55 fc       MOV
00101305 8b 45 f8       MOV
00101308 01 d0          ADD
0010130a 48 63 d0       MOV
0010130d 48 8b 45 d8    MOV
00101311 48 01 d0       ADD
00101314 0f b6 00       MOV
00101317 0f be c0       MOV
0010131a 83 c0 04       ADD
0010131d 09 45 f0       OR
00101320 eb 46          JMP

                     LAB_0010
00101322 8b 55 fc       MOV
00101325 8b 45 f8       MOV
00101328 01 d0          ADD
0010132a 48 63 d0       MOV
0010132d 48 8b 45 d8    MOV
00101331 48 01 d0       ADD
00101334 0f b6 00       MOV
00101337 3c 2b          CMP
00101339 75 06          JNZ
0010133b 83 4d f0 3e    OR
0010133f eb 27          JMP
```

```c

void * base64Decoder(long param_1,int param_2)

{
  void *pvVar1;
  int local_1c;
  uint local_18;
  int local_14;
  int local_10;
  int local_c;

  pvVar1 = malloc(100);
  local_14 = 0;
  for (local_c = 0; local_c < param_2; local_c = local_c + 4) {
    local_18 = 0;
    local_1c = 0;
    for (local_10 = 0; local_10 < 4; local_10 = local_10 + 1) {
      if (*(char *)(param_1 + (local_10 + local_c)) != '=') {
        local_18 = local_18 << 6;
        local_1c = local_1c + 6;
      }
      if ((*(char *)(param_1 + (local_10 + local_c)) < 'A') ||
         ('Z' < *(char *)(param_1 + (local_10 + local_c)))) {
        if ((*(char *)(param_1 + (local_10 + local_c)) < 'a') ||
           ('z' < *(char *)(param_1 + (local_10 + local_c)))) {
          if ((*(char *)(param_1 + (local_10 + local_c)) < '0') ||
             ('9' < *(char *)(param_1 + (local_10 + local_c)))) {
            if (*(char *)(param_1 + (local_10 + local_c)) == '+') {
              local_18 = local_18 | 0x3e;
            }
            else if (*(char *)(param_1 + (local_10 + local_c)) == '/') {
              local_18 = local_18 | 0x3f;
```

This is theFunction to decode Base64



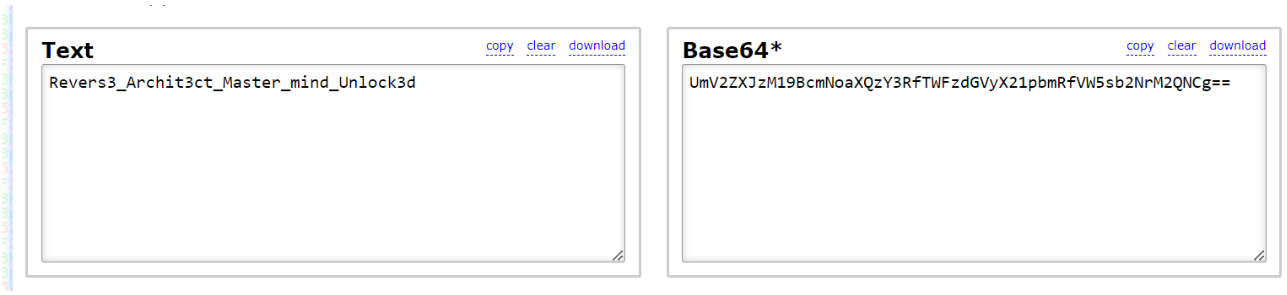
```
                     **************************************************************
                     *                          FUNCTION                          *
                     **************************************************************
                     undefined function()
       undefined        AL:1           <RETURN>
       undefined8       Stack[-0x10]:8 local_10                        XREF[1]:     00101194(W)
                     function                                       XREF[3]:     Entry Point(*), 001020ec,
                                                                                 001021a8(*)
00101189 55              PUSH       RBP
0010118a 48 89 e5        MOV        RBP,RSP
0010118d 48 8d 05        LEA        RAX,[s_NOVA{Fake_flag}_00102008]              = "NOVA{Fake_flag}"
         74 0e 00 00
00101194 48 89 45 f8     MOV        qword ptr [RBP + local_10],RAX=>s_NOVA{Fake_fl... = "NOVA{Fake_flag}"
00101198 90              NOP
00101199 5d              POP        RBP
```



8.3.0.230608

```c
  int v7; // [rsp+Ch] [rbp-4h]

  if ( argc == 2 )
  {
    strcpy(v5, "UmV2ZXJzM19BcmNoaXQzY3RfTWFzdGVyX21pbmRfVW5sb2NrM2QNCg==");
    v7 = 56;
    src = base64Decoder((__int64)v5, 56);
    strcpy(dest, src);
    if ( !strncmp(argv[1], dest, 0x26uLL) )
      puts("Cracked! Success! NOVA{confidential}");
    else
      puts("Decipher, conquer, inspire! Never surrender ");
    free(src);
    return 0;
  }
  else
```

This is the base-64 encrypted text we need to decode to get the secret password.

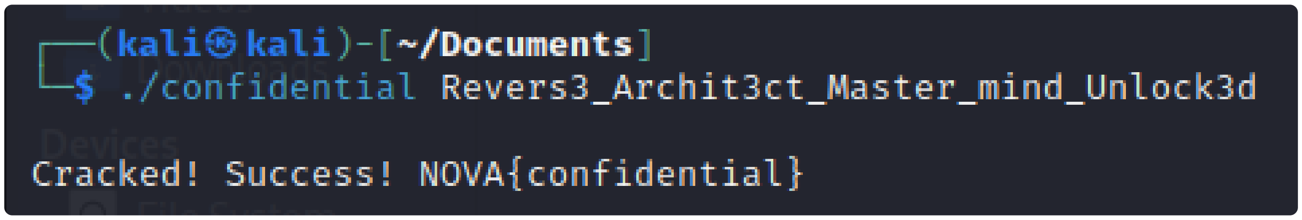


Text copy clear download

```
Revers3_Archit3ct_Master_mind_Unlock3d
```

Base64* copy clear download

```
UmV2ZXJzM19BcmNoaXQzY3RfTWFzdGVyX21pbmRfVW5sb2NrM2QNCg==
```



```
┌──(kali㉿kali)-[~/Documents]
└─$ ./confidential Revers3_Archit3ct_Master_mind_Unlock3d

Cracked! Success! NOVA{confidential}
```

Now enclose the flag in Flag Format.

Flag

NOVA{Revers3_Archit3ct_Master_mind_Unlock3d}