# Prototype Pollution

**Prototype Pollution** is a vulnerability specific to **JavaScript and Node.js**, where an attacker manipulates the **prototype of base objects** (Object.prototype, Array.prototype, etc.) by injecting properties into them.

Since **all JavaScript objects inherit from Object.prototype**, this means **new properties can “magically” appear in all objects** — leading to:

* Logic bypasses
* Denial of Service (DoS)
* Remote Code Execution (RCE) (in some cases)

# Why It Happens

JavaScript allows dynamic creation of object properties. Vulnerable code may do:

let user = {};

user[input.key] = input.value;

If attacker sets:

{

"key": "\_\_proto\_\_",

"value": { "isAdmin": true }

}

Then:

{}.isAdmin → true

All objects inherit isAdmin: true = **polluted prototype**

# Impact

| Impact | Description |
| --- | --- |
| Privilege Escalation | App logic trusts obj.isAdmin to grant access |
| Denial of Service | Overwrite internal properties like toString, constructor, length |
| RCE | In rare cases, it leads to arbitrary code execution via polluted objects in unsafe sinks (e.g., eval()) |
| Security Logic Bypass | Undermine validation, session state, permissions, etc. |

# Common Attack Payloads

## JavaScript:

payload = {

"\_\_proto\_\_": {

"isAdmin": true

}

}

## Query Param:

?\_\_proto\_\_[isAdmin]=true

## JSON Input:

{

"\_\_proto\_\_": {

"access": "granted"

}

}

## Nested Keys:

user['\_\_proto\_\_']['debug'] = true;

# Where It Happens

Node.js APIs

Object merging utilities: lodash, merge, deepMerge, jQuery.extend, etc.

JSON-based APIs that deserialize user input into objects

Libraries that don’t restrict prototype keys

# Real-World Examples

| App | Issue |
| --- | --- |
| Lodash (<4.17.11) | \_.merge() allowed prototype pollution |
| jQuery’s $.extend() | Unsafe merges with \_\_proto\_\_ injected keys |
| AngularJS | Some versions allowed \_\_proto\_\_ pollution in templates |
| npm modules | Dozens of npm packages were vulnerable, leading to widespread supply chain risk |

# Detection Techniques

| Technique | Description |
| --- | --- |
| Fuzz API with \_\_proto\_\_, constructor, prototype keys |  |
| Observe side-effects | Unexpected property appearances in objects |
| Security headers/flags suddenly bypassed |  |
| Source code audit | Look for unsafe usage of Object.assign, merge, deepMerge |
| Tools | Use ppscan, snyk, npm audit, or Burp extensions to scan |

# Mitigation

## 1. Prevent Unsafe Merge/Assignment

* Block reserved prototype keys like:
  + \_\_proto\_\_, constructor, prototype

function isSafeKey(key) {

return !['\_\_proto\_\_', 'constructor', 'prototype'].includes(key);

}

## 2. Use Safe Libraries

* Use patched versions of:
  + lodash >= 4.17.11
  + deepmerge with allowPrototypes: false
* Avoid Object.assign() on untrusted input

## 3. Freeze Object Prototypes

Prevent pollution:

Object.freeze(Object.prototype);

Object.seal(Object.prototype);

Works for frontend apps and Node.js APIs

## 4. Input Validation

* Strictly validate object structure in:
  + JSON bodies
  + Query parameters
  + WebSocket and GraphQL requests

## 5. Run SCA Tools

Use tools like:

* npm audit
* snyk test
* ppscan
* retire.js

They detect prototype pollution in dependencies.

# Points

“Prototype Pollution is unique to JavaScript because **all objects inherit from a common prototype**.”

“A polluted prototype can result in **all future object instances being affected**.”

“Test for this by **injecting \_\_proto\_\_ or constructor keys**, and watching for logic breaks, permission bypasses, or errors.”