Crackme Write-Up

License Validation via Weak File Check (x86 / Windows)

Overview

This crackme implements a simple file-based license validation mechanism. The program expects a license file, reads a fixed number of bytes, and performs a minimal arithmetic check on only the first four bytes. If the check passes, the license is considered valid.

No cryptography, hashing, or obfuscation is used, making the protection trivial to bypass or emulate.

File Handling Logic

The license validation flow begins in the function:

```
FUN_0040139d(LPCSTR license_path)
```

Behavior

1. Opens the license file using `CreateFileA`
2. Reads **exactly 0x13 (19) bytes**
3. If fewer than 19 bytes are read → license rejected
4. If exactly 19 bytes are read → buffer passed to validator function

Relevant Windows APIs:

- `CreateFileA`
- `ReadFile`
- `CloseHandle`

Core Validation Function

The actual validation logic resides in:

```
FUN_00401409(byte *buffer)
```

This function determines whether the license file is valid.

Validation Logic Analysis

Loop Setup

```
iVar1 = 4;    // Number of iterations
bVar4 = 2;    // First divisor
bVar2 = 3;    // Second divisor
```

- Only **4 bytes** are checked
- The remaining **15 bytes are ignored entirely**

Validation Loop

For each iteration:

```
if (buffer[i] % current_divisor != 0)
  return INVALID;
```

After each iteration, the divisors are updated:

```
next = bVar2 + bVar4;
bVar4 = bVar2;
bVar2 = next;
```

Derived Divisor Sequence

This produces a Fibonacci-like sequence:

Byte Index	Divisor	Condition
0	2	`byte[0] % 2 == 0`
1	3	`byte[1] % 3 == 0`
2	5	`byte[2] % 5 == 0`
3	8	`byte[3] % 8 == 0`

Only the **first four bytes** are validated.

If all four checks pass, the function returns success.

Key Observations

- No checksum, hash, or signature
- No dependency between bytes
- No validation of remaining 15 bytes
- Divisors are generated dynamically but predictably
- License file contents can be arbitrary as long as the first four bytes pass

This makes the crackme **functionally equivalent to a 4-byte modulo check**.

License File Requirements

A valid license file must:

- Be **exactly 19 bytes**
- Satisfy:
- Byte 0 divisible by 2
- Byte 1 divisible by 3
- Byte 2 divisible by 5
- Byte 3 divisible by 8
- Remaining bytes: unused / ignored

Exploitation Strategy

Method 1 — Key Generation (No Patch)

Create a 19-byte file where:

- First four bytes satisfy the divisibility rules
- Remaining bytes are arbitrary (zeroed for simplicity)

This bypasses the license check cleanly without modifying the binary.

Method 2 — Binary Patch

Alternatively, the validation function can be neutralized by:

- Patching the conditional jump after the modulo check
- Forcing the function to always return success

This approach is unnecessary given the weakness of the check.

Security Assessment

Aspect	Evaluation
Obfuscation	None
Entropy	Extremely low
Cryptography	None
Tamper resistance	None
Effort to bypass	Trivial

This crackme demonstrates **what not to do** when implementing license protection.

Conclusion

The license system relies on a predictable, non-cryptographic check applied to only 4 bytes of input. As a result, generating a valid license file is trivial and does not require reverse engineering beyond basic control-flow analysis.

This crackme serves as a good beginner example for:

- Static analysis
- File I/O reversing
- Control-flow reconstruction
- Keygen logic extraction