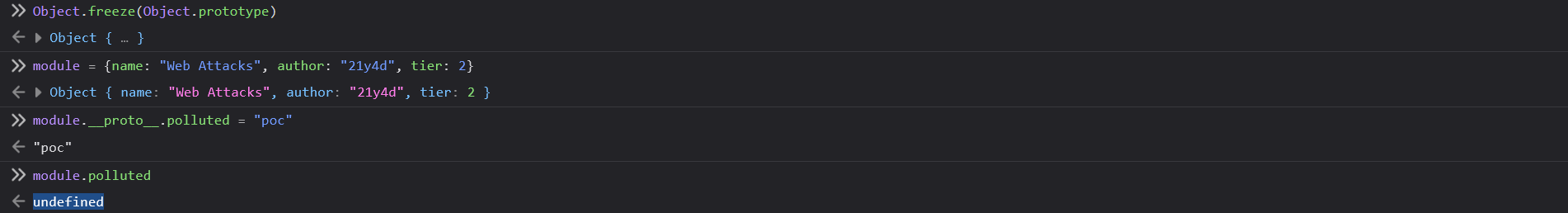
**Prevention & Patching**

There are multiple ways of tackling prototype pollution vulnerabilities.

The most obvious is sanitizing keys to ensure an attacker cannot inject keys referencing the prototype. However, while such an approach is simple in theory, implementing such a sanitizer is no easy task. As we have seen in previous sections, blocking the obvious key \_\_proto\_\_ is insufficient to prevent prototype pollution entirely. There are other ways of obtaining a reference to an object's prototype using the keys constructor and prototype. Thus, a sanitizer should block at least these three keys. However, a more secure approach would be implementing a whitelist approach that consists of a list of explicitly whitelisted keys. These keys need to be chosen carefully for the corresponding context and may even help to prevent further vulnerabilities such as [Mass Assignment](https://owasp.org/www-project-web-security-testing-guide/latest/4-Web_Application_Security_Testing/07-Input_Validation_Testing/20-Testing_for_Mass_Assignment).

Another way to prevent prototype pollution is by freezing an object, meaning it cannot be modified. This can be done using the [Object.freeze()](https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global_Objects/Object/freeze) function. If we call the function on the global Object.prototype object that all objects inherit from, any modifications to it are prevented. As an example, consider the following steps:



As we can see, the property module.polluted is undefined. That is because we froze the Object.prototype object using the Object.freeze function. Therefore, we prevented prototype pollution by disallowing the polluted property from being set.

However, this is not a universal fix since freezing the Object.prototype property alone may be insufficient. Recall the prototype pollution vulnerability we exploited to gain remote code execution in a previous sections. In that case, we polluted a property in the User.prototype object and did not modify the Object.prototype object. Therefore, in order to prevent that prototype pollution vulnerability, the User.prototype object needs to be frozen.

Lastly, we can also manipulate inheritance to set the prototype to null. This can be achieved using Object.create(null) to create the object, which sets the prototype of the newly created object to null. Thus, there are no inherited properties and no possibility of prototype pollution. However, since there is no prototype, the object does not contain properties like toString() and other useful properties provided by the global Object.prototype object. It only contains properties explicitly added to the object. While this can prevent prototype pollution vulnerabilities, it is probably impractical in many use cases.

Prototype pollution vulnerabilities arise when recursively manipulating an object's properties from user input, a functionality we should import from available libraries. As such, patching prototype pollution vulnerabilities is often as simple as using secure libraries and keeping them updated. An additional line of defense is provided by packages like [nopp](https://github.com/snyk-labs/nopp) which ensure some of the defenses discussed are implemented.