Ofri Kastenbaum 208708768

Orel Shai 206586109

Cheating is Bad

Reverse Engineering and Binary Patching Steps

1. **Tool Selection:**
   Chose IDA as the primary reverse engineering tool for this task.
2. **Initial Analysis:**
   Loaded the binary patchwork into IDA.
   Performed an initial analysis to understand the overall structure and flow of the program.
3. **Locating the Flag Check:**
   Searched through the disassembled code for string references and function calls that might be related to the flag.
   Identified a section of code that seemed responsible for checking conditions and printing the flag.
4. **Understanding the Condition:**
   Found a conditional jump (JZ [rbp+var_4], 0) instruction that was crucial to the control flow.
   Analyzed the preceding instructions to understand what condition was being checked.
5. **Patch Identification:**
   Noted that the conditional jump led to a block that set a value to 0 and then exited the program if the condition was true.
   This was likely the point where the program decided not to print the flag based on the result of a comparison.
6. **Binary Patching:**
   Modified the binary to change the value moved into the register (1 instead of 0) that was being checked by the conditional jump.
   This involved finding the exact instruction that set the register's value and altering it to ensure the condition for the jump would not be met.
7. **Testing the Patch:**
   Saved the patched binary.
   Executed the patched binary to verify the flag was printed as expected.

Conclusion

By identifying and modifying the crucial jump instruction, we were able to bypass the condition that prevented the flag from being printed. The key steps included:

- Analyzing the control flow.
- Understanding the condition that determined the program's behavior. **(jz [rbp+var_4], 0)**
- Patching the binary to alter the program’s execution path **(mov [rbp+var_4], 1)**

This approach allowed us to successfully reveal the secret flag.