Challenge

The Elf's Wager

Attachments:day4

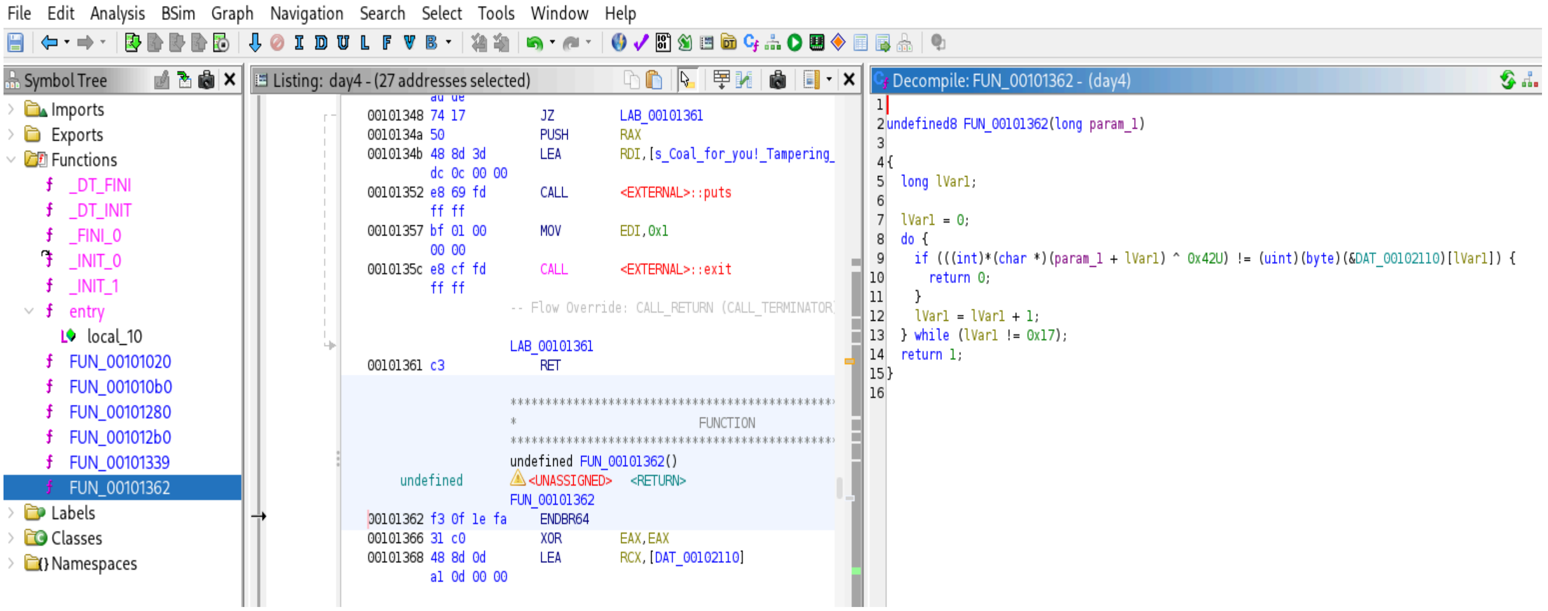
Solution:

run the commands to make it executable

1- chmod +x day4

2- ./day4

Open this ELF in Ghidra tool to reverse it



```
                   ad de
00101348 74 17          JZ         LAB_00101361
0010134a 50             PUSH       RAX
0010134b 48 8d 3d       LEA        RDI,[s_Coal_for_you!_Tampering_
         dc 0c 00 00
00101352 e8 69 fd       CALL       <EXTERNAL>::puts
         ff ff
00101357 bf 01 00       MOV        EDI,0x1
         00 00
0010135c e8 cf fd       CALL       <EXTERNAL>::exit
         ff ff
                   -- Flow Override: CALL_RETURN (CALL_TERMINATOR

                   LAB_00101361
00101361 c3             RET

                   **********************************************
                   *                          FUNCTION
                   **********************************************
                   undefined FUN_00101362()
    undefined         <UNASSIGNED>   <RETURN>
                   FUN_00101362
00101362 f3 0f 1e fa    ENDBR64
00101366 31 c0          XOR        EAX,EAX
00101368 48 8d 0d       LEA        RCX,[DAT_00102110]
         a1 0d 00 00
```

```

undefined8 FUN_00101362(long param_1)

{
  long lVar1;

  lVar1 = 0;
  do {
    if (((int)*(char *)(param_1 + lVar1) ^ 0x42U) != (uint)(byte)(&DAT_00102110)[lVar1]) {
      return 0;
    }
    lVar1 = lVar1 + 1;
  } while (lVar1 != 0x17);
  return 1;
}

```

```
undefined8 FUN_00101362(long param_1)

{

  long lVar1 = 0;

  do {

      if (((int)*(char *)(param_1 + lVar1) ^ 0x42U) !=
(uint)(byte)(&DAT_00102110)[lVar1]) {

      return 0;

      }

      lVar1 = lVar1 + 1;

  } while (lVar1 != 0x17);

  return 1;

}
```

For each character of your input:

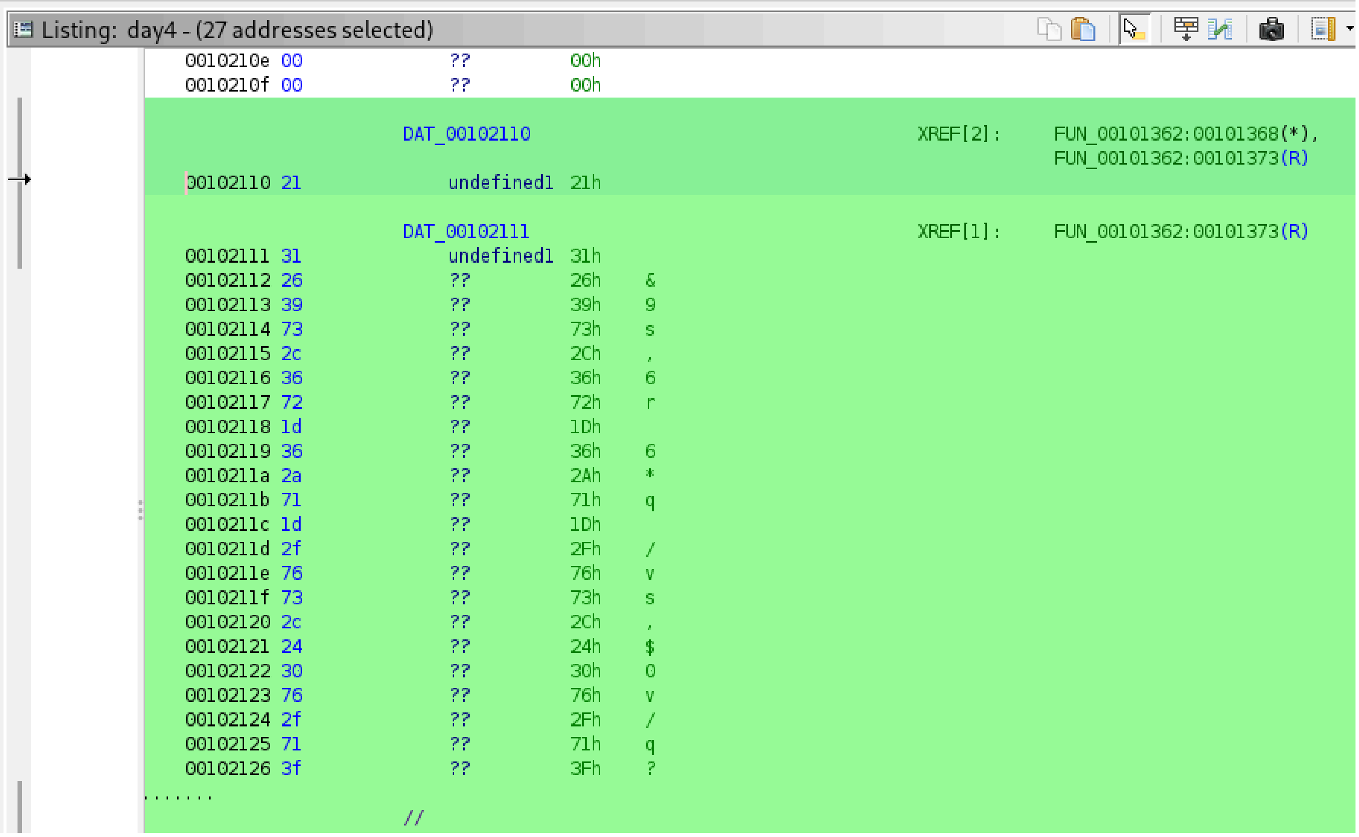
```
input[i] XOR 0x42 == DAT_00102110[i]
```

So:

```
input[i] = DAT[i] XOR 0x42
```

And it checks for 0x17 = 23 characters.

Extracting the &DAT_00102110:



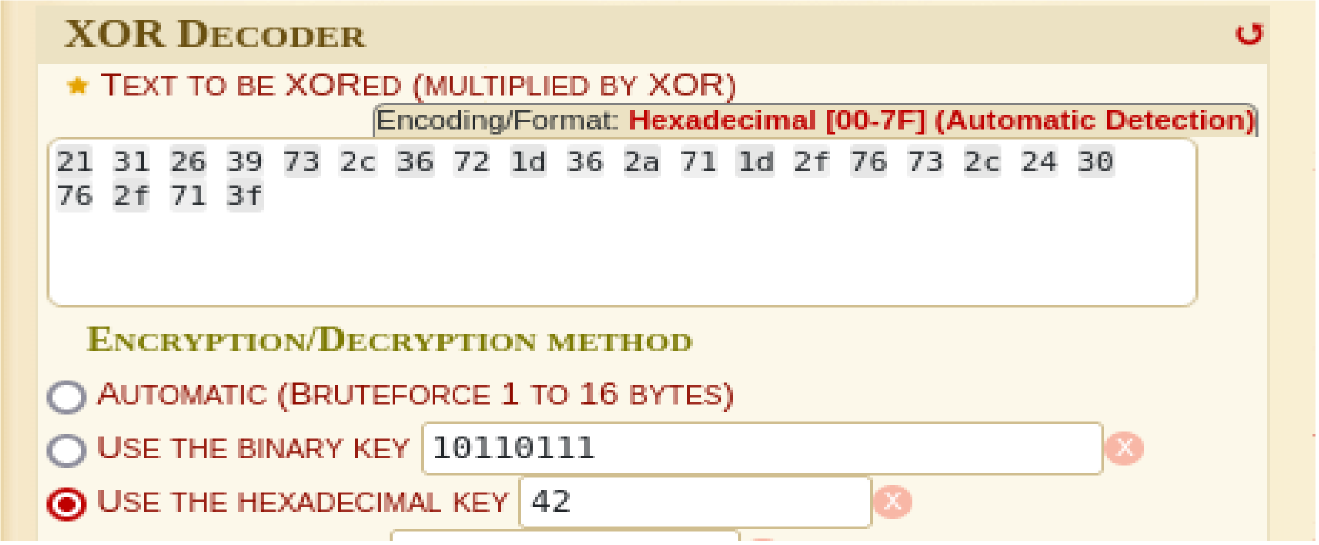
Listing: day4 - (27 addresses selected)

```
0010210e 00         ??          00h
0010210f 00         ??          00h

                 DAT_00102110                            XREF[2]:     FUN_00101362:00101368(*),
                                                                      FUN_00101362:00101373(R)
00102110 21         undefined1  21h

                 DAT_00102111                            XREF[1]:     FUN_00101362:00101373(R)
00102111 31         undefined1  31h
00102112 26         ??          26h    &
00102113 39         ??          39h    9
00102114 73         ??          73h    s
00102115 2c         ??          2Ch    ,
00102116 36         ??          36h    6
00102117 72         ??          72h    r
00102118 1d         ??          1Dh
00102119 36         ??          36h    6
0010211a 2a         ??          2Ah    *
0010211b 71         ??          71h    q
0010211c 1d         ??          1Dh
0010211d 2f         ??          2Fh    /
0010211e 76         ??          76h    v
0010211f 73         ??          73h    s
00102120 2c         ??          2Ch    ,
00102121 24         ??          24h    $
00102122 30         ??          30h    0
00102123 76         ??          76h    v
00102124 2f         ??          2Fh    /
00102125 71         ??          71h    q
00102126 3f         ??          3Fh    ?
.......
                 //
```

21 31 26 39 73 2c 36 72 1d 36 2a 71 1d 2f 76 73 2c 24 30 76 2f 71 3f



XOR Decoder

★ Text to be XORed (multiplied by XOR)

Encoding/Format: Hexadecimal [00-7F] (Automatic Detection)

```
21 31 26 39 73 2c 36 72 1d 36 2a 71 1d 2f 76 73 2c 24 30
76 2f 71 3f
```

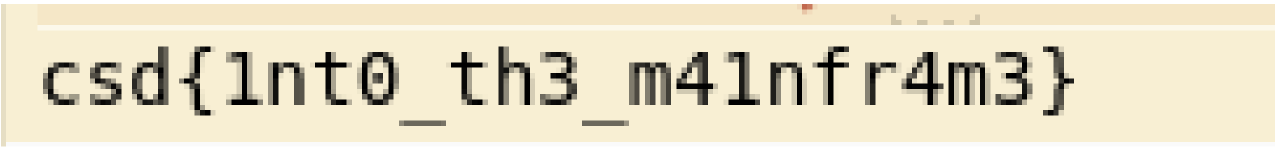
Encryption/Decryption method

○ Automatic (Bruteforce 1 to 16 bytes)

○ Use the binary key 10110111

● Use the hexadecimal key 42

Final Output:



```
csd{1nt0_th3_m41nfr4m3}
```