ctfd-plus

Let's look at the decompiled output.

Our input gets compared with the result of somefunction(buf[i]), character by character.

```
  do {
    cVar1 = somefunction(buf[i]);
    if (cVar1 != userinput[i]) {
      puts("Incorrect flag.");
      return 0;
    }
    i = i + 1;
  } while (i != 47);
  puts("You got the flag! Unfortunately we don\'t exactly have a database to store the solve in...")
  ;

int somefunction(uint buf1char)

{
  uint uVar1;
  int i;
  byte x;

  uVar1 = 0;
  i = 0;
  do {
    x = (byte)i & 0x1f;
    i = i + 1;
    buf1char = (buf1char * buf1char >> x | buf1char * buf1char << 32 - x) * 0x1337 + 0x4201337 ^
               uVar1;
    uVar1 = uVar1 + 0x13371337;
  } while (i != 32);
  return (buf1char >> 8) + (buf1char >> 16) + buf1char + (buf1char >> 24);
}
```

We don't really have to understand the logic of the function. We can just use GDB and set a breakpoint right after the call returns and stops at the CMP operation. The result of the function call is stored in RAX, but we are only interested in the lower byte AL.

```
Breakpoint 6, 0x000055555555510b in ?? ()
LEGEND: STACK | HEAP | CODE | DATA | RWX | RODATA
──────────────────────────────[ REGISTERS / show-flags off / show-compact-regs off ]──────────────────────────────
*RAX  0xe900e26c
*RBX  0x7fffffffdcb0 ◂— 0x74736574 /* 'test' */
*RCX  0x20
*RDX  0xe817
*RDI  0xe8
*RSI  0x0
*R8   0x555555558060 ◂— 0xc2bb0ce99c7f9274
*R9   0x0
*R10  0x7ffff7dddd68 ◂— 0x10001a000008b5
*R11  0x7ffff7f3d030 (__strcspn_sse42) ◂— cmp byte ptr [rsi], 0
 R12  0x0
 R13  0x7fffffffded8 —▸ 0x7fffffffe259 ◂— 'COLORFGBG=15;0'
 R14  0x0
 R15  0x7ffff7ffd020 (_rtld_global) —▸ 0x7ffff7ffe2e0 —▸ 0x555555554000 ◂— 0x10102464c457f
 RBP  0x1
 RSP  0x7fffffffdcb0 ◂— 0x74736574 /* 'test' */
*RIP  0x55555555510b ◂— cmp al, byte ptr [rbx + rsi]
```

The first character returned from the function call is (0x6c) 'l'.
Use 'l' as the first character for our input in the next run.

```
*RAX  0×955ddd61
 RBX  0×7fffffffdcb0 ◂— 'lqwerqwer'
 RCX  0×20
*RDX  0×94c8
*RDI  0×94
*RSI  0×1
 R8   0×555555558060 ◂— 0×c2bb0ce99c7f9274
 R9   0×0
 R10  0×7ffff7dddd68 ◂— 0×10001a000008b5
 R11  0×7ffff7f3d030 (__strcspn_sse42) ◂— cmp byte ptr [rsi], 0
 R12  0×0
 R13  0×7fffffffded8 —▸ 0×7fffffffe259 ◂— 'COLORFGBG=15;0'
 R14  0×0
 R15  0×7ffff7ffd020 (_rtld_global) —▸ 0×7ffff7ffe2e0 —▸ 0×555555554000 ◂— 0×10102464c457f
 RBP  0×1
 RSP  0×7fffffffdcb0 ◂— 'lqwerqwer'
 RIP  0×55555555510b ◂— cmp al, byte ptr [rbx + rsi]
```

The second compared byte is 0x61 (a).
Concatenate ‘a’ as our second character for our input in the next run.

Repeat the process until we get the flag.

```
└─$ ./ctfd_plus
Welcome to CTFd+!
So far, we only have one challenge, which is one more than the number of databases we have

Very Doable Pwn - 500 points, 0 solves
Can you help me pwn this program?
#include <stdio.h>
int main(void) {
    puts("Bye!");
    return 0;
}

Enter the flag:
lactf{m4yb3_th3r3_1s_s0m3_m3r1t_t0_us1ng_4_db}
You got the flag! Unfortunately we don't exactly have a database to store the solve in...
```