New issue

Promise Hang on AbortSignal in `@tootallnate/once` #8

Closed

nanak-singh opened on Feb 2

Package Information

- **Package Name:** `@tootallnate/once`
- **Affected Versions:** All versions up to and including `3.0.0` (latest)
- **Vulnerability Type:** Promise Hang / Denial of Service (DoS)
- **Severity:** Low / Medium
- **Status:** Unpatched (project appears to be in maintenance / legacy mode)

Summary

The `@tootallnate/once` package supports an optional `AbortSignal` to allow cancellation before the Promise resolves. However, when the signal is aborted, the implementation removes event listeners but **does not resolve or reject the Promise**.

As a result, the returned Promise remains in a **permanently pending state**, causing any `await` or `.then()` usage to hang indefinitely.

This creates a control-flow leak that can lead to stalled requests, blocked workers, or degraded application availability.

Technical Details

Root Cause

When an `AbortSignal` is provided and `abort()` is called:

- Event listeners are removed correctly (good for memory safety)
- **The Promise is never settled (neither resolved nor rejected)**

Relevant code (from `src/index.ts`):

```
export default function once(emitter, name, { signal } = {}) {
  return new Promise((resolve, reject) => {
    function cleanup() {
      signal?.removeEventListener('abort', cleanup);
      emitter.removeListener(name, onEvent);
      emitter.removeListener('error', onError);
    }

    // onEvent() and onError() call cleanup() and resolve/reject

    signal?.addEventListener('abort', cleanup); // <-- Promise never settles
    emitter.on(name, onEvent);
    emitter.on('error', onError);
  });
}
```

The `cleanup()` function removes listeners but does not call `resolve()` or `reject()`, leaving the Promise permanently pending.

Proof of Concept (PoC)

Tested with: Node.js v20.18.2

```
const EventEmitter = require('events');
const once = require('@tootallnate/once');

async function test() {
  const emitter = new EventEmitter();
  const controller = new AbortController();

  console.log('Starting once()...');
  const promise = once(emitter, 'foo', { signal: controller.signal });

  let settled = false;
  promise.then(() => settled = true).catch(() => settled = true);

  console.log('Aborting...');
  controller.abort();

  await new Promise(r => setTimeout(r, 1000));

  if (!settled) {
    console.log('[!] VULNERABILITY VERIFIED: Promise remains permanently pending');
  }
}
```

```
test();
```

Output:

```
Starting once()...
Aborting...
[!] VULNERABILITY VERIFIED: Promise remains permanently pending
```

Impact

Applications using `AbortSignal` for cancellation or timeouts may experience:

- Hung `await once(...)` calls
- Stalled HTTP request handlers
- Blocked worker threads or job queues
- Resource exhaustion over time

This is particularly risky in **concurrency-limited environments**, where permanently pending Promises can accumulate and exhaust available execution capacity.

Severity Justification

- **Low → Medium severity**
- No direct data compromise
- Can lead to **application-level Denial of Service (DoS)** via hung control flow
- Impact depends on how widely the function is awaited in critical execution paths

---

Recommendation / Fix

Expected Behavior

When the `AbortSignal` is aborted, the Promise returned by `once()` should be **rejected**, ideally with an `AbortError`.

Suggested Fix

Introduce a dedicated abort handler that:

- Calls `cleanup()`

- Rejects the Promise

```
function onAbort() {
  cleanup();
  const err = new Error('The operation was aborted');
  err.name = 'AbortError';
  reject(err);
}

signal?.addEventListener('abort', onAbort, { once: true });
```

Additionally, if `signal.aborted === true` at invocation time, the Promise should reject immediately.

Workaround for Consumers

Until patched, consumers should avoid awaiting `once()` directly with an `AbortSignal`, or wrap it using `Promise.race()` with a Promise that explicitly rejects on abort.

Notes

- This issue is **not a memory leak** (event listeners are correctly removed).
- It is a **control-flow leak**, where a Promise becomes permanently pending, breaking expected cancellation semantics.

Disclosure

This issue was identified through manual code review and runtime testing.
No exploit code beyond the PoC above is required to trigger the issue.

TooTallNate closed this as completed in b9f43cc on Feb 9

G-Rath on Mar 5

fwiw while this is a bug, I don't think this is actually a security vulnerability because it requires specific js implementation to trigger and "exploit" e.g. unlike prototype pollution which can be exploited by uploading JSON or via URL.

Note that the PoC does not show how this could be exploited or easily triggered by an external party, which is a key difference between "bug" and "vulnerability"

I'm also pretty sure an unresolved promise uses very little resources so you'd have to have a *ton* of them to actually have a noticeable impact

G-Rath on Mar 6

@TooTallNate would you be open to backporting this at all? I'm happy to do the patching, if you're happy to do the publishing 🙂

👍 2

jiangxin0503 on Mar 13

Hi **@TooTallNate** ,
Looks like the issue is not valid for version 1, if so, **@nanak-singh** CVE should be updated only version above 2 is affected.

Thanks,
Xin

gChiqueti on Mar 13 · edited by gChiqueti Edits

@jiangxin0503 it seems version 1.0.2 is vulnerable too. I executed the POC code with node v18.20.3 and version 1.0.2 and accused the vulnerability.

Updating to version 3.0.1 and adapting the POC code for this version, it shows that it was fixed

YashVardhanTrip on Mar 17

> fwiw while this is a bug, I don't think this is actually a security vulnerability because it requires specific js implementation to trigger and "exploit" e.g. unlike prototype pollution which can be exploited by uploading JSON or via URL.
>
> Note that the PoC does not show how this could be exploited or easily triggered by an external party, which is a key difference between "bug" and "vulnerability"
>
> I'm also pretty sure an unresolved promise uses very little resources so you'd have to have a *ton* of them to actually have a noticeable impact

achaw ji yesa hai kya?

coderabbitai mentioned this on Mar 22

🔒 Security: Vulnerabilities found in Frontend dependencies YKDBontekoe/miru#263

Copilot mentioned this last month

Override @tootallnate/once to 3.0.1 to resolve CVE-2026-3449 navikt/crm-ips#2712

Copilot mentioned this 2 weeks ago

Patch transitive `@tootallnate/once` to 3.0.1 via npm override KoshkiKode/unshelvd#91

delanni added a commit that references this issue last week

`Fix promise hang when `AbortSignal` is aborted ···` 1bdbc00

Metadata

Assignees

No one assigned

Labels

No labels

Projects

No projects

Milestone

No milestone

Relationships

None yet

Development

No branches or pull requests

Participants