Observable Timing Discrepancy in HMAC Verification (CWE-208)

Moderate **notamitgamer** published **GHSA-wqq3-wfmp-v85g** last week

Package

mojic (npm)

Affected versions

<= 2.1.3

Patched versions

2.1.4

Description

Summary

The `CipherEngine` in Mojic v2.1.3 uses a standard equality operator (`!==`) to verify the HMAC-SHA256 integrity seal during the decryption phase. This creates an Observable Timing Discrepancy (CWE-208), allowing a potential attacker to bypass the file integrity check via a timing attack.

Details

In `lib/CipherEngine.js`, the footer check validates the HMAC signature using a standard string comparison:
`if (footerHex !== calcDigest) { ... }`

Standard string comparisons in JavaScript short-circuit; they return `false` the moment a character mismatch occurs. Because the time taken to evaluate the comparison is proportional to the number of matching leading bytes, an attacker can measure the exact microseconds it takes for the engine to throw the `FILE_TAMPERED` error. By repeatedly altering the signature byte-by-byte and analyzing these minute timing differences, a malicious actor can theoretically forge a valid HMAC signature without possessing the decryption password.

PoC

The vulnerable implementation is located in `lib/CipherEngine.js`, within the `getDecryptStream()` flush method (approximately line 265):

```
// Vulnerable Code
if (footerHex !== calcDigest) {
    this.emit('error', new Error("FILE_TAMPERED"));
    return;
}
```

Recommended Remediation:

Replace the standard equality operator with Node.js's built-in constant-time comparison utility, crypto.timingSafeEqual().

```
// Remediated Code
const footerBuffer = Buffer.from(footerHex, 'hex');
const calcBuffer = Buffer.from(calcDigest, 'hex');

if (footerBuffer.length !== calcBuffer.length || !crypto.timingSafeEqual(footerBuffer, c
    this.emit('error', new Error("FILE_TAMPERED"));
    return;
}
```

Impact

If successfully exploited, an attacker could tamper with the encrypted .mojic payload and forge a valid HMAC signature. This bypasses the integrity seal, tricking the decryption engine into processing maliciously injected emoji streams. Because the engine translates these emojis back into C keywords and raw data chunks, this could ultimately result in arbitrary Code Injection into the restored .c source code when an unsuspecting user decrypts the tampered file.

Severity

Moderate 4.7 / 10

CVSS v3 base metrics

Attack vector	Local
Attack complexity	High
Privileges required	None
User interaction	Required
Scope	Unchanged
Confidentiality	None
Integrity	High
Availability	None

Learn more about base metrics

CVSS:3.1/AV:L/AC:H/PR:N/UI:R/S:U/C:N/I:H/A:N

CVE ID

CVE-2026-41244

Weaknesses

▶ CWE-208

Credits

 **notamitgamer2** Reporter

 **notamitgamer** Analyst