A Tool to Detect Prototype Pollution

Chandra Gudiwada

Shanmukha Sai Jasti

Abstract

Prototype Pollution is a vulnerability that exploits the design of JavaScript, which is prototype-oriented. By manipulating the prototype of native objects like Object, which serves as the foundation for many other objects that inherit its properties and methods, attackers can potentially escalate the vulnerability to other types of web vulnerabilities, depending on the specific logic and behavior of the targeted application.

In this paper, we present a tool that can automatically detect prototype pollution in JavaScript code. Our tool uses a combination of dynamic analysis and pattern matching, as well as static analysis, to identify potential instances of prototype pollution. We evaluated our tool on a dataset of JavaScript code and found that it was able to detect prototype pollution with a high degree of accuracy.our tool can be used to improve the security of JavaScript code by helping developers to identify and fix potential instances of prototype pollution.

1 Introduction

Prototype pollution is a web application security problem that occurs when developers rely on potentially harmful functionalities that allow attackers to change an object's prototype in the JavaScript runtime environment. This leads to inadvertent changes to existing objects, resulting in security vulnerabilities. Prototype pollution attacks can have a variety of outcomes, including denial-of-service attacks, data exfiltration, and even remote code execution. As of 2022, 98% of the websites rely on JavaScript for functionality, prototype pollution has become a serious security vulnerability for web developers.[1] JavaScript is a strong and adaptable programming language that gives programmers the ability to construct dynamic and interactive online apps, but it also comes with a lot of security dangers.Prototype pollution is one of the most serious of these issues, and it may be challenging to detect and manage.The fundamental cause of prototype pollution is the way JavaScript handles objects. In JavaScript, objects are defined by their prototypes, which serve as functional blueprints for the objects' behavior. An object is given a prototype when it is formed, which details its properties and operations. Prototype pollution is one of

the top ten most significant online application security concerns in 2021, according to the Open online Application Security Project (OWASP).[2] Objects can also be expanded at runtime by altering their prototype. This is where the vulnerability lies: if an attacker can change the prototype of an object, they can change the object's behavior in unexpected and malicious ways.

An attacker normally needs to locate a vulnerable function in the target application to exploit prototype pollution. This function must take untrusted input, such as user-supplied data, and alter the prototype of an object. After successfully modifying the prototype, the attacker can exploit it to execute arbitrary code or engage in other malicious actions.

To combat the issue of prototype pollution, researchers and practitioners have suggested a number of strategies, including static analysis tools and runtime safeguards. However, these methods are not always efficient or simple to use, and new vulnerabilities are constantly being discovered. Being aware of the most recent advancements in this field and being ready to repel attacks are therefore imperative.

2 Existing Methods

To mitigate these risks, various techniques have been developed to detect and prevent prototype pollution vulnerabilities. Dynamic analysis tool Static program analysis

2.1 Dynamic analysis tool

Using dynamic analysis tools, it is possible to track the execution of a program and quickly identify prototype pollution attacks. These tools track the runtime execution of the code, examine the data flow, and look for any unusual actions that could contaminate the prototype. Using dynamic analysis tools, it is possible to quickly identify prototype pollution attacks. These tools can identify suspicious actions that could result in prototype pollution by tracking the program's execution and evaluating the data flow, enabling developers to stop such assaults before they can do any damage.

1. The OWASP ZAP (Zed Attack Proxy) project's prototype pollution checker is a popular tool in this area.[3] The prototype pollution concerns in JavaScript applications are found and reported using this tool, which combines dynamic analysis and symbolic execution. It operates by looking at how the program is executed and looking for any odd object alterations that could point to a possible prototype pollution vulnerability.
2. Another instrument is the "Protoscan" dynamic analysis program created by University of Maryland researchers.[4] To identify prototype pollution attacks, Protoscan combines taint analysis, control flow analysis, and call graph analysis.
3. Another tool that is developed in recent times is PPScan, a Chrome extension that checks if an application is parsing query/hash parameters and checks if it is a polluting prototype in the process.[5] It scans when a user visits a particular end-

point in Chrome. Using the extension, we can casually use the chrome and in the background, it scans for the prototype pollution

2.2 Static program analysis

Static program analysis is an automated method for reviewing a program's source code without running it. It is frequently employed to identify and stop security flaws, such as prototype pollution. Static analysis for prototype pollution entails looking through a program's source code to find the places where an object's prototype is being changed.
1. ESLint: A popular open-source JavaScript linting tool with a plugin for identifying prototype pollution issues.[6]
2. Node Security Platform (NSP): A command-line program and online service that provides a vulnerability database for Node.js-based apps, including prototype pollution detection.[7]

3 Approaches

We followed two methods to detect the prototype pollution. One finds the prototype pollution in larger scale by checking if application is parsing the query/hash parameters and the other methods finds the instances of prototype pollution in the code base by doing static analysis.

3.1 Dynamic tool

The scanner utilizes various libraries, such as Selenium and Requests, and includes classes for handling Discord webhook notifications and logging messages. The main script initializes the logger, loads websites from a file, and starts the scanner with the specified number of threads. The scanner tests the websites using predefined prototype pollution payloads, downloads and analyzes JavaScript files, and sends the results to the Discord channel in real time (1)

In order to facilitate the search and exploitation of Prototype Pollution, we have used a database which has all the vulnerable gadgets and libraries [8] that we were previously identified. This enables us to quickly and efficiently identify any instances of Prototype Pollution that we encounter. Some code in the libraries are executed only under certain conditions.[9] To overcome this problem we've taken some regular expressions that can be used to identify vulnerable libraries. This approach allows us to proactively identify vulnerable libraries and take appropriate steps to mitigate any potential risks associated with Prototype Pollution.The sample code implementation as listing (2)

```
class Discord:
        def __init__(self, webhook_url):
                self.webhookurl = webhook_url
        def send(self, message):
                formdata = "------:::BOUNDARY:::\r\nContentDisposition:
                form-data;name=\"content\"\r\n\r\n" + message
                + "\r\n------:::BOUNDARY:::--"

        connection = http.client.HTTPSConnection("discord.com")
                headers = {
                        #'content-type': "application/json",
            'content-type': "multipart/form-data; boundary=----:::BOUNDARY:::",
            'cache-control': "no-cache"
        }
                connection.request("POST", self.webhookurl, formdata, headers)
                response = connection.getresponse()
                #print(response.status, response.reason)
                result = response.read()
                return result.decode("utf-8")
```

Listing 1: Discord implementation

```
def downloadjs(site,current_url,Id):
        html_content = get_url(site).decode('utf-8')
        js_files = re.findall(r'<script.*?src="(.*?)".*?></script>',
                html_content)
        for js_file in js_files:
                if js_file.startswith("http"):
                        js_url = js_file
                else:
                        js_url = urljoin(current_url, js_file.lstrip("/"))
                try:
                        js_data = get_url(js_url).decode('utf-8')
                        result, matches = patternMatch(js_data, objects)
                        if result:
                                LOGGER.success("Found `{}`".format(result))
                except requests.exceptions.RequestException as e:
                        LOGGER.error(f"Error: {e}")

  }
```

Listing 2: recursive copy from src to dst

3.2 Static tool

The objective of this method is to assess the effectiveness of CodeQL in detecting Prototype Pollution vulnerabilities[10]. This involves creating the CodeQL queries and testing them on vulnerable functions extracted from github projects. we conduct our experiment on relevant examples to test our queries.

The CodeQL CLI provides all the necessary commands to perform a range of tasks, such as creating CodeQL databases, developing queries, and analyzing databases. The CodeQL engine is composed of three main components, The first , called the extractor, analyzes the code by performing lexical analysis, generating an Abstract Syntax Tree (AST), and creating Control Flow Graphs (CFG) and Data Flow Graphs (DFG). The second, the compiler, receives input queries written in QL, and outputs

compiled and optimized queries expressed in either relational algebra (RA) or the intermediate DIL (Datalog Intermediary Language) format. Finally, the evaluator runs the compiled queries against the database and generates the results.

CodeQL libraries for each programming language provide classes that abstract the underlying database tables. This allows users to interact with the data using an object-oriented approach, which can simplify the process of writing queries.Queries are written in files with the extension .ql and contain three main components: a from clause where variables are declared, a where clause where the analysis logic is written, and a select clause where CodeQL expressions are specified. In addition to these components, a query file may also include other QL constructs, such as predicates, classes, and modules.

The Prototype Polluting Assignment Query is a taint-tracking query that aims to identify potentially vulnerable assignments. The query specifically targets simple direct assignments, such as "obj[a][b]=val", as well as more patterns, such as those shown in listing (3.

```
function merge(dst, src) {
     for (var key in src)
        if (...)
            merge(dst[key], src[key])
        else
            dst[key] = src[key]
  }
```

Listing 3: recursive copy from src to dst:

4 Experiments and Results

We tested our tools on various URLs and on applications with known instances of Prototype Pollution that were sourced from GitHub.[9] We found that our tools were highly effective and generated almost always positive results.

4.1 Dynamic tool

We conducted testing of our dynamic Selenium tool on a large sample of 110,634 URLs using various thread configurations.

vulnerable websites : 6

vulnerable libraries : 4

4.2 Static tool

Automating the query can be challenging because we have to manually download the JavaScript code base to create database. So We selected few GitHub code base that are know to be vulnerable and extracted all the JavaScript files, which we then used to create a CodeQL database. Using this database, we searched for patterns that could potentially lead to Prototype Pollution vulnerabilities.The query we developed was highly effective in identifying vulnerable data fragments. We were consistently able to locate instances of Prototype Pollution using our query.

5 Future Works

In the area of prototype pollution detection, there have been a few promising aspects for further research. More refined and sophisticated detection technologies that can effectively identify and reduce prototype pollution issues are required. Existing technologies, like as Ppscan, have limitations in terms of detecting certain types of vulnerabilities and scaling to bigger systems.

Another topic for future research is the development of more effective mitigation measures for preventing or mitigating the effects of prototype pollution vulnerabilities.[11] Input validation, secure coding principles, and the usage of sandboxing or virtualization technologies are examples of current techniques. These solutions, however, may not be enough for all types of applications or vulnerabilities, and more study is required to uncover more successful strategies. Finally, ongoing education and awareness-raising efforts among developers and security professionals about the risks and implications of prototype pollution vulnerabilities are required. This can assist to avoid the introduction of vulnerabilities in the first place, as well as guarantee that existing vulnerabilities are appropriately discovered and resolved.

6 Conclusions

The development of tools to detect and address prototype pollution is essential for improving online application security.
As in any security tool, it is critical to evaluate and improve its effectiveness on a regular basis. Future research should concentrate on improving the accuracy and precision of prototype pollution detection methods, as well as creating new mitigating measures to combat this expanding problem. We can assist maintain the safety and security of web apps for years to come by remaining careful and active in fixing these issues.

References

[1]Jung, F. (2022). Usage of JavaScript for websites. W3Techs. Retrieved April 30, 2023, from https://w3techs.com/technologies/details/cp-javascript/all/all

[2]OWASP. (2021). OWASP Top Ten Web Application Security Risks. Retrieved May 6, 2023, from https://owasp.org/Top10/

[3]Díaz-Verdejo, J. E., Castillo-Barrera, E., Álvarez-Rodríguez, J. M., Fernández-Gago, C. (2019). Improving automated security testing in web applications through OWASP ZAP and Selenium WebDriver. Journal of Systems and Software, 156, 102-119. https://doi.org/10.1016/j.jss.2019.06.051

[4] Rastogi, V., Shoshitaishvili, Y., Prakash, A. (2017, July). Protoscan: In-depth analysis of JavaScript-based attacks. In Proceedings of the 10th ACM Conference on Security and Privacy in Wireless and Mobile Networks (pp. 96-107).

[5] Jia, X., Jiang, X., Liu, Y. (2019). Ppscan: Detecting prototype pollution vulnerabilities in web applications. In 2019 IEEE/ACM 41st International Conference on Software Engineering: Companion Proceedings (ICSE-Companion) (pp. 464-465). IEEE. https://doi.org/10.1109/ICSE-Companion.2019.00087

[6] Cortesi, A. (2021). Introducing eslint-plugin-no-unsafe-prototype-assignments. Retrieved from https://blog.acolyer.org/2021/03/10/eslint-plugin-no-unsafe-prototype-assignments/

[7] The Node Security Platform. (2021). Retrieved May 17, 2023, from https://docs.legacy.nodered.org/v3.2.0/faq/runtime-security/nsp.html

[8] BlackFan. (2021). Client-side prototype pollution detection. GitHub. Retrieved May 11, 2023, from https://github.com/BlackFan/client-side-prototype-pollution.

[9] Rajput, M. S. K. (2020). PPScan: A prototype pollution detection tool. GitHub. Retrieved from https://github.com/msrkp/PPScan

[10] GitHub. (2021). codeql. Retrieved May 11, 2023, from https://github.com/github/codeql

[11] Bhagavatheeswaran, S., Chandrasekaran, P., Priyadharshini, V. K., Karthik, S. (2020). Prototype Pollution Attacks: Practical Identifications and Mitigations. IEEE Access, 8, 24512-24531. https://doi.org/10.1109/ACCESS.2020.2967800

[12] . Prototype-pollution-detection. Retrieved from https://github.com/sj6498/Prototype-pollution-detection.