Running_out_of_time

Description

CHALLENGE 196 SOLVES

Running Out of Time

300

A mysterious program asks for a specific number, but the correct value changes every time you run it. Can you figure out how the number is generated and retrieve the hidden flag?

Analyze the binary, reverse-engineer the logic, and find a way to predict the correct input to trigger the win condition.

Running_Out...

Flag

Submit

Category

#binary

Solution

The obvious thing to do when I see an elf file is to check out in Ghidra. I find a main function comparing the string with a local_c value, if corrects it calls a function.

```
int __cdecl main(int _Argc,char **_Argv,char **_Env)

{
  int iVar1;
  __time64_t _Var2;
  FILE *_File;
  char local_38 [44];
  int local_c;

  __main();
  _Var2 = time((__time64_t *)0x0);
  srand((uint)_Var2);
  local_c = rand();
  local_c = local_c % 100;
  printf("Provide the correct value: ");
  _File = __iob_func();
  fgets(local_38,0x20,_File);
  iVar1 = atoi(local_38);
  if (iVar1 == local_c) {
    p3xr9q_tlzz();
  }
  else {
    puts("Incorrect. Please try again.");
  }
  return 0;
}
```

So I check what that function does:

Decompile: p3xr9q_tlzz - (Running_Out_Of_Time.exe)

```
void p3xr9q_tlzz(void)

{
  byte local_28 [27];
  byte local_d;
  uint local_c;

  local_28[0] = 0x1d;
  local_28[1] = 0x1b;
  local_28[2] = 0x47;
  local_28[3] = 0x19;
  local_28[4] = 0x75;
  local_28[5] = 0x1f;
  local_28[6] = 0x1d;
  local_28[7] = 0x1a;
  local_28[8] = 0x5a;
  local_28[9] = 0x5a;
  local_28[10] = 0x19;
  local_28[11] = 0x4e;
  local_d = 0x2a;
  printf("Success! Here is your output: ");
  for (local_c = 0; local_c < 0xc; local_c = local_c + 1) {
    putchar((int)(char)(local_28[(int)local_c] ^ local_d));
  }
  putchar(10);
  return;
}
```

The loop performs a XOR operation for each character in the local_28 array
So technically, this problem is solved by understanding the output encryption and not even the input sanitization.

Which is literally the flag.