./level13

A critical observation in the code is the dependency on the getuid() function to yield 0x1092 (or 4242). To gain a more granular understanding, an assembly level inspection was conducted.

```c
void main(void)
{
    __uid_t uid;
    char *token;

    uid = getuid();
    if (uid != 0x1092)
    {
        uid = getuid();
        printf("UID %d started us but we we expect %d\n", uid, 0x1092);
        exit(1);
    }
    token = ft_des("boe]!ai0FB@.:|L6l@A?>qJ}I");
    printf("your token is %s\n", token);
    return;
}
```

As in many previous levels, upon accessing the level13 user's home directory, we found an executable. To glean insights into its inner workings, we opted again for Ghidra.

```
(gdb) disas main
Dump of assembler code for function main:
   0x0804858c <+0>:     push   %ebp
   0x0804858d <+1>:     mov    %esp,%ebp
   0x0804858f <+3>:     and    $0xfffff0,%esp
   0x08048592 <+6>:     sub    $0x10,%esp
   0x08048595 <+9>:     call   0x8048380 <getuid@plt>
   0x0804859a <+14>:    cmp    $0x1092,%eax           <<<<<<<<<<<<<<<<<<<
   0x0804859f <+19>:    je     0x80485cb <main+63>
   0x080485a1 <+21>:    call   0x8048380 <getuid@plt>
   0x080485a6 <+26>:    mov    $0x80486c8,%edx
   0x080485ab <+31>:    movl   $0x1092,0x8(%esp)
   0x080485b3 <+39>:    mov    %eax,0x4(%esp)
   0x080485b7 <+43>:    mov    %edx,(%esp)
   0x080485ba <+46>:    call   0x8048360 <printf@plt>
   0x080485bf <+51>:    movl   $0x1,(%esp)
   0x080485c6 <+58>:    call   0x80483a0 <exit@plt>
   0x080485cb <+63>:    movl   $0x80486ef,(%esp)
   0x080485d2 <+70>:    call   0x8048474 <ft_des>
   0x080485d7 <+75>:    mov    $0x8048709,%edx
   0x080485dc <+80>:    mov    %eax,0x4(%esp)
   0x080485e0 <+84>:    mov    %edx,(%esp)
   0x080485e3 <+87>:    call   0x8048360 <printf@plt>
   0x080485e8 <+92>:    leave
   0x080485e9 <+93>:    ret
End of assembler dump.
```

The assembly instruction at *0x0804859a* compares the value present in the eax register to *0x1092*. Given the capabilities of GDB, we can actively manipulate the register values during runtime. By setting a breakpoint at the previously mentioned instruction, modifying the "*eax*" register, and then proceeding with the execution, we can effectively bypass the conditional check, circumventing the undesired *exit* call.

```
level13@SnowCrash:~$ gdb level13
(gdb) break *0x804859a
Breakpoint 1 at 0x804859a
(gdb) run
Starting program: /home/user/level13/level13

Breakpoint 1, 0x0804859a in main ()
(gdb) frame
#0  0x0804859a in main ()
(gdb) print $eax=0x1092
$1 = 4242
(gdb) continue
Continuing.
your token is 2A31L79asukciNyi8uppkEuSx
[Inferior 1 (process 2979) exited with code 050]
(gdb) q

level13@SnowCrash:~$ su level14
Password: 2A31L79asukciNyi8uppkEuSx

level14@SnowCrash:~$
```