

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

The screenshot shows the Ghidra debugger interface with three main panes:

- Symbol Tree:** On the left, it shows the structure of the ELF file, including sections like Imports, Exports, Functions, and Labels. The function `FUN_00101362` is selected.
- Listing:** The middle pane displays the assembly code for the selected function. The assembly listing shows instructions such as `JZ LAB_00101361`, `PUSH RAX`, `LEA RDI, [s_Coal_for_you!_Tampering_]`, `CALL <EXTERNAL>::puts`, `MOV EDI, 0x1`, `CALL <EXTERNAL>::exit`, and `RET`. A flow override is shown for the `CALL_TERMINATOR` instruction.
- Decompile:** The right pane shows the decompiled C code for the function. The code initializes a long variable `lVar1` to 0, enters a do-while loop, and checks if the value at memory location `param_1 + lVar1` matches `0x42U`. If it does, it returns 0; otherwise, it increments `lVar1` and continues the loop until `lVar1` is not equal to `0x17`.

```
1 undefined8 FUN_00101362(long param_1)
2 {
3     long lVar1;
4     lVar1 = 0;
5     do {
6         if (((int)*(char *))(param_1 + lVar1) ^ 0x42U) != (uint)(byte)(DAT_00102110)[lVar1]) {
7             return 0;
8         }
9         lVar1 = lVar1 + 1;
10    } while (lVar1 != 0x17);
11    return 1;
12 }
```

```
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:

The screenshot shows a debugger interface with a listing window. The title bar says "Listing: day4 - (27 addresses selected)". The assembly code is as follows:

```
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 ×

USE THE HEXADECIMAL KEY ×

Final Output:

```
csd{1nt0_th3_m41nfr4m3}
```