Prototype Pollution

Affecting org.webjars.npm:cached-path-relative package, versions **[0,]**

INTRODUCED: 19 JAN 2022 CVE-2021-23518 CWE-1321

Share

How to fix?

There is no fixed version for `org.webjars.npm:cached-path-relative`.

Overview

org.webjars.npm:cached-path-relative is a memoize the results of the path.relative function.

Affected versions of this package are vulnerable to Prototype Pollution via the `cache` variable that is set as `{}` instead of `Object.create(null)` in the `cachedPathRelative` function, which allows access to the parent prototype properties when the object is used to create the cached relative path. When using the origin path as `__proto__`, the attribute of the object is accessed instead of a path.

Note: This vulnerability derives from an incomplete fix in https://security.snyk.io/vuln/SNYK-JS-CACHEDPATHRELATIVE-72573

###PoC

```
let cpr = require("cached-path-relative"); process.chdir("tst"); cpr("__proto__", "polluted")
console.log({}.polluted);

//output: ../polluted
```

Details

Prototype Pollution is a vulnerability affecting JavaScript. Prototype Pollution refers to the ability to inject properties into existing JavaScript language construct prototypes, such as objects. JavaScript allows all Object attributes to be altered, including their magical attributes such as `__proto__`, `constructor` and `prototype`. An attacker manipulates these attributes to overwrite, or pollute, a JavaScript application object prototype of the base object by injecting other values. Properties on the `Object.prototype` are then inherited by all the JavaScript objects through the prototype chain. When that happens, this leads to either denial of service by triggering JavaScript exceptions, or it tampers with the application source code to force the code path that the attacker injects, thereby leading to remote code execution.

There are two main ways in which the pollution of prototypes occurs:

- Unsafe `Object` recursive merge
- Property definition by path

Unsafe Object recursive merge

The logic of a vulnerable recursive merge function follows the following high-level model:

```
merge (target, source)
foreach property of source

  if property exists and is an object on both the target and the source merge(target[property], source[property]) else
  target[property] = source[property]
```

When the source object contains a property named `__proto__` defined with `Object.defineProperty()`, the condition that checks if the property exists and is an object on both the target and the source passes and the merge recurses with the target, being the prototype of `Object` and the source of `Object` as defined by the attacker. Properties are then copied on the `Object` prototype.

Clone operations are a special sub-class of unsafe recursive merges, which occur when a recursive merge is conducted on an empty object: `merge({},source)`.

`lodash` and `Hoek` are examples of libraries susceptible to recursive merge attacks.

Property definition by path

There are a few JavaScript libraries that use an API to define property values on an object based on a given path. The function that is generally affected contains this signature: `theFunction(object, path, value)`

If the attacker can control the value of "path", they can set this value to `__proto__.myValue`. `myValue` is then assigned to the prototype of the class of the object.

Types of attacks

There are a few methods by which Prototype Pollution can be manipulated:

Snyk CVSS

Exploit Maturity	Proof of concept
Attack Complexity	Low

See more

NVD 9.8 CRITICAL

Do your applications use this vulnerable package?

In a few clicks we can analyze your entire application and see what components are vulnerable in your application, and suggest you quick fixes.

Test your applications

Snyk ID	SNYK-JAVA-ORGWEBJARSNPM-2348246
Published	20 Jan 2022
Disclosed	19 Jan 2022
Credit	P.Adithya Srinivas, Masudul Hasan Masud Bhuiyan, Cristian-Alexandru Staicu

Report a new vulnerability | Found a mistake?

Type	Origin	Short description
Denial of service (DoS)	Client	This is the most likely attack. DoS occurs when `Object` holds generic functions that are implicitly called for various operations (for example, `toString` and `valueOf`). The attacker pollutes `Object.prototype.someattr` and alters its state to an unexpected value such as `Int` or `Object` . In this case, the code fails and is likely to cause a denial of service. **For example:** if an attacker pollutes `Object.prototype.toString` by defining it as an integer, if the codebase at any point was reliant on `someobject.toString()` it would fail.
Remote Code Execution	Client	Remote code execution is generally only possible in cases where the codebase evaluates a specific attribute of an object, and then executes that evaluation. **For example:** `eval(someobject.someattr)` . In this case, if the attacker pollutes `Object.prototype.someattr` they are likely to be able to leverage this in order to execute code.
Property Injection	Client	The attacker pollutes properties that the codebase relies on for their informative value, including security properties such as cookies or tokens. **For example:** if a codebase checks privileges for `someuser.isAdmin` , then when the attacker pollutes `Object.prototype.isAdmin` and sets it to equal `true` , they can then achieve admin privileges.

Affected environments

The following environments are susceptible to a Prototype Pollution attack:

- Application server
- Web server
- Web browser

How to prevent

1. Freeze the prototype— use `Object.freeze (Object.prototype)` .
2. Require schema validation of JSON input.
3. Avoid using unsafe recursive merge functions.
4. Consider using objects without prototypes (for example, `Object.create(null)`), breaking the prototype chain and preventing pollution.
5. As a best practice use `Map` instead of `Object` .

For more information on this vulnerability type:

Arteau, Oliver. “JavaScript prototype pollution attack in NodeJS application.” GitHub, 26 May 2018

References

- GitHub Fix Commit