GameBob

Reversing

80 Points

TLDR

- A stack object created on the __ in which the corrupted flag resides.
- A secret function that fixes the corrupted flag.
- Redirection of PC to **secret** before printing the stack and another redirection back to printing the fixed flag.

Booting up a debugger called BGB (which is nice since it's suited for Gameboy debugging), and loading the Gamebob.gb file, we get this printout:

Ah ha! Easy, right? We have the flag. Not precisely. We can see that the printed flag is all corrupted.

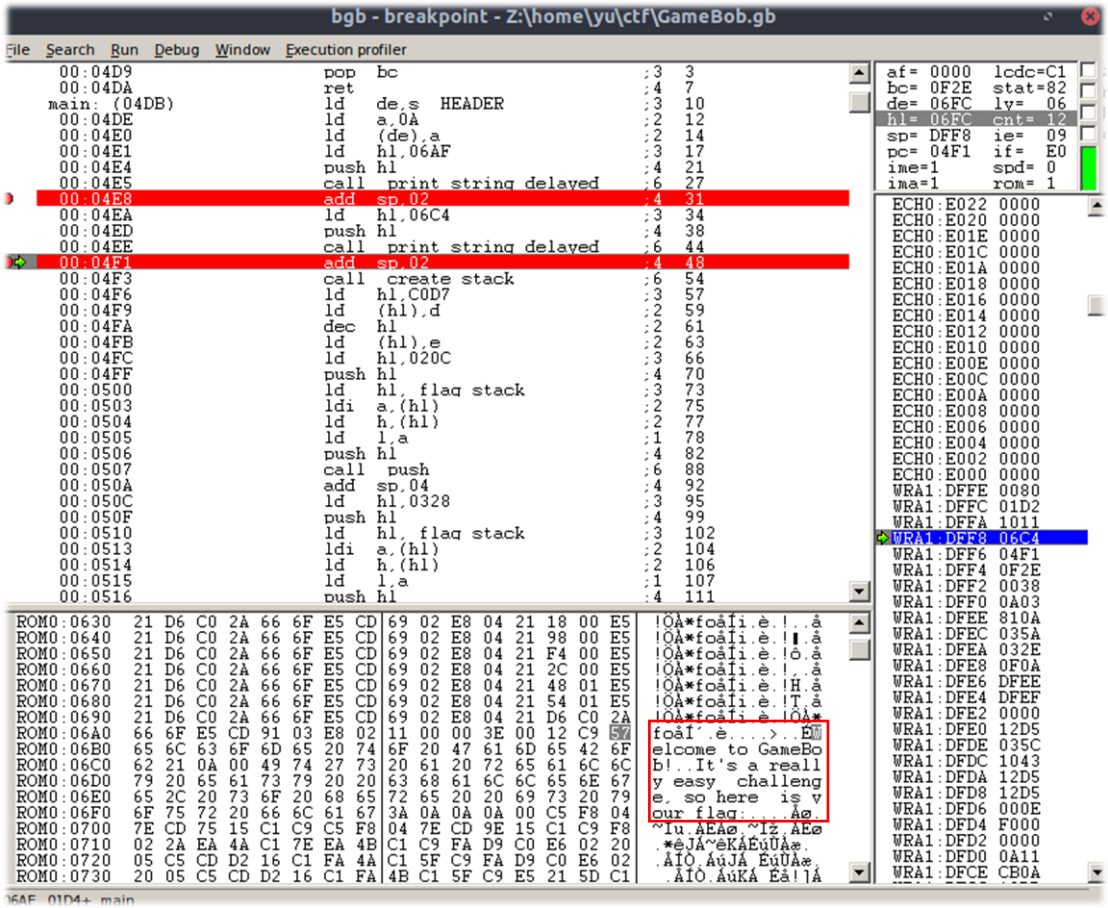
Opening up BGB there some new or different assembly commands seen, so after awhile of reading through a Gameboy commands and CPU explanation file:

http://marc.rawer.de/Gameboy/Docs/GBCPUman.pdf

And figuring out how to work with BGB, I started reading through the disassembled file and found a main function using CTRL+F to search.

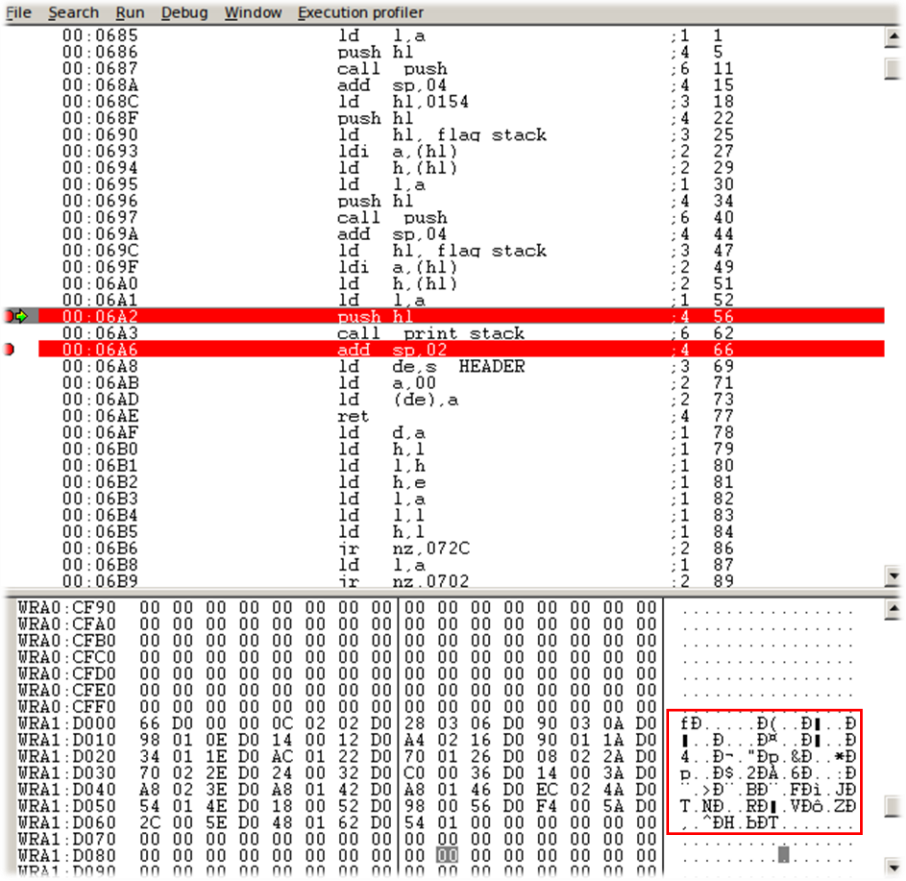
Seeing 2 calls at the beginning of the main function to **print_string_delayed** and putting breakpoints around them, we could see, sure enough, that they are responsible for printing out the **Welcome to GameBob!**, and **It's a really easy challenge, so here is your flag:**.

And looking at the address that was pushed to the stack right before the call, we could see that those strings actually reside on address 06FC:



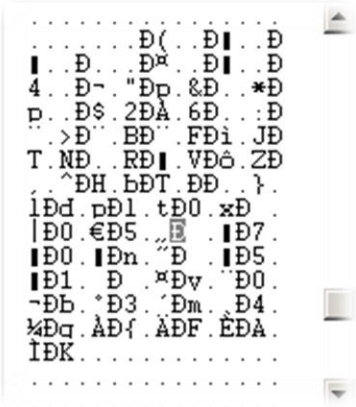
Our flag is still missing, however. If we keep reading through the main function, we could see that they implemented some sort of a Stack object there using **create_stack** and that they repeatedly push characters into it, when the new stack itself resides on address D000.

Going to that address in memory and putting breakpoints after each call to push, we could see the "stack" filling up with characters that really reminds us of the corrupted flag (but reversed), and afterwards in main there's a call to **print_stack**, which essentially prints our newly created corrupted flag:

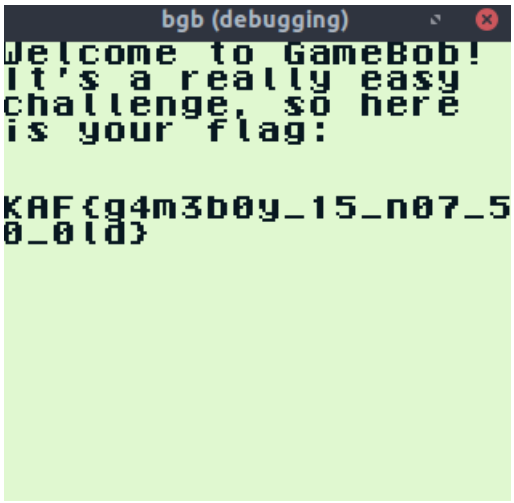


Another thing that bothered me was a function called secret which was really suspicious, and after reading through it I figured it essentially pops characters out of the stack, changes them and puts them back in – this might be our flag fixing function (!)

Putting a breakpoint on it to see if it is called during execution, it in fact was not called. So, we could redirect the main function to it once the corrupted flag stack is ready by changing the value of the PC to the address of secret. Doing that and looking at the memory where the corrupted flag stack resided, we could see that it's now changed to something that really resembles a real flag:



Now all we have to do is make sure this new stack is printed – so we can redirect the function back to pushing the parameters to the stack before calling **print_stack**, and sure enough:



We got out flag!

KAF{g4m3b0y_15_n07_50_0ld}