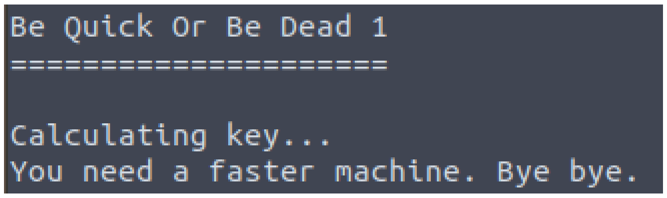
Reverse 5: Be Quick Or Be Dead 1

Issue: IDA doesn't allow me to modify as I want the instructions. For example, I would like to increase the timer to 8 (as proposed in the solution) but, however, I can't and I receive an error. Is it due to the fact that I do not use the pro version?

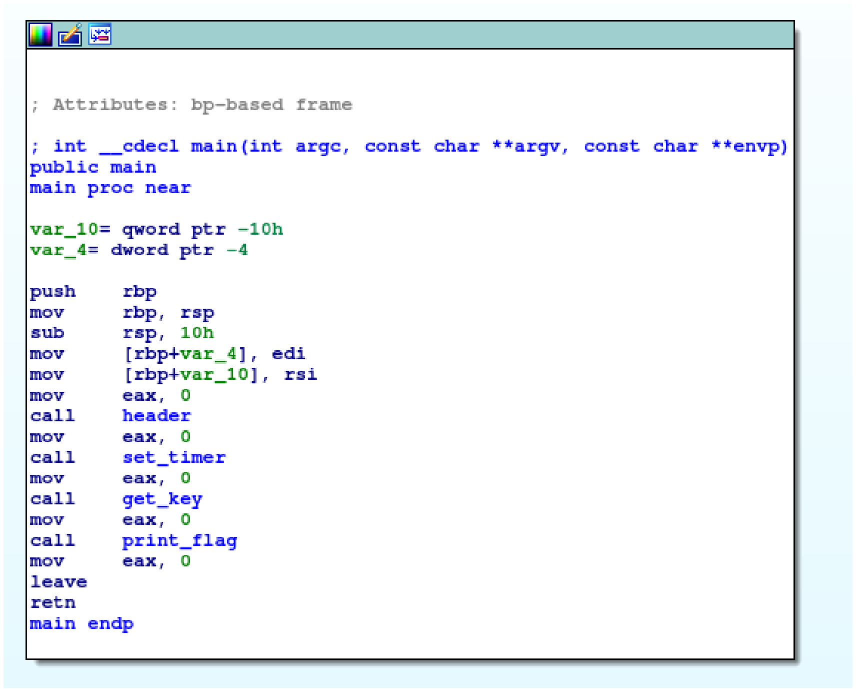
The description says "my machine is too slow for executing the program and reach the flag". At this point, I have no idea of what it means.

We can execute the program.



```
Be Quick Or Be Dead 1
=====================

Calculating key...
You need a faster machine. Bye bye.
```

We cannot neither insert / interact with the program. Based on our "big" set of tools (objdump - strings - IDA), a proper guess is that we need to "help" the flow execution of the program. Let's open the file with IDA.

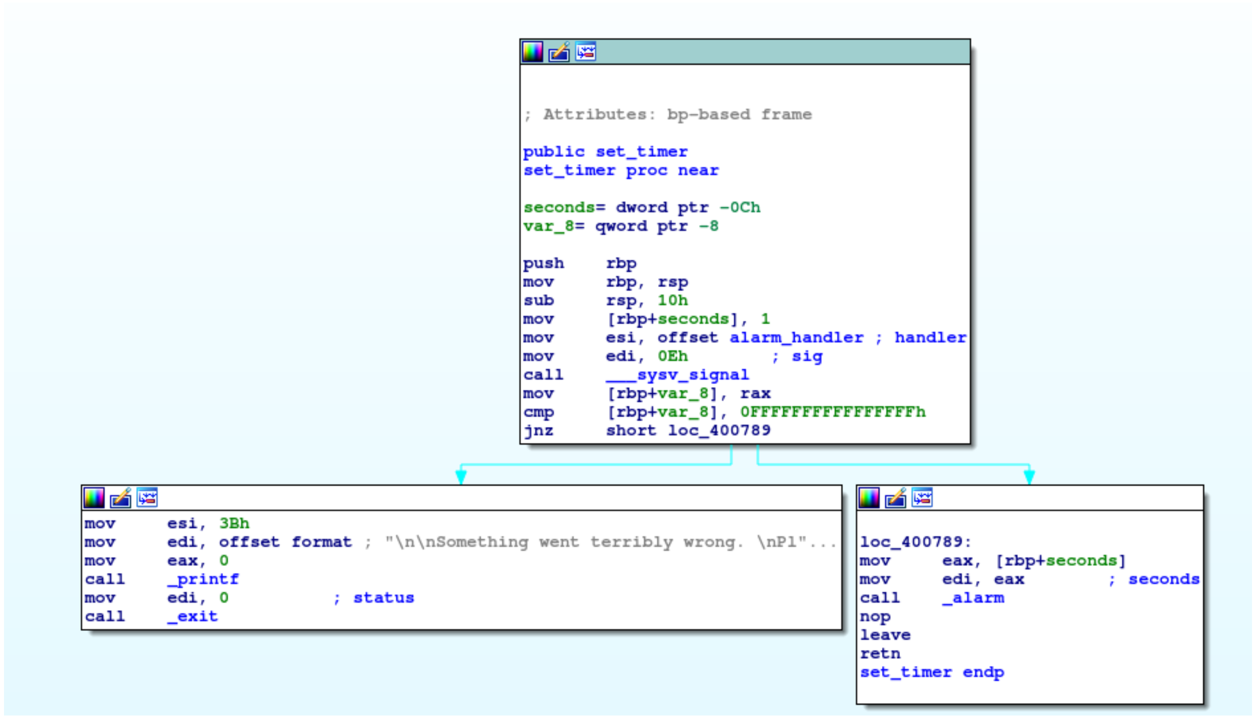


```
; Attributes: bp-based frame

; int __cdecl main(int argc, const char **argv, const char **envp)
public main
main proc near

var_10= qword ptr -10h
var_4= dword ptr -4

push    rbp
mov     rbp, rsp
sub     rsp, 10h
mov     [rbp+var_4], edi
mov     [rbp+var_10], rsi
mov     eax, 0
call    header
mov     eax, 0
call    set_timer
mov     eax, 0
call    get_key
mov     eax, 0
call    print_flag
mov     eax, 0
leave
retn
main endp
```

The program seems "naive". Some functions are called, such as "set timer". Two functions seem interesting:

- Set timer
- Get key

Double click on “set_timer” and we’ll see its assembly:

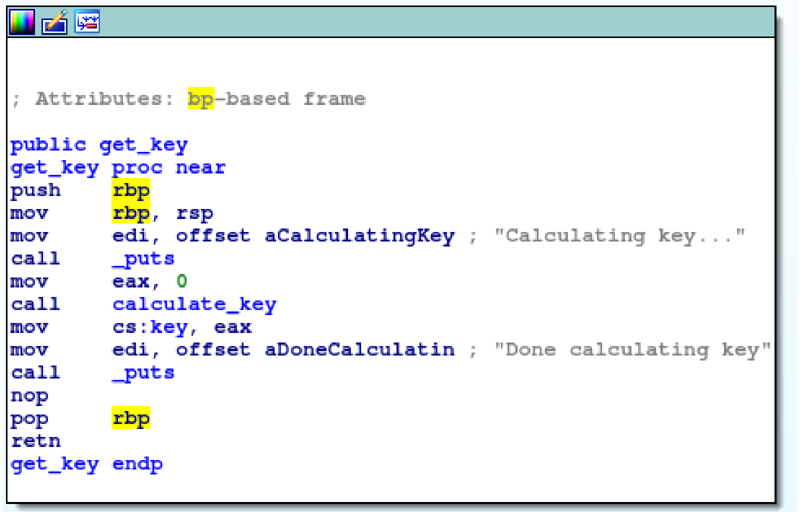


```
; Attributes: bp-based frame

public set_timer
set_timer proc near

seconds= dword ptr -0Ch
var_8= qword ptr -8

push    rbp
mov     rbp, rsp
sub     rsp, 10h
mov     [rbp+seconds], 1
mov     esi, offset alarm_handler ; handler
mov     edi, 0Eh        ; sig
call    ___sysv_signal
mov     [rbp+var_8], rax
cmp     [rbp+var_8], 0FFFFFFFFFFFFFFFFh
jnz     short loc_400789
```

```
mov     esi, 3Bh
mov     edi, offset format ; "\n\nSomething went terribly wrong. \nPl"...
mov     eax, 0
call    _printf
mov     edi, 0          ; status
call    _exit
```

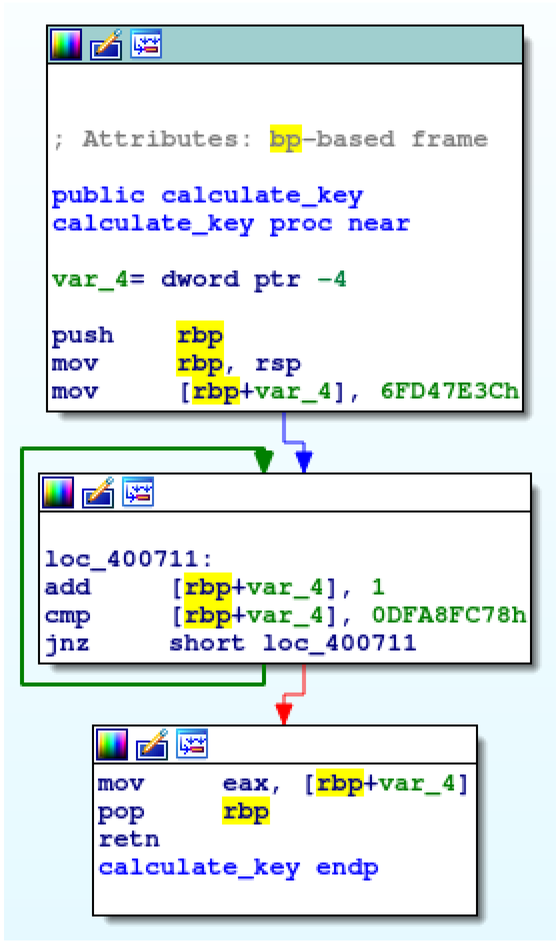
```
loc_400789:
mov     eax, [rbp+seconds]
mov     edi, eax        ; seconds
call    _alarm
nop
leave
retn
set_timer endp
```

This function set an alarm to 1 second and when it goes to 0 the program’s execution will terminate. Let’s move to *get key*:



```
; Attributes: bp-based frame

public get_key
get_key proc near
push    rbp
mov     rbp, rsp
mov     edi, offset aCalculatingKey ; "Calculating key..."
call    _puts
mov     eax, 0
call    calculate_key
mov     cs:key, eax
mov     edi, offset aDoneCalculatin ; "Done calculating key"
call    _puts
nop
pop     rbp
retn
get_key endp
```

This function calculates a key with *calculate_key* … nothing more. Let’s study also *calculate_key*.



```
; Attributes: bp-based frame

public calculate_key
calculate_key proc near

var_4= dword ptr -4

push    rbp
mov     rbp, rsp
mov     [rbp+var_4], 6FD47E3Ch
```

```
loc_400711:
add     [rbp+var_4], 1
cmp     [rbp+var_4], 0DFA8FC78h
jnz     short loc_400711
```

```
mov     eax, [rbp+var_4]
pop     rbp
retn
calculate_key endp
```

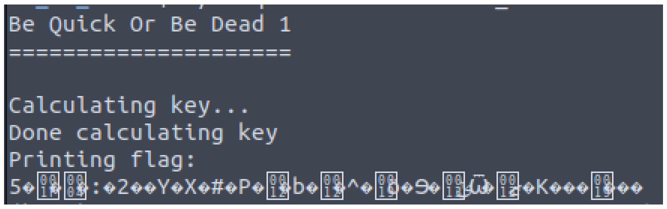
The second block is a clear *while* loop, where a variable is incremented until a certain amount is reached. If we see also the function *print_flag*, the last one, no *cmp* or additional *loops* are defined.

Well, the description talk about issues with the time of execution. A possible explanation is that the alarm stops the program's execution before the print_flag is called, due to the loop.

With IDA we can modify the program and try to reach the flag … for example we can:

- Increase the amount of seconds of the alarm;
- Generate the key immediately, by changing the *jnz* with *jz*;
- There are tons of other possibilities.

I'll try by replacing the *jnz* with *jz* in the loop.



```
Be Quick Or Be Dead 1
=====================

Calculating key...
Done calculating key
Printing flag:
5�[illegible]:�2��Y�X�#�P�[illegible]b�[illegible]^�[illegible]�9�[illegible]�[illegible]�K���[illegible]��
```

We reached the flag .. but it is unreadable! Maybe we need to reach the proper key in order to have the flag .. it makes sense, since *print_flag* uses *decrypt_flag* (we know that the encryption / decryption algorithms must use correct key).

I can try with my first hypothesis: increase the seconds of the alarm; let’s try with 60 seconds (hopefully it is enough).