Ciphertext CTF 2020

Reverse Engineering

The_old_snake

Description:

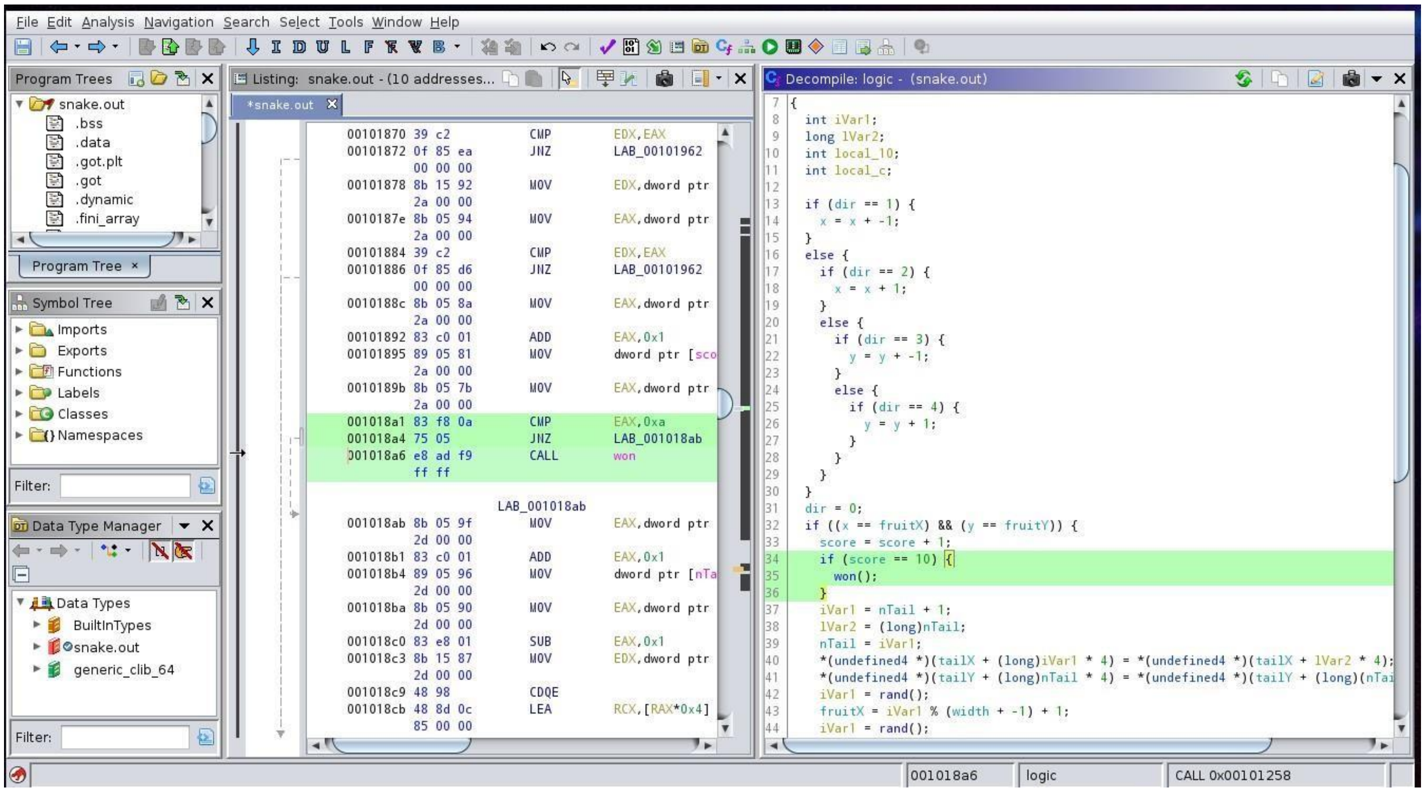
people say that it is impossible to win this game, can you disprove them?

Files:

snake.out size: 17.96 KB MD5: 9399de2090152ccfaa4226b214558c82

Solution:

When you open the executable in a decompiler (I used Ghidra), start browsing the code to understand what that program does, you can see that when your points reach 10 you will win (as shown below):

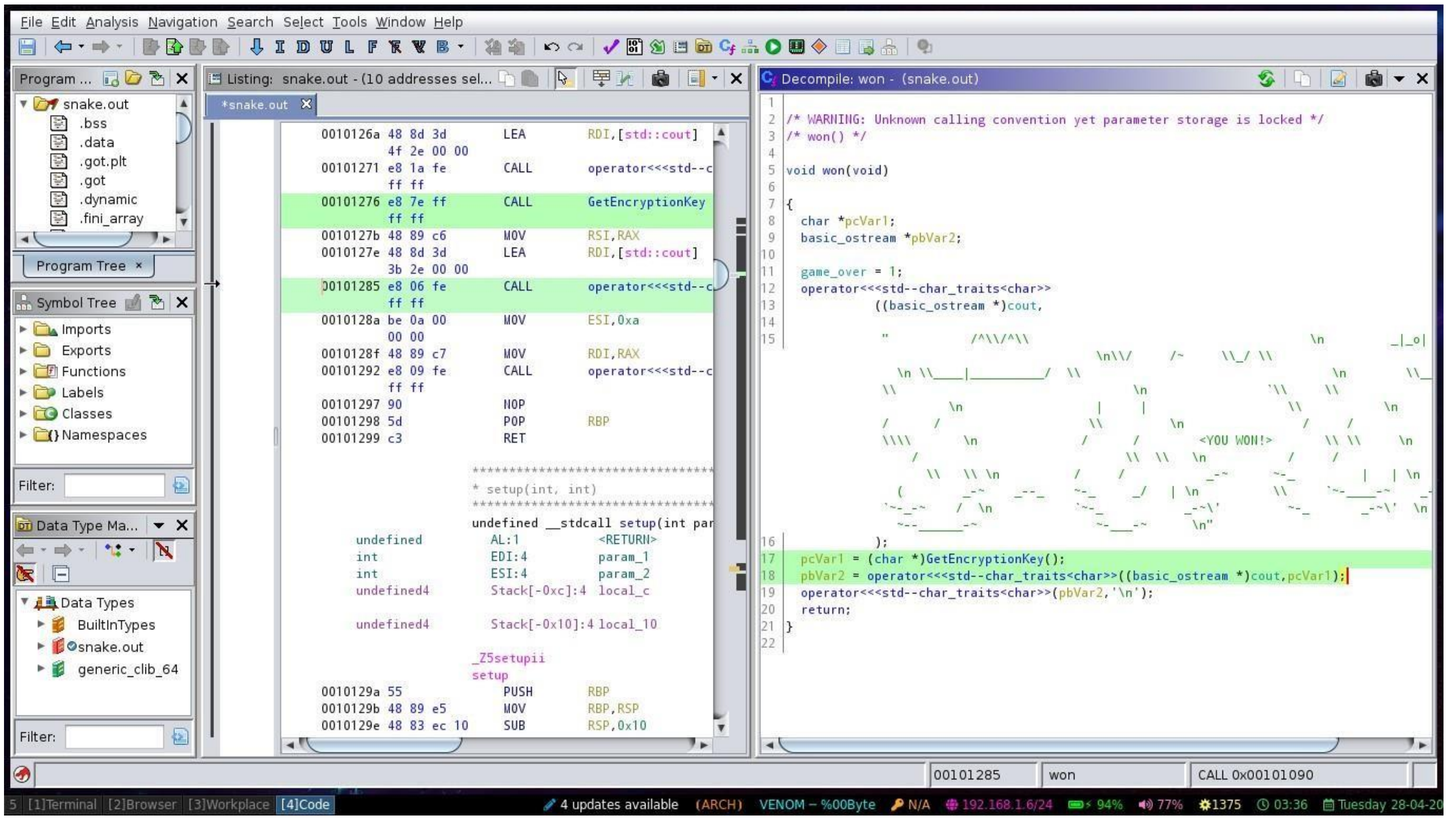


```
00101870 39 c2          CMP     EDX,EAX
00101872 0f 85 ea       JNZ     LAB_00101962
         00 00 00
00101878 8b 15 92       MOV     EDX,dword ptr
         2a 00 00
0010187e 8b 05 94       MOV     EAX,dword ptr
         2a 00 00
00101884 39 c2          CMP     EDX,EAX
00101886 0f 85 d6       JNZ     LAB_00101962
         00 00 00
0010188c 8b 05 8a       MOV     EAX,dword ptr
         2a 00 00
00101892 83 c0 01       ADD     EAX,0x1
00101895 89 05 81       MOV     dword ptr [sco
         2a 00 00
0010189b 8b 05 7b       MOV     EAX,dword ptr
         2a 00 00
001018a1 83 f8 0a       CMP     EAX,0xa
001018a4 75 05          JNZ     LAB_001018ab
001018a6 e8 ad f9       CALL    won
         ff ff

                    LAB_001018ab
001018ab 8b 05 9f       MOV     EAX,dword ptr
         2d 00 00
001018b1 83 c0 01       ADD     EAX,0x1
001018b4 89 05 96       MOV     dword ptr [nTa
         2d 00 00
001018ba 8b 05 90       MOV     EAX,dword ptr
         2d 00 00
001018c0 83 e8 01       SUB     EAX,0x1
001018c3 8b 15 87       MOV     EDX,dword ptr
         2d 00 00
001018c9 48 98          CDQE
001018cb 48 8d 0c       LEA     RCX,[RAX*0x4]
         85 00 00
```

```
{
  int iVar1;
  long lVar2;
  int local_10;
  int local_c;

  if (dir == 1) {
    x = x + -1;
  }
  else {
    if (dir == 2) {
      x = x + 1;
    }
    else {
      if (dir == 3) {
        y = y + -1;
      }
      else {
        if (dir == 4) {
          y = y + 1;
        }
      }
    }
  }
  dir = 0;
  if ((x == fruitX) && (y == fruitY)) {
    score = score + 1;
    if (score == 10) {
      won();
    }
    iVar1 = nTail + 1;
    lVar2 = (long)nTail;
    nTail = iVar1;
    *(undefined4 *)(tailX + (long)iVar1 * 4) = *(undefined4 *)(tailX + lVar2 * 4);
    *(undefined4 *)(tailY + (long)nTail * 4) = *(undefined4 *)(tailY + (long)(nTai
    iVar1 = rand();
    fruitX = iVar1 % (width + -1) + 1;
    iVar1 = rand();
```

```
001018a6    logic    CALL 0x00101258
```

So, what will happen if you win?



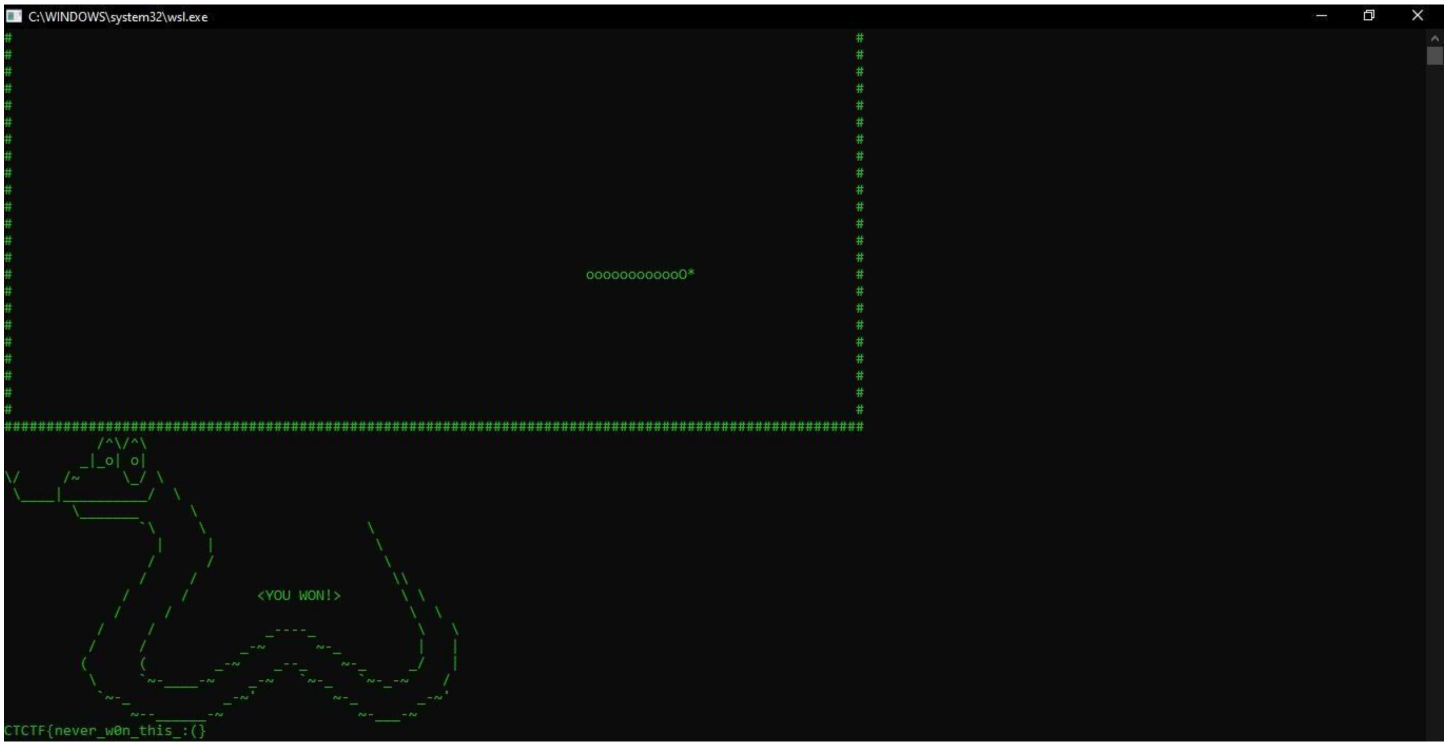
```
0010126a 48 8d 3d      LEA      RDI,[std::cout]
         4f 2e 00 00
00101271 e8 1a fe      CALL     operator<<<std--c
         ff ff
00101276 e8 7e ff      CALL     GetEncryptionKey
         ff ff
0010127b 48 89 c6      MOV      RSI,RAX
0010127e 48 8d 3d      LEA      RDI,[std::cout]
         3b 2e 00 00
00101285 e8 06 fe      CALL     operator<<<std--c
         ff ff
0010128a be 0a 00      MOV      ESI,0xa
         00 00
0010128f 48 89 c7      MOV      RDI,RAX
00101292 e8 09 fe      CALL     operator<<<std--c
         ff ff
00101297 90            NOP
00101298 5d            POP      RBP
00101299 c3            RET

                  **********************************
                  * setup(int, int)
                  **********************************
                  undefined __stdcall setup(int par
     undefined        AL:1            <RETURN>
     int              EDI:4           param_1
     int              ESI:4           param_2
     undefined4       Stack[-0xc]:4   local_c

     undefined4       Stack[-0x10]:4  local_10

                  _Z5setupii
                  setup
0010129a 55            PUSH     RBP
0010129b 48 89 e5      MOV      RBP,RSP
0010129e 48 83 ec 10   SUB      RSP,0x10
```

```c
/* WARNING: Unknown calling convention yet parameter storage is locked */
/* won() */

void won(void)

{
  char *pcVar1;
  basic_ostream *pbVar2;

  game_over = 1;
  operator<<<std--char_traits<char>>
            ((basic_ostream *)cout,
             "          /^\\/^\\                         \n          _|_o|
                          \n\\/     /~     \\_/ \\
             \n \\____|__________/  \\                       \n          \\_
             \\\\                     \n                     '\\\\      \\\\
                     \n              |     |                  \\\\          \n
             /      /                  \\\\         \n                  /     /
             \\\\\\\\        \n              /     /        <YOU WON!>      \\\\ \\\\     \n
                /                   \\\\  \\\\   \n                /     /
                  \\\\   \\\\ \n          /     /              _-~         ~-_          |   | \n
              (         _-~    _--_    ~-_     _/   | \n              \\          ~-_
             '~-_-~     /  \n            '~-_            _-~\\'        ~-_         _-~\\'  \n
               ~--______-~          ~-___-~       \n"
            );
  pcVar1 = (char *)GetEncryptionKey();
  pbVar2 = operator<<<std--char_traits<char>>((basic_ostream *)cout,pcVar1);
  operator<<<std--char_traits<char>>(pbVar2,'\n');
  return;
}
```

```
00101285    won    CALL 0x00101090
```

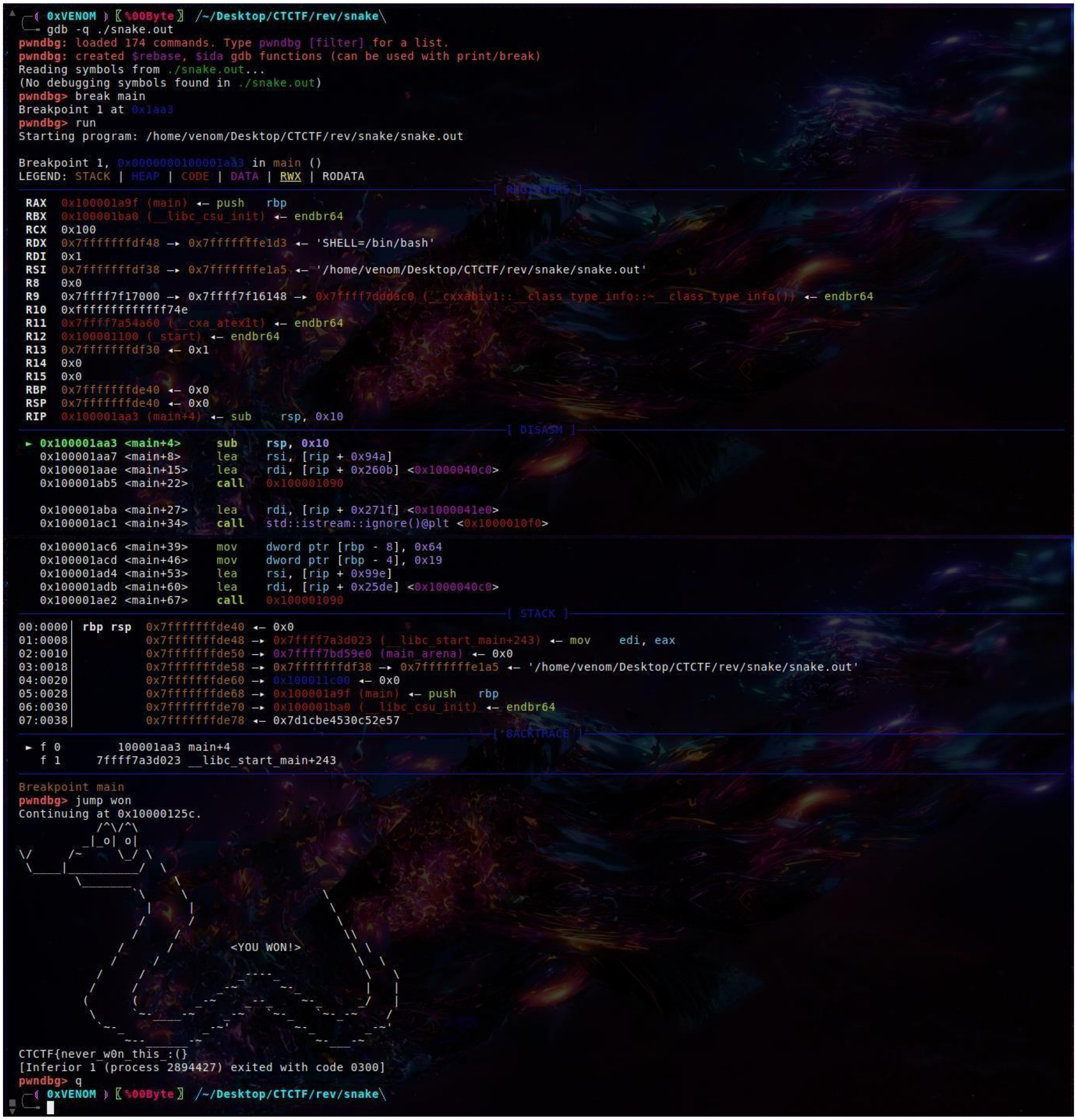
As we can see, an interesting function $GetEncryptionKey()$ is called, and its return value is printed out.

So, you can just play it, score 10 points, and you will get it printed out which is actually the flag.



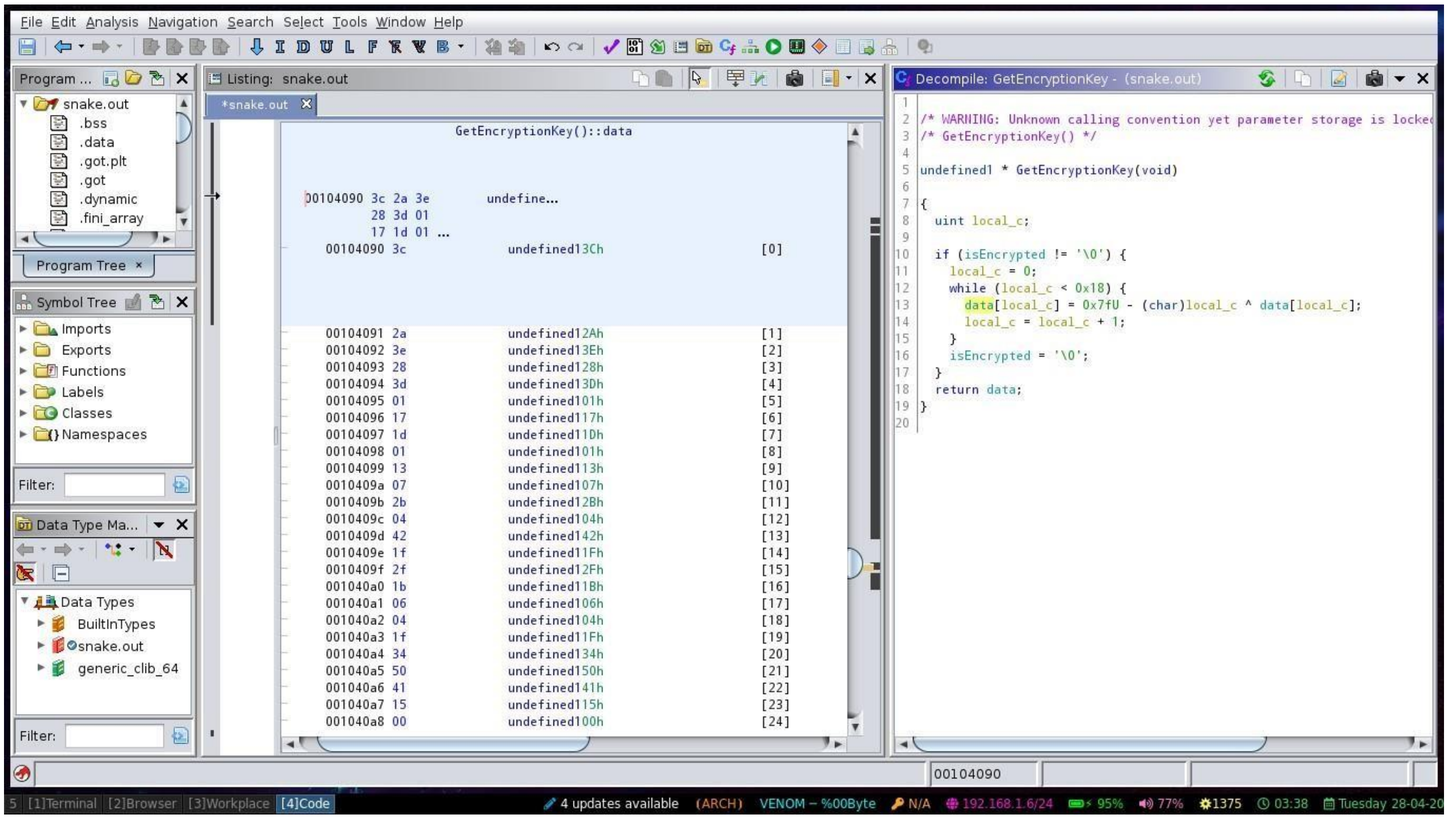
```
#                                                                     #
#                                                                     #
#                                               oooooooooooO*         #
#                                                                     #
#                                                                     #
#######################################################################
          /^\/^\
        _|_o| o|
\/     /~     \_/ \
 \____|__________/  \
        \_______      \
                `\     \                 \
                  |     |                  \
                 /      /                    \
                /     /                       \\
              /      /        <YOU WON!>        \ \
             /     /                             \  \
            /     /            _----_            \   \
           /     /          _-~      ~-_          |   |
          (      (        _-~    _--_    ~-_     _/   |
           \      `~-____-~    _-~    `~-_    `~-_-~   /
            `~-_           _-~'           ~-_       _-~'
                ~--______-~                 ~-___-~
CTCTF{never_w0n_this_:(}
```

If this way doesn't suit a hacker like you, then fire up you GDB and simply jump to $won()$ function:



```
┌─( 0xVENOM )〖%00Byte〗 /~/Desktop/CTCTF/rev/snake\
└─ gdb -q ./snake.out
pwndbg: loaded 174 commands. Type pwndbg [filter] for a list.
pwndbg: created $rebase, $ida gdb functions (can be used with print/break)
Reading symbols from ./snake.out...
(No debugging symbols found in ./snake.out)
pwndbg> break main
Breakpoint 1 at 0x1aa3
pwndbg> run
Starting program: /home/venom/Desktop/CTCTF/rev/snake/snake.out

Breakpoint 1, 0x0000000100001aa3 in main ()
LEGEND: STACK | HEAP | CODE | DATA | RWX | RODATA
──────────────────────────────────────────────────────[ REGISTERS ]──────
 RAX  0x100001a9f (main) ◂— push   rbp
 RBX  0x100001ba0 (__libc_csu_init) ◂— endbr64
 RCX  0x100
 RDX  0x7fffffffdf48 —▸ 0x7fffffffe1d3 ◂— 'SHELL=/bin/bash'
 RDI  0x1
 RSI  0x7fffffffdf38 —▸ 0x7fffffffe1a5 ◂— '/home/venom/Desktop/CTCTF/rev/snake/snake.out'
 R8   0x0
 R9   0x7ffff7f17000 —▸ 0x7ffff7f16148 —▸ 0x7ffff7dddac0 (__cxxabiv1::__class_type_info::~__class_type_info()) ◂— endbr64
 R10  0xfffffffffffff74e
 R11  0x7ffff7a54a60 (__cxa_atexit) ◂— endbr64
 R12  0x100001100 (_start) ◂— endbr64
 R13  0x7fffffffdf30 ◂— 0x1
 R14  0x0
 R15  0x0
 RBP  0x7fffffffde40 ◂— 0x0
 RSP  0x7fffffffde40 ◂— 0x0
 RIP  0x100001aa3 (main+4) ◂— sub    rsp, 0x10
───────────────────────────────────────────────────────[ DISASM ]──────
 ► 0x100001aa3 <main+4>     sub    rsp, 0x10
   0x100001aa7 <main+8>     lea    rsi, [rip + 0x94a]
   0x100001aae <main+15>    lea    rdi, [rip + 0x260b] <0x1000040c0>
   0x100001ab5 <main+22>    call   0x100001090

   0x100001aba <main+27>    lea    rdi, [rip + 0x271f] <0x1000041e0>
   0x100001ac1 <main+34>    call   std::istream::ignore()@plt <0x1000010f0>

   0x100001ac6 <main+39>    mov    dword ptr [rbp - 8], 0x64
   0x100001acd <main+46>    mov    dword ptr [rbp - 4], 0x19
   0x100001ad4 <main+53>    lea    rsi, [rip + 0x99e]
   0x100001adb <main+60>    lea    rdi, [rip + 0x25de] <0x1000040c0>
   0x100001ae2 <main+67>    call   0x100001090
───────────────────────────────────────────────────────[ STACK ]──────
00:0000│ rbp rsp  0x7fffffffde40 ◂— 0x0
01:0008│          0x7fffffffde48 —▸ 0x7ffff7a3d023 (__libc_start_main+243) ◂— mov    edi, eax
02:0010│          0x7fffffffde50 —▸ 0x7ffff7bd59e0 (main_arena) ◂— 0x0
03:0018│          0x7fffffffde58 —▸ 0x7fffffffdf38 —▸ 0x7fffffffe1a5 ◂— '/home/venom/Desktop/CTCTF/rev/snake/snake.out'
04:0020│          0x7fffffffde60 —▸ 0x100011c00 ◂— 0x0
05:0028│          0x7fffffffde68 —▸ 0x100001a9f (main) ◂— push   rbp
06:0030│          0x7fffffffde70 —▸ 0x100001ba0 (__libc_csu_init) ◂— endbr64
07:0038│          0x7fffffffde78 ◂— 0x7d1cbe4530c52e57
─────────────────────────────────────────────────────[ BACKTRACE ]──────
 ► f 0        100001aa3 main+4
   f 1     7ffff7a3d023 __libc_start_main+243
────────────────────────────────────────────────────────────────────────
Breakpoint main
pwndbg> jump won
Continuing at 0x10000125c.
          /^\/^\
        _|_o| o|
\/     /~     \_/ \
 \____|__________/  \
        \_______      \
                `\     \                 \
                  |     |                  \
                 /      /                    \
                /     /                       \\
              /      /        <YOU WON!>       \ \
             /     /                            \  \
           /     /             _----_            \   \
          /     /           _-~      ~-_         |   |
         (      (        _-~    _--_    ~-_     _/   |
          \      ~-____-~    _-~    ~-_    ~-_-~    /
            ~-_           _-~          ~-_       _-~
               ~--______-~                ~-___-~
CTCTF{never_w0n_this_:(}
[Inferior 1 (process 2894427) exited with code 0300]
pwndbg> q
┌─( 0xVENOM )〖%00Byte〗 /~/Desktop/CTCTF/rev/snake\
└─ █
```

If previous methods didn't satisfy you, then let's see what is happening inside the GetEncryptionKey() function:


File Edit Analysis Navigation Search Select Tools Window Help
Program ...
snake.out
.bss
.data
.got.plt
.got
.dynamic
.fini_array
Program Tree
Symbol Tree
Imports
Exports
Functions
Labels
Classes
Namespaces
Filter:
Data Type Ma...
Data Types
BuiltInTypes
snake.out
generic_clib_64
Filter:
Listing: snake.out
*snake.out
Decompile: GetEncryptionKey - (snake.out)
00104090
5 [1]Terminal [2]Browser [3]Workplace [4]Code
4 updates available (ARCH) VENOM – %00Byte N/A 192.168.1.6/24 95% 77% 1375 03:38 Tuesday 28-04-20

```
                    GetEncryptionKey()::data

00104090 3c 2a 3e        undefine…
         28 3d 01
         17 1d 01 …
   00104090 3c              undefined13Ch              [0]

   00104091 2a              undefined12Ah              [1]
   00104092 3e              undefined13Eh              [2]
   00104093 28              undefined128h              [3]
   00104094 3d              undefined13Dh              [4]
   00104095 01              undefined101h              [5]
   00104096 17              undefined117h              [6]
   00104097 1d              undefined11Dh              [7]
   00104098 01              undefined101h              [8]
   00104099 13              undefined113h              [9]
   0010409a 07              undefined107h              [10]
   0010409b 2b              undefined12Bh              [11]
   0010409c 04              undefined104h              [12]
   0010409d 42              undefined142h              [13]
   0010409e 1f              undefined11Fh              [14]
   0010409f 2f              undefined12Fh              [15]
   001040a0 1b              undefined11Bh              [16]
   001040a1 06              undefined106h              [17]
   001040a2 04              undefined104h              [18]
   001040a3 1f              undefined11Fh              [19]
   001040a4 34              undefined134h              [20]
   001040a5 50              undefined150h              [21]
   001040a6 41              undefined141h              [22]
   001040a7 15              undefined115h              [23]
   001040a8 00              undefined100h              [24]
```

```c

/* WARNING: Unknown calling convention yet parameter storage is locke
/* GetEncryptionKey() */

undefined1 * GetEncryptionKey(void)

{
  uint local_c;

  if (isEncrypted != '\0') {
    local_c = 0;
    while (local_c < 0x18) {
      data[local_c] = 0x7fU - (char)local_c ^ data[local_c];
      local_c = local_c + 1;
    }
    isEncrypted = '\0';
  }
  return data;
}

```

Fortunately, it is not that complicated, and we can see that it uses a variable called $data$ which has the actual data encrypted. When the function is called it iterates through that data and apply $data[i] = 0x7f - (i \hat{} data[i])$ for each $0 \leq i < 0x18$ then returns $data$ which contains the decrypted flag and it will be printed out after returning, so let's get the data and decrypt it manually, in this case we can solve it without even running the executable:

```python
#!/bin/python3 data =
[0x3C,0x2A,0x3E,0x28,0x3D,0x01,0x17,0x1D,0x01,0x13,0x07,0x2B,0x04,0x42,0x1F,0x2F,0x1B,
0x06,0x04,0x1F,0x34,0x50,0x41,0x15] dec_data
= ''   i = 0
while i <
len(data):
    dec_data += chr(0x7f - (i ^
data[i]))     i += 1   print(dec_data)
```

Output: CTCTF{never_w0n_this_:(}