6 napi_get_value_string_X allow various kinds of memory corruption

Share:

TIMELINE

tniessen submitted a report to Node.js. Jan 27th (3 years ago)

Summary:

`napi_get_value_string_latin1`, `napi_get_value_string_utf8`, `napi_get_value_string_utf16` are vulnerable to buffer overflows, partially due to an integer underflow.

Description:

`napi_get_value_string_latin1`, `napi_get_value_string_utf8`, and `napi_get_value_string_utf16` behave like this:

1. If the output pointer is `NULL`, return.
2. Write `min(string_length, bufsize - 1)` bytes to the output buffer. Note that `bufsize` is an unsigned type, so this leads to an integer underflow for `bufsize == 0`. Since this is a `size_t`, the underflow will cause the entire string to be written to memory, no matter how long the string is.
3. Finally, write to `buf[copied]`, where `copied` is the number of bytes previously written. Even if step 2 hadn't written out of bounds, this would (for `bufsize == 0`).

Steps To Reproduce:

Code 230 Bytes Wrap lines Copy Download

```
Napi::Value Test(const Napi::CallbackInfo& info) {
  char buf[1];
  // This should be a valid call, e.g., due to a malloc(0).
  napi_get_value_string_latin1(info.Env(), info[0], buf, 0, nullptr);
  return info.Env().Undefined();
}
```

Code 128 Bytes Wrap lines Copy Download

```
const binding = require('bindings')('validation');
console.log(binding.test('this could be code that might later be executed'));
```

Running the above script corrupts the call stack:

Code 120 Bytes Wrap lines Copy Download

```
tniessen@local-vm:~/validation-fails$ node .
*** stack smashing detected ***: <unknown> terminated
Aborted (core dumped)
```

The best outcome is a crash, but a very likely outcome is data corruption. If the attacker can control the string's contents, they can even insert code into the process heap, or modify the call stack. Depending on the architecture and application, this can lead to various issues, up to remote code execution.

It is perfectly valid to pass in a non-NULL pointer for `buf` while specifying `bufsize == 0`. For example, `malloc(0)` is not guaranteed to return `NULL`. A npm package might correctly work on one machine based on the assumption that `malloc(0) == NULL`, but might create severe security issues on a different host. Passing a non-NULL pointer is also not ruled out by the documentation of N-API, so it is not valid to assume that `buf` will always be `NULL` if `bufsize == 0`.

Impact

npm packages and other applications that use N-API may involuntarily open up severe security issues, that might even be exploitable remotely. Even if `buf` is a valid pointer, passing `bufsize == 0` allows to write outside of the boundaries of that buffer.

Step 2 of the description allows an attacker to precisely define what is written to memory by passing in a custom string. Depending on whether the pointer points to heap or stack, possible results include data corruption, crashes (and thus DoS), and possibly even remote code execution, either by writing instructions to heap memory or by corrupting the stack.

Many attacks are likely caught by kernel and hardware protection mechanisms, but that depends on the specific hardware, kernel, and application, and memory layout. Even if they are caught, the entire process will crash (which is still good compared to other outcomes).

addaleax posted a comment. Jan 27th (3 years ago)

I'm not too worried about this being a common case, as most usage of these functions will first figure out the length of the buffer based on the length of the string (like e.g. the node-addon-api code does), but I'm okay with treating this as a security issue.

Would you be okay opening PRs that fix this in the private repository? It should be fairly straightforward to do so.

I think this also means that we'd want to push out new releases for node-addon-api@v1.x and @v2.x? Maybe ping Gabriel when you open the PRs?

tniessen posted a comment. Jan 27th (3 years ago)

> I'm not too worried about this being a common case

I am also convinced that it is not common. But it might still affect real applications, and from my perspective, the implementation violates the general contract of not writing to a zero-length buffer in a dangerous way that can potentially allow targeted memory corruption, up to remote code execution.

> most usage of these functions will first figure out the length of the buffer based on the length of the string

I agree, but even then, OOB access is possible: Assume that `len(string) == 0`. The application probably allocates the buffer using `malloc(0)`, which is not guaranteed to return `NULL`, and may in fact return any pointer, that `napi_get_value_string_*` will then write at least one byte to, even though the string is empty.

> Would you be okay opening PRs that fix this in the private repository? It should be fairly straightforward to do so.

@addaleax, I assume the suggestion to push out new releases of node-addon-api@v1.x and @v2.x is because they included a copy of the related code that was used for version of Node.js that did not yet provide N-API ? If so we should probalby loop in Nicola Del Gobbo (@NickNaso) as he's being helping with publishing the recent node-addon-api releases.

octetcloud changed the status to **Triaged**. Feb 12th (3 years ago)

May 4th (3 years ago)
octetcloud changed the report title from **napi_get_value_string_* allow various kinds of memory corruption** to **napi_get_value_string_X allow various kinds of memory corruption**.

octetcloud closed the report and changed the status to **Resolved**. Jun 2nd (3 years ago)
https://nodejs.org/en/blog/vulnerability/june-2020-security-releases/

octetcloud requested to disclose this report. Jun 2nd (3 years ago)

This report has been disclosed. Jul 2nd (2 years ago)

The Internet Bug Bounty rewarded tniessen with a **$250** bounty. Jul 15th (2 years ago)