

Prototype Pollution

Affecting json-ptr package, versions <3.0.0

INTRODUCED: 31 AUG 2021
CVE-2021-23509 ⓘ
CWE-1321 ⓘ
FIRST ADDED BY SNYK
Share

How to fix?

Upgrade json-ptr to version 3.0.0 or higher.

Overview

json-ptr is a complete implementation of JSON Pointer (RFC 6901) for nodejs and modern browsers.

Affected versions of this package are vulnerable to Prototype Pollution. A type confusion vulnerability can lead to a bypass of CVE-2020-7766 when the user-provided keys used in the pointer parameter are arrays.

PoC

```
const { JsonPointer } = require("json-ptr");
// JsonPointer.set({}, ['proto', 'polluted'], 'yes', true); // console.log(polluted); // Error: Attempted prototype
pollution disallowed.

JsonPointer.set({}, [['proto'], 'polluted'], 'yes', true); console.log(polluted); // yes
```

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:

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

MEDIUM

Search by package name or CVE

Snyk CVSS

Exploit Maturity
Proof of concept ⓘ

Attack Complexity
High ⓘ

See more

> NVD

3.8 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 Learn

Learn about Prototype Pollution vulnerabilities in an interactive lesson.

Start learning

Snyk ID
SNYK-JS-JSONPTR-1577291

Published
3 Nov 2021

Disclosed
31 Aug 2021

Credit
Alessio Della Libera of Snyk Research Team

Report a new vulnerability

Found a mistake?

Type	Origin	Short description
		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 Commit](#)
- [GitHub PR](#)
- [GitHub README](#)
- [Snyk Blog](#)

PRODUCT

[Snyk Open Source](#)

[Snyk Code](#)

[Snyk Container](#)

[Snyk Infrastructure as Code](#)

[Test with Github](#)

[Test with CLI](#)

RESOURCES

[Vulnerability DB](#)

[Documentation](#)

[Disclosed Vulnerabilities](#)

[Blog](#)

[FAQs](#)

COMPANY

[About](#)

[Jobs](#)

[Contact](#)

[Policies](#)

[Do Not Sell My Personal Information](#)

CONTACT US

[Support](#)

[Report a new vuln](#)

[Press Kit](#)

[Events](#)

FIND US ONLINE

TRACK OUR DEVELOPMENT



© 2022 Snyk Limited

Registered in England and Wales. Company number: 09677925

Registered address: Highlands House, Basingstoke Road, Spencers Wood, Reading, Berkshire, RG7 1NT.