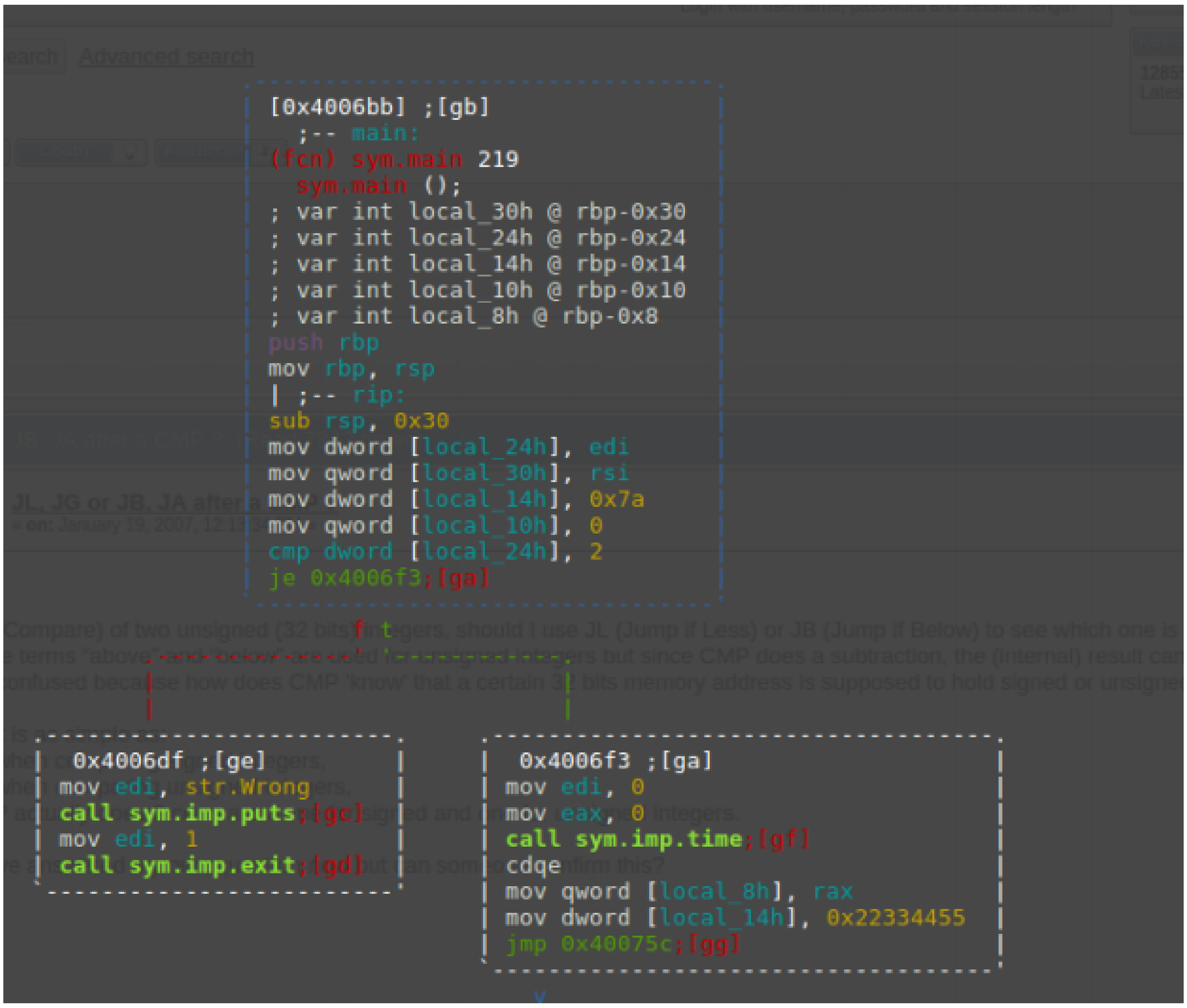
I am using radare to solve this one ! It was time to learn a new tool.

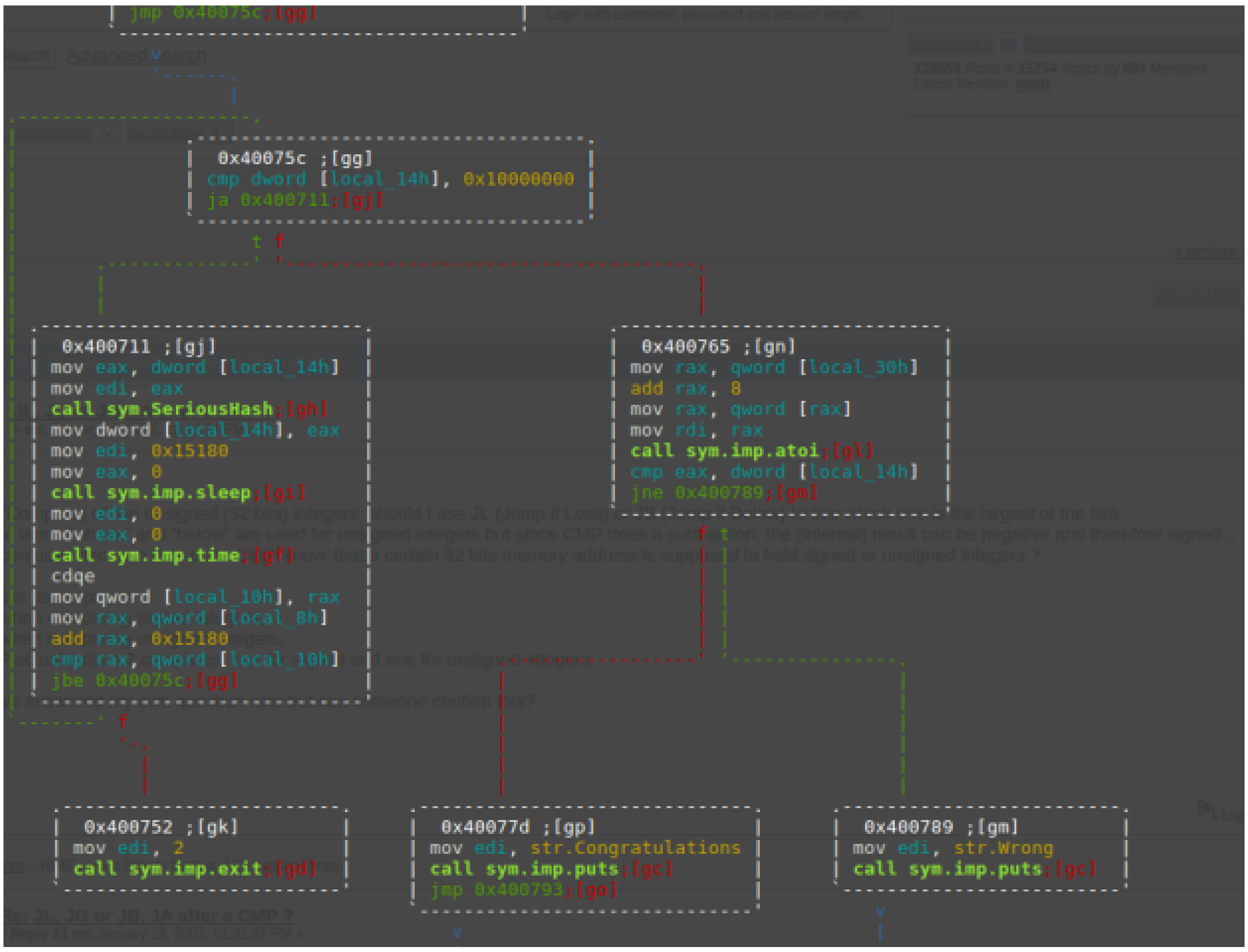


```
[0x4006bb] ;[gb]
  ;-- main:
(fcn) sym.main 219
  sym.main ();
; var int local_30h @ rbp-0x30
; var int local_24h @ rbp-0x24
; var int local_14h @ rbp-0x14
; var int local_10h @ rbp-0x10
; var int local_8h @ rbp-0x8
push rbp
mov rbp, rsp
| ;-- rip:
sub rsp, 0x30
mov dword [local_24h], edi
mov qword [local_30h], rsi
mov dword [local_14h], 0x7a
mov qword [local_10h], 0
cmp dword [local_24h], 2
je 0x4006f3;[ga]
```

```
0x4006df ;[ge]
mov edi, str.Wrong
call sym.imp.puts;[gc]
mov edi, 1
call sym.imp.exit;[gd]
```

```
0x4006f3 ;[ga]
mov edi, 0
mov eax, 0
call sym.imp.time;[gf]
cdqe
mov qword [local_8h], rax
mov dword [local_14h], 0x22334455
jmp 0x40075c;[gg]
```

The program calls function time() and stores the result returned in local_8h. It then stores the value 0x22334455 in the local variable called local_14h



```
jmp 0x40075c;[gg]
```

```
0x40075c ;[gg]
cmp dword [local_14h], 0x10000000
ja 0x400711;[gj]
```

```
0x400711 ;[gj]
mov eax, dword [local_14h]
mov edi, eax
call sym.SeriousHash;[gh]
mov dword [local_14h], eax
mov edi, 0x15180
mov eax, 0
call sym.imp.sleep;[gi]
mov edi, 0
mov eax, 0
call sym.imp.time;[gf]
cdqe
mov qword [local_10h], rax
mov rax, qword [local_8h]
add rax, 0x15180
cmp rax, qword [local_10h]
jbe 0x40075c;[gg]
```

```
0x400765 ;[gn]
mov rax, qword [local_30h]
add rax, 8
mov rax, qword [rax]
mov rdi, rax
call sym.imp.atoi;[gl]
cmp eax, dword [local_14h]
jne 0x400789;[gm]
```

```
0x400752 ;[gk]
mov edi, 2
call sym.imp.exit;[gd]
```

```
0x40077d ;[gp]
mov edi, str.Congratulations
call sym.imp.puts;[gc]
jmp 0x400793;[go]
```

```
0x400789 ;[gm]
mov edi, str.Wrong
call sym.imp.puts;[gc]
```

It compares the local_14h with 0x10000000 and if local_14h > 0x10000000 it jumps to 0x400711. Here the first time this instruction is executed the condition of the jump is true. This is in order to get to the function serious hash. The program calls serioushash() then stores the result returned by the function (by convention it's in the eax register) in the local_14h It then sleeps for a time 0x15180 (or 86400 seconds ie it sleeps for a day). It then stores the value returned by time() in in local_10h and compares it with the value returned by first call of time() (stored in local_8h) to which it adds the sleep time(0x15180). This is probably to verify that there are no breakpoints and that the program is not being debugged. If it's not the case, meaning that the time in local_10h is not superior to the time in local_8h + sleeptime then it exists. It does this many times until the hash returned is inferior to 0x10000000.Only then, the program enters the right branch (0x400765). Here the program calls C function atoi() in order to return the integer value of our entered key (stored in local_30h) and compares it with the content of local_14h which contains the hash of the right key. If it's equal it prints congratulations. I can conclude 3 things : First that the key is a number and second that local_30h contains the key that I entered and finally that the program keeps hashing the return value of serioushash applied the first time to 0x22334455 until it is inferior to 0x10000000. The question now is how to get the content of local_14h (ie the hash of 0x22334455) ?

There are many ways to do this : Either put a breakpoint just after the call to the function serioushash() and look at the registers to see the value of eax register or put another breakpoint after the atoi() call in the compare instruction and look at the content of local_14h and the most obvious and boring way is to look at the serioushash() function itself and try to figure out what it does but this would take too long since the program enters serioushash() more than one time. So in order to determine the right value of the hash which is inferior to 0x10000000 I put 3 breakpoints : The first at 0x0040071b just after the serioushash() call in order to look at the value of the hash stored in the rax register. The second is at 0x00400723 right before the sleep() call : this is in order to set the value of rdi to 0 so that the programs sleeps for 0 seconds ie skips the sleep(). In radare we can do this by the command dr rdi=0 Finally the 3rd breakpoint is at address 0x0040074c where I set the value of rax to 0 so that the jbe instruction is true. (dr rax =0) I do this many times until the value returned is inferior to 0x10000000. The values of the hashes returned by serioushash() are: 0xced700e1 ; 0x9309160a ; 0x4bd6bba5 ; 0x93989810 ; 0x9ae1f75b ; 0x98b37c27 ;0x208922c4 0x8b54a502 ; 0x13436e68 ; 0x4f9ea950 and 0x05c0cbd0 0x05c0cbd0 is the first hash value inferior to 0x10000000 which is our key. In decimal it’s 96521168. Hallelujah !