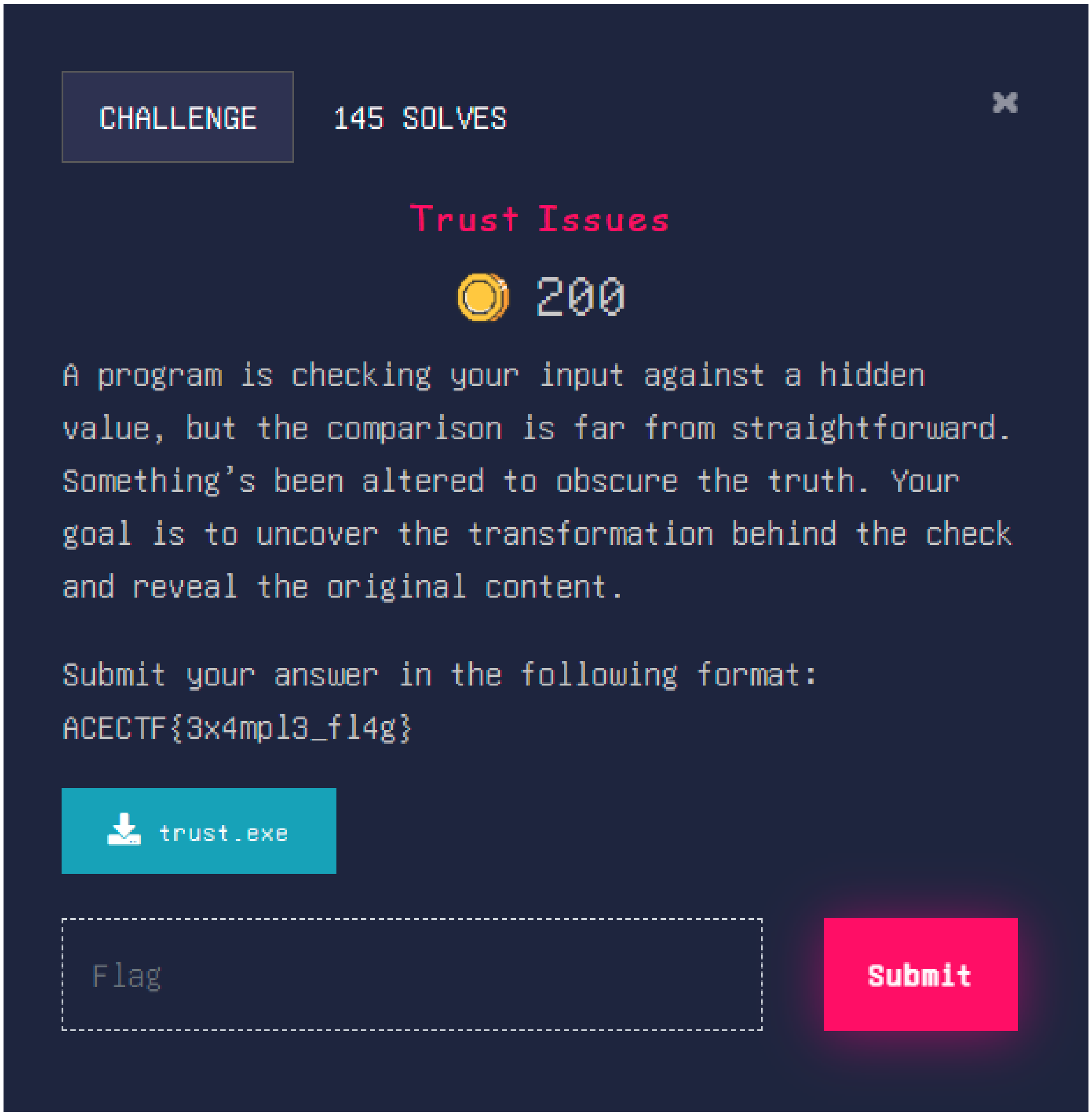
Trust_Issues

Description



CHALLENGE

145 SOLVES

Trust Issues

200

A program is checking your input against a hidden value, but the comparison is far from straightforward. Something's been altered to obscure the truth. Your goal is to uncover the transformation behind the check and reveal the original content.

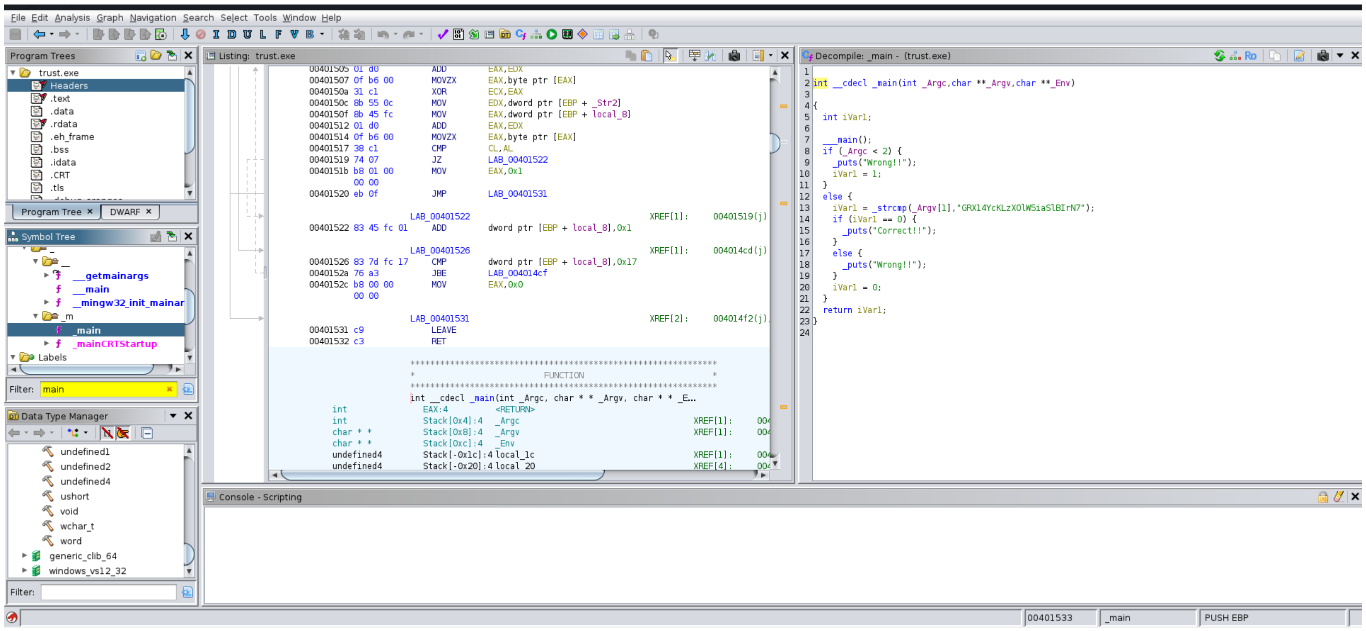
Submit your answer in the following format: ACECTF{3x4mpl3_f14g}

Category

#reverse

I first open the file with Ghidra since running `file` says it is a PE32 file, so it's worth checking the code first

Solution



```
00401505 01 d0          ADD        EAX,EDX
00401507 0f b6 00       MOVZX      EAX,byte ptr [EAX]
0040150a 31 c1          XOR        ECX,EAX
0040150c 8b 55 0c       MOV        EDX,dword ptr [EBP + _Str2]
0040150f 8b 45 fc       MOV        EAX,dword ptr [EBP + local_8]
00401512 01 d0          ADD        EAX,EDX
00401514 0f b6 00       MOVZX      EAX,byte ptr [EAX]
00401517 38 c1          CMP        CL,AL
00401519 74 07          JZ         LAB_00401522
0040151b b8 01 00       MOV        EAX,0x1
         00 00
00401520 eb 0f          JMP        LAB_00401531

                     LAB_00401522                          XREF[1]:     00401519(j)
00401522 83 45 fc 01    ADD        dword ptr [EBP + local_8],0x1

                     LAB_00401526                          XREF[1]:     004014cd(j)
00401526 83 7d fc 17    CMP        dword ptr [EBP + local_8],0x17
0040152a 76 a3          JBE        LAB_004014cf
0040152c b8 00 00       MOV        EAX,0x0
         00 00

                     LAB_00401531                          XREF[2]:     004014f2(j)
00401531 c9             LEAVE
00401532 c3             RET

                     **************************************************************
                     *                          FUNCTION                          *
                     **************************************************************
                     int __cdecl _main(int _Argc, char * * _Argv, char * * _E…
     int              EAX:4          <RETURN>
     int              Stack[0x4]:4   _Argc                          XREF[1]:     004
     char * *         Stack[0x8]:4   _Argv                          XREF[1]:     004
     char * *         Stack[0xc]:4   _Env
     undefined4       Stack[-0x1c]:4 local_1c                       XREF[1]:     004
     undefined4       Stack[-0x20]:4 local_20                       XREF[4]:     004
```

Decompile: _main - (trust.exe)

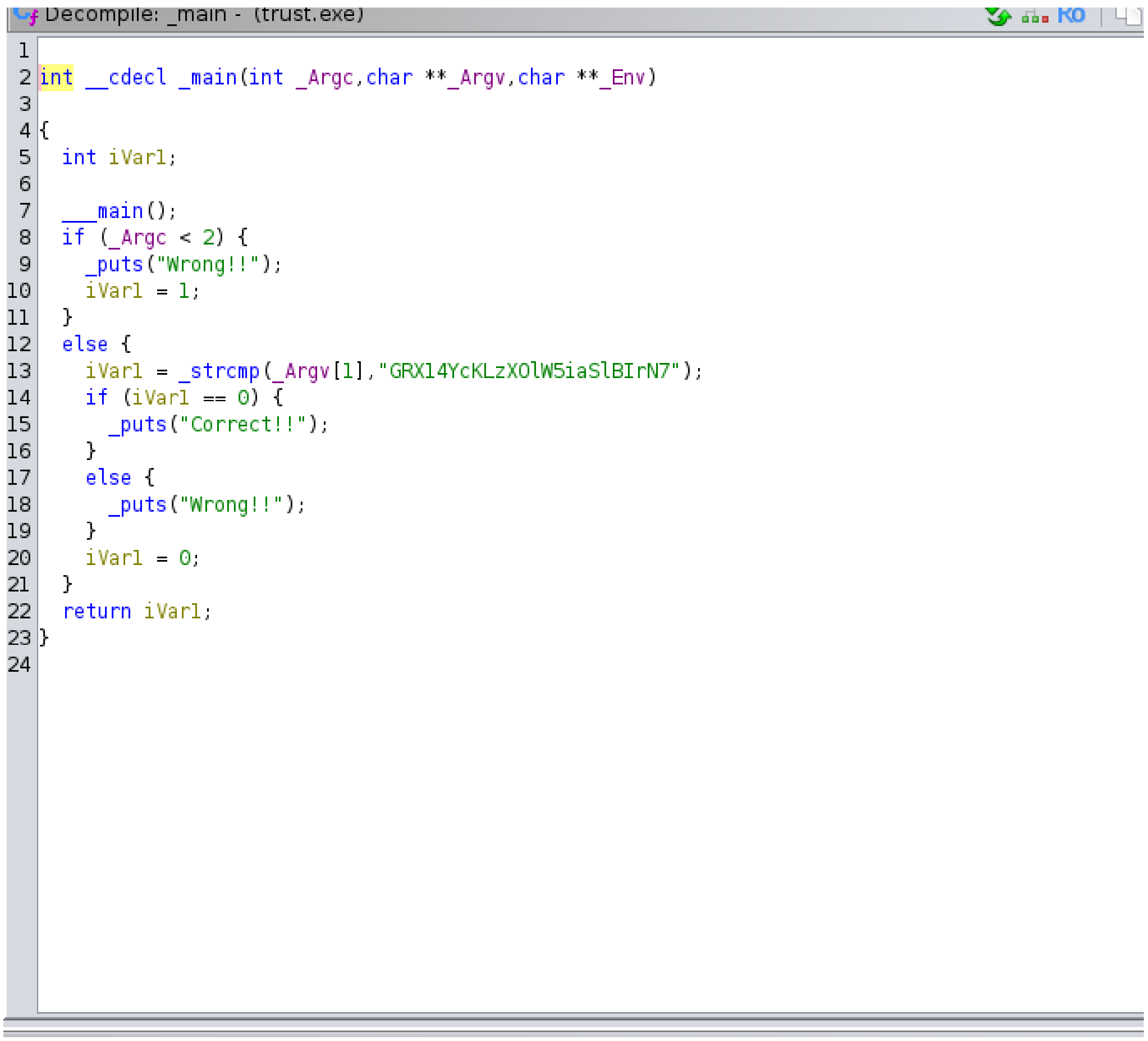
```c

int __cdecl _main(int _Argc,char **_Argv,char **_Env)

{
  int iVar1;
  
  ___main();
  if (_Argc < 2) {
    _puts("Wrong!!");
    iVar1 = 1;
  }
  else {
    iVar1 = _strcmp(_Argv[1],"GRX14YcKLzXOlW5iaSlBIrN7");
    if (iVar1 == 0) {
      _puts("Correct!!");
    }
    else {
      _puts("Wrong!!");
    }
    iVar1 = 0;
  }
  return iVar1;
}

```

The decompiler reveals a main function with a string. But it seems like the string is not the password.



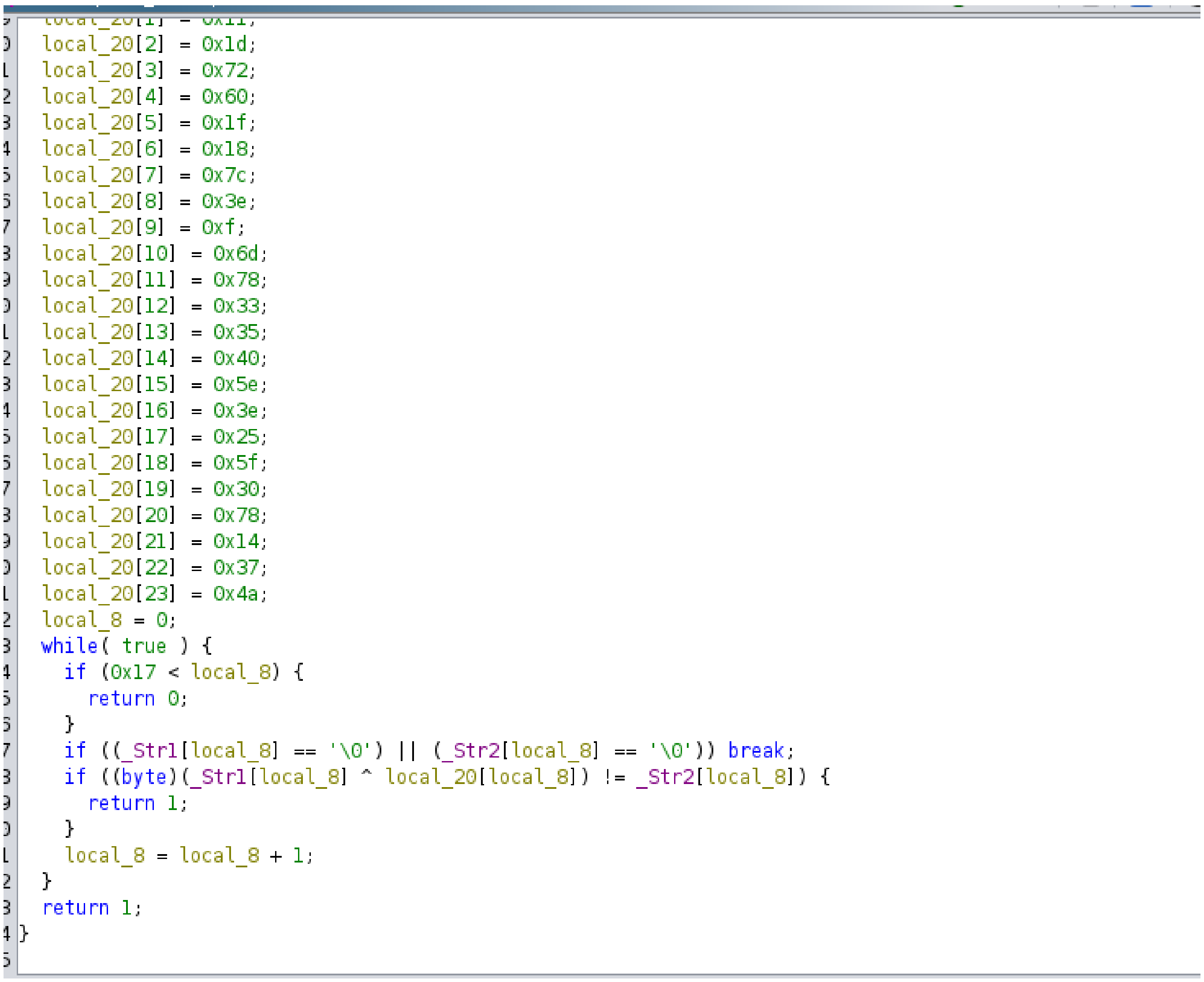
```c

int __cdecl _main(int _Argc,char **_Argv,char **_Env)

{
  int iVar1;
  
  ___main();
  if (_Argc < 2) {
    _puts("Wrong!!");
    iVar1 = 1;
  }
  else {
    iVar1 = _strcmp(_Argv[1],"GRX14YcKLzXOlW5iaSlBIrN7");
    if (iVar1 == 0) {
      _puts("Correct!!");
    }
    else {
      _puts("Wrong!!");
    }
    iVar1 = 0;
  }
  return iVar1;
}

```

So I poke around till I stumble upon the *strcmp* method is a custom one.



```
  local_20[1] = 0x11;
  local_20[2] = 0x1d;
  local_20[3] = 0x72;
  local_20[4] = 0x60;
  local_20[5] = 0x1f;
  local_20[6] = 0x18;
  local_20[7] = 0x7c;
  local_20[8] = 0x3e;
  local_20[9] = 0xf;
  local_20[10] = 0x6d;
  local_20[11] = 0x78;
  local_20[12] = 0x33;
  local_20[13] = 0x35;
  local_20[14] = 0x40;
  local_20[15] = 0x5e;
  local_20[16] = 0x3e;
  local_20[17] = 0x25;
  local_20[18] = 0x5f;
  local_20[19] = 0x30;
  local_20[20] = 0x78;
  local_20[21] = 0x14;
  local_20[22] = 0x37;
  local_20[23] = 0x4a;
  local_8 = 0;
  while( true ) {
    if (0x17 < local_8) {
      return 0;
    }
    if ((_Str1[local_8] == '\0') || (_Str2[local_8] == '\0')) break;
    if ((byte)(_Str1[local_8] ^ local_20[local_8]) != _Str2[local_8]) {
      return 1;
    }
    local_8 = local_8 + 1;
  }
  return 1;
}
```

1. The code has a hardcoded array of bytes (`local_20`), which serves as an encryption key.
2. It's comparing each character of `_Str2` (which is "GRX14YcKLzXOlW5iaSlBIrN7") with each character of `_Str1` (the input) XORed with the corresponding byte from `local_20` .

This code

```python
# Known comparison string
str2 = "GRX14YcKLzXOlW5iaSlBIrN7"

# XOR key from the local_20 array
xor_key = [0x06, 0x11, 0x1d, 0x72, 0x60, 0x1f, 0x18, 0x7c,
           0x3e, 0x0f, 0x6d, 0x78, 0x33, 0x35, 0x40, 0x5e,
           0x3e, 0x25, 0x5f, 0x30, 0x78, 0x14, 0x37, 0x4a]

# Calculate the password
password = ""
for i in range(len(str2)):
    if i < len(xor_key):
        # XOR each character with the corresponding key byte
        password += chr(ord(str2[i]) ^ xor_key[i])

print("Password to enter:", password)
```

Will perform the xor for us