Lab Assignment #1: Analyzing Compiled Programs

CYBR 570 : Reverse Engineering
Spring 2025

Submitted By: Lilly Carlascio
Submission Date: 04/08/25

1 Static Analysis with objdump

1.1 Disassembling The Program

1. **Identify each function and write down its offset address.**
   (a) create_node: 1179
   (b) append: 11cd
   (c) prepend: 1237
   (d) insert: 1271
   (e) del: 132f
   (f) pop_head: 1406
   (g) pop_tail: 144c
   (h) print_ll: 14db
   (i) free_list: 153e
   (j) main: 1577

2. **How is a function call made in assembly? Look at a call instruction and describe what happens.**
   (a) When a function call is made in assembly, it is "calling" the memory address at which that function is sitting at. For example, in main, the call function is taking the argument "11cd append", which is the memory address for the append function.

3. **What is the relationship between the offset address shown in objdump and the actual RIP value when debugging in gdb? Why might these differ?**
   (a) All of the function calls end in the last 3 characters (ex: main ends in 577 in both) but the difference is that the objdump addresses all begin in 0s and the RIP value in gdb all begin in 5s. These might differ because the binary from objdump is loaded into the memory at a specific address by the OS and the RIP value is calculated by adding this load address to the offset. (https://stackoverflow.com/questions/66064377/are-the-addresses-displayed-by-objdump-final-addresses-or-just-offsets)

1.2 Analyzing Function Calls and Stack Usage

1. **Why do we push values to the stack? Explain in terms of function calls and return addresses.**
   (a) When a function is called, the return address is pushed onto the stack, so the program knows where to return to after the function has completed.
2. **What does the RIP register represent? How does it change during a function call?**
   (a) The RIP register represents the address of the next instruction to executes. During a function call, the RIP register changes because now it will hold the address of the next instruction of the function it is in. For example when I was in the main function it holds the return address of the main function, but when I stepped into the append function, the RIP register value was changed to the next address instruction to execute in the append function.
3. **What instruction is used to call your add last function from main?**
   (a) call 0x5555555551cd append
4. **What does the ret instruction do at the end of a function? Why is it important?**
   (a) The ret instruction at the end of a function returns the program to the instruction it left off on before entering the function that was called. This is important because the program would just normally execute the next line of code which is likely another function, and the program needs to return to where it left off from in the main function.

2 Dynamic Analysis with gbd

2.1 Starting gdb and Setting Breakpoints

1. **What value does RIP hold at each breakpoint? How does it help you know where you are in the code?**
   (a) At the break-point set in main, the RIP register holds the value for the first line in the main function. As I step through it moves through each value, since the RIP value holds the address of the next instruction to be executed.
2. **Compare the RIP values at specific breakpoints with the offset addresses from objdump. What patterns do you observe? What accounts for any differences?**
   (a) Seeing the same last three chars in the offsets and the RIP values between the gdb assembly and the asm file from the objdump. Even the jne and jmp calls have the similar last 3 chars in the addresses they are referring to.

2.2 Inspecting Conditionals and Jumps

1. **Describe the conditional statement and its assembly code. What type of jump and comparison is used, and which EFLAGS bit (e.g., ZF for zero) is being checked?**

(a) The conditional statements use mov, test, jne, to get into the conditional, and then mov and jmp calls within the conditional. When the jne instruction is being called, the PF, ZF, IF, RF flags are set. After the conditional is finished and on the jmp call, the PF, ZF, IF, RF flags are being set. (https://stackoverflow.com/questions/12395477/interpreting-eflags-in-ddd)

(b) BF is the Parity Flag which is set if a low-order byte of the result has even parity

(c) ZF is the Zero Flag and is set when the jne call is made, if ZF == 0, then jne takes the jump (so the values being compared in the conditional are not equal) and if ZF == 1, then jne does no jump (the values being compared in the conditional are equal or 0)

(d) IF is the Interrupt Flag which controls whether the CPU responds to maskable hardware interrupts and since it is set, means interrupts are enabled so it can halt its current execution and jump to the approproate interrupt handler routine (https://en.wikipedia.org/wiki/Interrupt_flag)

(e) RF is the Resume Flag which is used by the processor during debugging to manage exceptions. Likely not being altered by the logic in the program (https://www.intel.com/content/dam/www/public/us/en/documents/manuals/64-ia-32-architectures-software-developer-vol-1-manual.pdf)