Research on Vulnerability: CVE-2023-26136

Description:
Versions of the package tough-cookie before 4.1.3 are vulnerable to Prototype Pollution due to improper handling of Cookies when using CookieJar in rejectPublicSuffixes=false mode. This issue arises from the manner in which the objects are initialized.

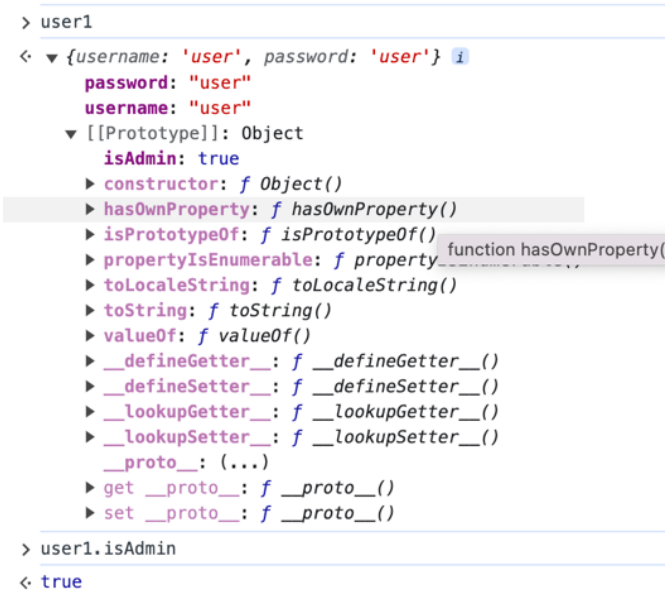
https://nvd.nist.gov/vuln/detail/CVE-2023-26136

Introduction:

What is Prototype Pollution?
JavaScript has the concept of objects, which is somewhat like dictionaries. An object can hold a set of variables of different types (e.g. string, Boolean, int etc.), the variable name is a key, and its value is the value, if we continue with the analogy of dictionaries and objects. More precisely, the object is analog to the dictionary **data structure** itself, and each variable is analog to a **key-value pair**. Objects also has the keyword **__proto__** which enables to add additional variables to an object via modifying its prototype. **Prototype Pollution** is an attack in which the attacker pollutes his object, by adding to it additional variables, via the __proto__ keyword. The below interaction with the browser's console demonstrates the attack.

```
> var admin1 = {username:"admin", password:"admin", isAdmin:true}
< undefined
> var user1 = {username:"user", password:"user"}
< undefined
> admin1.isAdmin
< true
> user1.isAdmin
< undefined
> user1.__proto__.isAdmin = true
< true
> user1
< ▶ {username: 'user', password: 'user'}
> user1.isAdmin
< true
```

As we can see, we have two users: admin1 and user1. admin1 object has a Boolean variable: "isAdmin", which is set to true. User1 doesn't has this variable altogether. At line 7, we add the isAdmin variable to the user1's prototype, and even though it is evident from line 10 that the variable hasn't been added to the user1's object itself, the console responds positively, this time, when we check for the value of user1.isAdmin. That's because user1 is inheriting the properties of its prototype.



It is also worth mentioning that the prototype is an object by itself, which in turn inherits properties and methods from its prototype, those creating a prototype chain. The chain ends with a null prototype, therefore a reasonable approach to prevent prototype pollution is to make the object we are working with, to explicitly inherit from a null prototype.

What are cookies?
Cookies (often known as internet cookies) are text files with small pieces of data — like a username and password — that are used to identify your computer as you use a network. Specific cookies are used to identify specific users and improve their web browsing experience.

Courtesy of Kaspersky: https://www.kaspersky.com/resource-center/definitions/cookies

What is CookieJar?
CookieJar is an object for storing cookies.

The Assignment:

The vulnerability:
According to the description of the vulnerability, it arises from the way Tough-Cookie initializes Cookies. Since Cookies are objects, they are theoretically at least, be vulnerable to **Prototype Pollution.**

The Risks:
By being able to temper with objects and particularly with cookies, via the object's prototype, the attacker can potentially access unauthorized data, execute remote code, cause a denial of service, hijack the session if the website relies on cookies for managing the session, and extracting sensitive data from the cookies themselves.

The Patch:
The patch was made at the file: **memstore.js**. According to the issue tracking as well as to the patch introduced at version 4.1.3, in order to patch the vulnerability, we need store the cookies in a map or create the this.idx object. By creating this.idx using: this.idx = Object.create(null); Instead of this.idx = {}, we practice my suggestion on how to prevent Prototype Pollution at the introduction, by inheriting from a null prototype, and cut the prototype chain.

Testing the vulnerability (index.js):
Snyk had published a Proof of Concept (PoC) for the discussed vulnerability. I built index.js upon it. Wrapped it in a try-catch logic to capture exceptions if they arise, added additional output to keep track of the tests progress and ended up with the required output (e.g. "EXPLOITED SUCCESSFULLY" or "EXPLOITED FAILED").
When running the command:
npm install tough-cookie@2.5.0 && node index.js
We're getting the following output:

```
Creating a new cookie jar...
Setting a normal cookie...
Setting an exploit cookie...
Trying to access the exploit cookie...
Cookie="Slonser=polluted; Domain=__proto__; Path=/notauth; hostOnly=false; aAge=0ms; cAge=0ms"
EXPLOITED SUCCESSFULLY
```

When running the command:
npm install ./tough-cookie-2.5.0-PATCHED.tgz && node index.js
We're getting the following output:

```
Creating a new cookie jar...
Setting a normal cookie...
Setting an exploit cookie...
Trying to access the exploit cookie...
Error: TypeError: Cannot read properties of undefined (reading 'Slonser')
    at main (/Users/orsn/Desktop/JS Projects/seal_security_project/index.js:41:34)
    at process.processTicksAndRejections (node:internal/process/task_queues:95:5)
EXPLOIT FAILED
```

At both scenarios (Running with the published 2.5.0 version and running with my patched version), we managed to set up both the normal cookie as well as the exploited cookie, but at the patched version, we failed to access the exploited cookie.

Summary:
At this assignment I learned about the Prototype Pollution attack, learned about the JavaScript object and been introduced to the Tough-Cookie package.