


| level14

In this level, after a really long search, we found out that we have nothing to work on. The only thing left for us to exploit is the `getflag` executable, so let's look at this, with *Ghidra*.



```
int main(void)
{
...
    ptraceResult = ptrace(PTRACE_TRACEME, 0, 1, 0);
    if (ptraceResult < 0) {
        puts("You should not reverse this");
        returnValue = 1;
    }
    else {
...
        uid = getuid();
...
        else if (uid == 3013) {
            token = (char *)ft_des("boe]!ai0FB@.:|L6l@A?>qJ}I");
            fputs(token, __stream);
        }
        else {
            if (uid != 3014) goto LAB_08048e06;
            token = (char *)ft_des("g <t61:|4_|!@IF.-62FH&G~DCK/Ekrvvdwz?v|»);
            fputs(token, __stream);
        }
...
        return returnValue;
    }
}
```

The *main* function is quite extensive and encompasses various operations. We've condensed it down, spotlighting only the key segments. Here's what's interesting:

```
ptraceResult = ptrace(PTRACE_TRACEME, 0, 1, 0);
if (ptraceResult < 0) {
    puts("You should not reverse this");
    returnValue = 1;
}
```

The code tries to stop debuggers like GDB from working. If it finds one, *ptrace* returns -1. So, we need to get around this.

```
if (uid != 3014) goto LAB_08048e06;
token = (char *)ft_des(«#hash»);
fputs(token, __stream);
```

Then, there's one last important "if" check. If the user ID is 3014 (which belongs to *flag14*), we can get access to his token.

|level14²

Alright, we know what to do. Let's figure out how to make it happen.

```
08048989 e8 b2 fb ff ff CALL <EXTERNAL>::ptrace
0804898e 85 c0 TEST EAX,EAX
08048990 79 16 JNS LAB_080489a8
```

The *ptrace* function call is followed by the assembly instruction *TEST EAX,EAX*.

This instruction checks if the value in the *EAX* register is zero by executing a bitwise *AND* operation on itself, effectively assessing the return value of *ptrace*. To bypass this check, our goal is to ensure *EAX* is set to zero after the *ptrace* call.

```
08048bb6 3d c6 0b 00 00 CMP uid,3014
08048bbb 0f 84 24 02 00 JZ LAB_08048de5
```

Next up, we have the *CMP* instruction which compares the value in the *EAX* register to the constant *0x00000bc6* (which is *3014* in decimal). To manipulate the result of this comparison, we need to modify the *EAX* value after the *getuid()* call. So let's execute all that:

```
level14@SnowCrash:~$ gdb getflag
(gdb) break *0x804898e // TEST EAX, EAX
Breakpoint 1 at 0x804898e
(gdb) run
Starting program: /bin/getflag

Breakpoint 1, 0x804898e in main ()
(gdb) print $eax
$1 = -1 // PTRACE return value
(gdb) print $eax=0
$2 = 0
(gdb) break getuid
Breakpoint 2 at 0xb7ee4cc0
(gdb) continue
Continuing.

Breakpoint 2, 0xb7ee4cc0 in getuid () from /lib/i386-linux-gnu/libc.so.6
(gdb) si // Step next instruction
0xb7fdd418 in __kernel_vsyscall () // Step out of syscall
...
0x08048b02 in main () // We are just before EAX get put in uid
(gdb) print $eax
$7 = 2014
(gdb) $eax=3014
(gdb) continue
Continuing.
Check flag.Here is your token : 7QiHafiNa3HVozsaXkawuYrTstxbpABHD8CPnHJ
[Inferior 1 (process 336) exited normally]
(gdb) q
level14@SnowCrash:~$ su flag14
Password: 7QiHafiNa3HVozsaXkawuYrTstxbpABHD8CPnHJ
Congratulation. Type getflag to get the key and send it to me the owner of this livedcd :)
```