

Vulnerability Title:

Buffer overflow observed in `parse_statement()` and `tokenizer` code in `js.c`

Security Level:

Critical

Vulnerable Component:

Function: `parse_statement()` and `next_token()` tokenizer.

File : `js.c`

Vulnerability Details :

Fuzzer discovered a crash caused by a buffer overflow in the file `js.c`. The program improperly copies unbounded strings into the fixed-size buffer (`current.text` and `name[64]`) using unsafe copying logic that lacks bound check, such as the use of `strcpy()`. This allows an attacker-supplied input to override adjacent memory and may potentially lead to control-flow corruption, remote code execution etc.

Evidence:

In the given screenshot below, the crash file was discovered

```
pxb@PXB: ~/Videos/assignment1/fixed/in/crashes$ ls
id_000000, sig_11, src_000001, time_182233, execs_36854, op_havoc, rep_2
pxb@PXB: ~/Videos/assignment1/fixed/in/crashes$
```

In the given screenshot below, ASAN(Address Sanitizer) output is observed when running the crash against the original vulnerable binary.

```
root@PXB: /home/pxb/Videos/assignment1/fuzz
==1147701==ERROR: AddressSanitizer: global-buffer-overflow on address 0x639a63ff7ea4 at pc 0x639a63a89d51 bp 0x7ffea714dd0 sp 0x7ffea714dc0
WRITE of size 1 at 0x639a63ff7ea4 thread T0
#0 0x639a63a89d50 in next_token /home/pxb/Videos/assignment1/fuzz/js.c:40
#1 0x639a63a8a57b in parse_statement /home/pxb/Videos/assignment1/fuzz/js.c:146
#2 0x639a63a8a871 in interpret /home/pxb/Videos/assignment1/fuzz/js.c:168
#3 0x639a63a8aa25 in main /home/pxb/Videos/assignment1/fuzz/js.c:191
#4 0x7db62c2a1c9 in __libc_start_main ../sysdeps/ntl/libc_start_main.h:58
#5 0x7db62c2a28a in __libc_start_main_impl ../csu/libc-start.c:360
#6 0x639a6397a684 in _start (/home/pxb/Videos/assignment1/fuzz/js+0x9684) (BuildId: fd72902775da7d604ee962b958c198a6b9d05350)

0x639a63ff7ea4 is located 0 bytes after global variable 'current' defined in 'js.c:22:7' (0x639a63ff7e60) of size 68
SUMMARY: AddressSanitizer: global-buffer-overflow /home/pxb/Videos/assignment1/fuzz/js.c:40 in next_token
Shadow bytes around the buggy address:
 0x639a63ff7e00: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
 0x639a63ff7e80: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
 0x639a63ff7d00: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
 0x639a63ff7d80: 00 00 00 00 00 00 00 00 F9 F9 F9 F9 04 F9 F9 F9
 0x639a63ff7e00: F9 F9 F9 F9 F9 F9 F9 F9 00 00 00 00 00 00 00 00
==0x639a63ff7e80: 00 00 00 00 [04]F9 F9 F9 F9 F9 F9 00 00 00
 0x639a63ff7f00: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
 0x639a63ff7f80: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
 0x639a63ff8000: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
 0x639a63ff8080: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
 0x639a63ff8100: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
Shadow byte legend (one shadow byte represents 8 application bytes):
Addressable: 00
Partially addressable: 01 02 03 04 05 06 07
Heap left redzone: f9
Freed heap region: fd
Stack left redzone: f1
Stack mid redzone: f2
Stack right redzone: f3
Stack after return: f5
Stack use after scope: f8
Global redzone: f9
Global init order: f6
Poisoned by user: f7
Container overflow: fc
Array cookie: ac
Intra object redzone: bb
ASan internal: fe
Left alloca redzone: ca
Right alloca redzone: cb
==1147701==ABORTING
root@PXB: /home/pxb/Videos/assignment1/fuzz#
```

The ASAN report in the screenshot confirms a global buffer overflow triggered by the fuzzed input. The overflow occurred inside the `next_token()` function where characters are copied into `current.text` without size validation. ASAN provides the specific memory address, stack trace, and shadow bytes that identify the out-of-bounds write. This evidence not only verifies the vulnerability but also aligns with the behavior observed in the fuzzing crash file.

Root Cause Analysis:

- The tokenizer (`next_token()`) copies numeric/identifiers tokens into the global `current.text[64]` buffer without checking that the token would fit in the 64 bytes buffer. If a very long sequence of digits or letters is produced from mutated input, it can exceed the size of the current text, leading to memory corruption in adjacent global memory. This issue has been observed as the overwriting of variable and return addresses.
- In the `parse_statement()` function, after parsing `let <id> = <expr>`, a new variable is added to the `vars[]` array with `vars[var_count++] = (Var){ .value = val }`. The name is copied using `strcpy(vars[var_count - 1].name, name)`. However, `'strcpy()'` does not check the size of the destination buffer, which can lead to buffer overflow vulnerability.

Rationale behind the patch:

- Token copying (numbers & identifiers) — add bounds checking while copying into current.text so we never write more than sizeof(current.text)-1 bytes and always terminated by a null.

```
if (*src == '\0') { current.type = TOKEN_EOF; return; }

if (isdigit(*src)) {
    int i = 0;
    while (isdigit(*src) || *src == '.') current.text[i++] = *src++;
    current.text[i] = '\0';
    current.type = TOKEN_NUMBER;
    return;
}

if (isalpha(*src)) {
    int i = 0;
    while (isdigit(*src) || *src == '.') {
        if (i < (int)sizeof(current.text) - 1) current.text[i++] = *src;
        src++;
    }
    current.text[i] = '\0';
    current.type = TOKEN_NUMBER;
    return;
}
```

- Insertion into vars[] and copying the name — check capacity of vars[] before appending; use bounded copy (strncpy) when storing name; always terminated by a null.

```
next_token();
double val = parse_expr();
if (current.type == TOKEN_SEMI) next_token();
vars[var_count++] = (Var){ .value = val };
strcpy(vars[var_count - 1].name, name);

/* Prevent overflow of vars[]: check capacity before inserting. */
if (var_count < (int)(sizeof(vars) / sizeof(vars[0]))) {
    vars[var_count].value = val;
    /* ensure name fits into vars[].name (both are 64 bytes) */
    strncpy(vars[var_count].name, name, sizeof(vars[var_count].name) - 1);
    vars[var_count].name[sizeof(vars[var_count].name) - 1] = '\0';
    var_count++;
} else {
    fprintf(stderr, "Too many variables (overflow attempt)\n");
    exit(1);
}
```

Patch Implementation:

- Token bounds checks:** By checking $i < (\text{int})\text{sizeof}(\text{current}.text) - 1$ before copying each byte, the code prevents writes beyond the buffer's last usable byte. The final $\text{current}.text[i] = '\0'$; ensures the stored token is always a valid C string.
- Safe insertion to vars[]:** The var_count capacity check prevents an out-of-bounds write into the vars array when it is full. Replacing strcpy() with strncpy() (plus explicit null-termination) prevents name from overflowing the vars[].name buffer. Together, these eliminate two different overflow vectors: overflowing current.text into adjacent global and overflowing a variable's name into adjacent memory.

```
pxb@PXB:~/Videos/assignment1/fixed$ sudo ./run.sh
[sudo] password for pxb:
[*] Docker detected - building and running container...
[+] Building 1.3s (11/11) FINISHED
--> [internal] load build definition from Dockerfile
--> --> transferring dockerfile: 553B
--> [internal] load metadata for docker.io/library/ubuntu:22.04
--> [auth] library/ubuntu:pull token for registry-1.docker.io
--> [internal] load .dockerrcignore
--> --> transferring context: 2B
--> [1/5] FROM docker.io/library/ubuntu:22.04@sha256:104ae83764a5119017b8e8d6218fa0832b09df65aae7d5a6de29a85d813da2fb
--> --> resolve docker.io/library/ubuntu:22.04@sha256:104ae83764a5119017b8e8d6218fa0832b09df65aae7d5a6de29a85d813da2fb
--> [internal] load build context
--> --> transferring context: 1.44MB
--> CACHED [2/5] WORKDIR /app
--> CACHED [3/5] RUN apt-get update && apt-get install -y --no-install-recommends build-essential clang patch nano vim git ca-certificates curl
--> [4/5] COPY . /app
--> [5/5] RUN chmod +x /app/run.sh
--> exporting to image
--> --> exporting layers
--> --> writing image sha256:a7542d1d593543dc99ace1cf93dfb33fb19e2508623466cbfb7c1e41f2650186
--> --> naming to docker.io/library/js_test_image
[*] Applying patch...
patching file js.c
Patch applied successfully ✅
[*] Using crash file: in/crashes/id_000000,sig_11,src_000001,time_182233,execs_36854,op_havoc,rep_2
[*] Compiling ASan (diagnostic) binary...
[+] ASan binary built: js_asan

[*] Running ASan binary on crash...
== ASan output (first 20 lines) ==
Result: 1e-63
=====
== RESULT: Crash no longer occurs - issue fixed ✅ ==
[*] Building fast instrumented binary for fuzzing...
[+] Fast binary built: js_fast
pxb@PXB:~/Videos/assignment1/fixed$
```

- Result:** The previous ASAN detected global buffer overflow arises from writing past current into vars — the token bounds checks stop current overflow; the vars capacity/name copy checks stop overflow on vars insertion.

References :

- <https://nvd.nist.gov/vuln-metrics/cvss/v3-calculator>
- <https://www.cve.org/CVERecord/SearchResults?query=buffer+overflow>
- <https://docs.docker.com/build/concepts/overview/>
- <https://clang.llvm.org/docs/AddressSanitizer.html>
- <https://aflplus.plus/>