

Module Description

- Some evil entity has put a binary bomb on your workstation. You're not quite sure what it is, but it has "bomb" in the name, so it has to be serious!
- Take on the challenge of defusing it and save everybody! Or well, at least your PC...

Binary Bomb

- There is a binary file named **bomb** located in your "/home/student/Desktop" directory. Your task is to reverse engineer it and get the flag without setting it up.

Note: Watch out! The executable will self-destruct if the bomb explodes!

→ Write all the local variables addresses on a different text file!

Observations:

1. Allocates 800 bytes for the main function.
2. There are four circumstances that the bomb will explode. I think I have to create a reason to use the conditional jumps to avoid the "call explode" instructions until I reach the last instruction in the main function.
3. "some_eight_letter_pass" is the password for the first one.
4. Length of the pass_2 is 14.
5. The second key requires **three integer** inputs.
6. When using Cutter, the bomb does NOT detonate when you restart it!

Note: In Cutter, you have to ENABLE the "console" at the beginning to interact with the ELF file if it need inputs! (Windows→Console enable THEN start debugging the ELF file!)

The console should look like this at the beginning to make sure it works:

```
Console
-- Execute commands on a temporary offset by appending '@ offset' to your command.

qt.qpa.xcb: QXcbConnection: XCB error: 3 (BadWindow), sequence: 1755, resource id: 10489724, major code: 40 (TranslateCoords), minor code: 0
Process with PID 5179 started...
= attach 5179 5179
PTRACE_GETREGSET: No such device
PTRACE_GETREGSET: No such device
PTRACE_GETREGSET: No such device
```

→ Also, make sure its in "Debuggee Input" instead of "Rizin console" when inputting data to the ELF file!

Note: This is the LOOP that rechecks each 3 inputs that we placed for Key no. 2!

```
0x00402094    call explode      ; sym.explode
0x00402099    mov byte [var_1h], 1
0x0040209d    jmp 0x40211b
0x0040209f    movzx eax, byte [var_1h]
0x004020a3    cvtsi2sd xmm1, eax
0x004020a7    movzx eax, byte [var_1h]
0x004020ab    cvtsi2sd xmm0, eax
0x004020af    call pow         ; sym.imp.pow
0x004020b4    movapd xmm1, xmm0
0x004020b8    movsd xmm0, qword [0x004026f0]
0x004020c0    mulsd xmm0, xmm1
0x004020c4    cvttss2si eax, xmm0
0x004020c8    mov dword [var_14h], eax
0x004020cb    mov eax, dword [var_14h]
0x004020ce    mov byte [var_15h], al
0x004020d1    movzx eax, byte [var_1h]
0x004020d5    sub eax, 1
0x004020d8    cdqe
0x004020da    mov edx, dword [rbp + rax*4 - 0x23c]
0x004020e1    movsx eax, byte [var_15h]
0x004020e5    cmp edx, eax
0x004020e7    je 0x402117
0x004020e9    movzx eax, byte [var_1h]
0x004020ed    mov esi, eax
0x004020ef    lea rdi, str.Input_number__d_is_wrong ; 0x40268b ; const char *format
0x004020f6    mov eax, 0
0x004020fb    call printf      ; sym.imp.printf ; int printf(const char *format)
0x00402100    mov rdx, qword [var_320h]
0x00402107    mov eax, dword [var_314h]
0x0040210d    mov rsi, rdx
```

→ We can know what possibly the 2nd input is! (and 3rd one!)

Final Tip

- You just have to figure out the keys, other than that, if you encounter segmentation fault by any chance, just execute the "bomb" ELF in terminal and place your answer there cause for some reason, the fread gives out seg fault!

```
student@desktop:~/Desktop$ ls
bomb cutter.desktop ghidra.desktop local_vars.txt vta.desktop
student@desktop:~/Desktop$ ./bomb
Bomb armed...

Enter key 1: gaf98gsv
Enter key 2: 18
72
4294967270

So you actually did it, huh...

Here's your flag:
8a7f473a24828b8e
```

