

# 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:

The screenshot shows the bgb debugger interface. The main window displays assembly code for the 'main' function. Two calls to 'print\_string\_delayed' are highlighted with red boxes. The first call is at address 00:04E8, and the second is at address 00:04EE. The code shows that the strings 'Welcome to GameBob!' and 'It's a really easy challenge' are pushed onto the stack before the first call. The memory dump on the right shows the stack contents, with the strings 'Welcome to GameBob!' and 'It's a really easy challenge' visible at addresses 06FC and 06FD respectively. The strings are reversed in memory, which is typical for a stack.

```

00:04D9      pop     bc                      ;3 3
00:04DA      ret                          ;4 7
main: (04DB)
00:04DE      ld      de, $HEADER            ;3 10
00:04DE      ld      a, 0A                 ;2 12
00:04E0      ld      (de), a               ;2 14
00:04E1      ld      hl, 06AF              ;3 17
00:04E4      push    hl                    ;4 21
00:04E5      call   print_string_delayed   ;6 27
00:04E8      add     sp, 02                 ;4 31
00:04EA      ld      hl, 06C4              ;3 34
00:04ED      push    hl                    ;4 38
00:04EE      call   print_string_delayed   ;6 44
00:04F1      add     sp, 02                 ;4 48
00:04F3      call   create_stack           ;6 54
00:04F6      ld      hl, 00D7              ;3 57
00:04F9      ld      (hl), d               ;2 59
00:04FA      dec     hl                    ;2 61
00:04FB      ld      (hl), e               ;2 63
00:04FC      ld      hl, 020C              ;3 66
00:04FF      push    hl                    ;4 70
00:0500      ld      hl, flag_stack        ;3 73
00:0503      ldi     a, (hl)               ;2 75
00:0504      ld      h, (hl)               ;2 77
00:0505      ld      l, a                  ;1 78
00:0506      push    hl                    ;4 82
00:0507      call   push                   ;6 88
00:050A      add     sp, 04                ;4 92
00:050C      ld      hl, 0328              ;3 95
00:050F      push    hl                    ;4 99
00:0510      ld      hl, flag_stack        ;3 102
00:0513      ldi     a, (hl)               ;2 104
00:0514      ld      h, (hl)               ;2 106
00:0515      ld      l, a                  ;1 107
00:0516      push    hl                    ;4 111

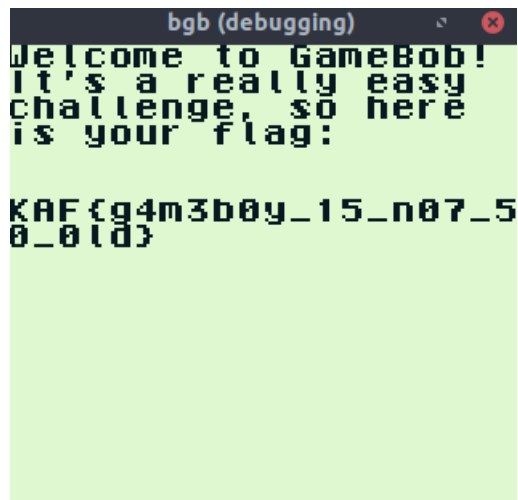
ROM0:0630  21 D6 C0 2A 66 6F E5 CD 69 02 E8 04 21 18 00 E5 10A*foa'i.e...a
ROM0:0640  21 D6 C0 2A 66 6F E5 CD 69 02 E8 04 21 98 00 E5 10A*foa'i.e...a
ROM0:0650  21 D6 C0 2A 66 6F E5 CD 69 02 E8 04 21 F4 00 E5 10A*foa'i.e...a
ROM0:0660  21 D6 C0 2A 66 6F E5 CD 69 02 E8 04 21 2C 00 E5 10A*foa'i.e...a
ROM0:0670  21 D6 C0 2A 66 6F E5 CD 69 02 E8 04 21 48 01 E5 10A*foa'i.e...a
ROM0:0680  21 D6 C0 2A 66 6F E5 CD 69 02 E8 04 21 54 01 E5 10A*foa'i.e...a
ROM0:0690  21 D6 C0 2A 66 6F E5 CD 69 02 E8 04 21 D6 C0 2A 10A*foa'i.e...a
ROM0:06A0  66 6F E5 CD 91 03 E8 02 11 00 00 3E 00 12 C9 50 fca'i.e...>.E
ROM0:06B0  65 6C 63 6F 6D 65 20 74 6F 20 47 61 6D 65 42 6F welcome to GameBo
ROM0:06C0  62 21 0A 00 49 74 27 73 20 61 20 72 65 61 6C 6C b!..It's a reall
ROM0:06D0  79 20 65 61 73 79 20 20 63 68 61 6C 6C 65 6E 67 y easy challeng
ROM0:06E0  65 2C 20 73 6F 20 68 65 72 65 20 20 69 73 20 79 e, so here is v
ROM0:06F0  6F 75 72 20 66 6C 61 67 3A 0A 0A 0A 00 C5 F8 04 our flag:..Ag
ROM0:0700  7E CD 75 15 C1 C9 C5 F8 04 7E CD 9E 15 C1 C9 F8 ~lu.AEa~iz.AE
ROM0:0710  02 2A EA 4A C1 7E EA 4B C1 C9 FA D9 C0 E6 02 20 *eJA~eKAeUa~
ROM0:0720  05 C5 CD D2 16 C1 FA 4A C1 5F C9 FA D9 C0 E6 02 .AIO.AuJA.EuUa~
ROM0:0730  20 05 C5 CD D2 16 C1 FA 4B C1 5F C9 E5 21 5D C1 .AIO.AuKA.Ea!JA
  
```

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:



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:



```
bgb (debugging)
Welcome to GameBob!
It's a really easy
challenge, so here
is your flag:
KAF{g4m3b0y_15_n07_5
0_0ld}
```

We got out flag!

KAF{g4m3b0y\_15\_n07\_50\_0ld}