Vulnerability Report

Vulnerability Details

Upon fuzzing, it was discovered that buffer overflows could occur as a result of feeding the program inputs containing very long tokens (continuous strings of alphanumerical characters), specifically those longer than 63 characters. This occurs during the tokenisation part of the program, specifically at js.c:32 and js.c:40, where next_token() loops over *src as it appends current.text, where current is a Token object.

As defined in the program, the Token struct has two attributes: type, of type TokenType; and text, which is an array that is 64 characters long. When the tokeniser finds a non-whitespace character, it creates a new token. If it begins with an alphanumerical character, it begins a loop where each next character that is also alphanumerical (or numerical if the first character was a number) is appended to the token's text attribute. Then, it adds a null terminator to the text attribute and moves on to the next token.

Since it does not perform any checks about the length of the token, the tokeniser will iterate continuously until it finds a non-alphanumerical character. This means it could write more than 64 characters to the token's text buffer, causing it to overflow and overwrite other aspects of memory, potentially including the return address.

Here is an example input that would induce this error:

```
let
aaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaa
aaaaaaaaaa =

222222222;let xa =

222222aaaaaaaaaaaaaaaaaaaaaaaaaaxaaaaaaaaaaaaaaaaaaa;

c
```

At its barest, this could be used to cause the program to crash due to it trying to return to an invalid address. More seriously, this vulnerability could be exploited to customise the return address to any other address in memory, opening the opportunity for something like a ret-to-libc attack. To do this exploit, the attacker simply needs to provide their own input to the program. Such a simple exploit can have significant consequences, making this vulnerability severe.

Root Cause Analysis

GDB was run on the instrumented source file revealing the crash occurred where part of the input was copied into a buffer that was shorter than its length. Combing through the program source code using the line numbers dumped by GDB, we found this was occurring when a loop was appending to a fixed 64 length array without checking if there was space remaining. We determined the root cause as a buffer overflow resulting from a lack of input validation.

Suggested Patch Reasoning

The root of the vulnerability is the lack of input validation. By implementing a length check before a token is copied into the fixed-length buffer, we can catch any occurrences where the token exceeds the buffer length and exit the process in a controlled way. This prevents the program writing data past the end of the buffer, which means it doesn't violate memory protection meaning it doesn't crash. It is also a very simple and efficient fix, simply involving a comparison of two integers during each iteration of the while loop, so it will negligibly impact performance.

AddressSanitizer was also introduced to compilation to be able to find and get reports on other memory errors.