Digital Safe Reverse Engineering Challenge

1. Challenge Overview

A compiled executable representing a digital safe was provided. The program prompts the user to enter six digits sequentially. If the correct sequence is entered, the safe unlocks. Otherwise, it remains locked. The objective was to recover the correct 6-digit code through binary analysis.

2. Initial Reconnaissance

The binary was inspected using static analysis tools to understand its structure and behavior. The 'file' command confirmed the executable format. The 'strings' utility revealed embedded prompts such as 'Enter Digit %d/6:', indicating a 6-digit validation mechanism.

3. Static Analysis

The executable was analyzed using a disassembler to inspect its control flow and validation logic. The program reads one digit at a time and compares it against a stored constant value. A loop executes exactly six iterations. If any comparison fails, execution branches to a failure routine. If all comparisons succeed, execution reaches the unlock routine.

4. Code Extraction

Within the validation routine, a hardcoded sequence of six digits was identified. These digits were stored sequentially and directly compared against user input.

Recovered Code

852741

5. Dynamic Verification

The program was executed and the extracted sequence was entered manually. The safe successfully unlocked, confirming the correctness of the recovered code.

6. Security Analysis

The implementation is insecure because the secret code is stored directly in the binary. No hashing, encryption, or obfuscation techniques were used. This makes the program vulnerable to reverse engineering, memory inspection, and static analysis.

7. Recommended Improvements

To improve security: • Do not store plaintext secrets in compiled binaries. • Store only hashed representations of the secret. • Use secure comparison techniques. • Apply code obfuscation or runtime integrity checks. • Separate secret validation logic from static constants.

Conclusion

Through structured static analysis and inspection of validation logic, the hardcoded 6-digit code was successfully extracted and verified. This challenge highlights the risks of embedding secrets directly in executable files.