PROTOTYPE POLLUTION: INTRO

Jam11Vie

CONTEXT JS has a notion of "prototypes", all JS objects inherit properties + methods from a prototype. All methods and properties from a prototype will apply to any object inheriting from that prototype.

In JS, most applications will modify and manage JS objects and use JSON to assist in the process.

THE EXPLOIT IN GENERAL

Proto pollution involves injecting values to modify the prototype of a JS object, in order to achieve unexpected results (like RCE)
↳ The most common way to achieve this is through manipulating unsafe JSON functions.

General flow: We have 3 vars defined. Only one, the var "Jam", has an admin property.

```
> var Jam = {name: "Jam", admin: true}
<· undefined
> var Ty = {name: "Ty"}
<· undefined
> var Jinny = {name: "Jinny"}
<· undefined
```

Despite the var "Ty" not possessing the "admin" variable, we can poison its prototype to give "Ty" the admin value.

```
> Ty.__proto__.admin = true;
<· true
> Ty.admin
<· true
```

We can see that the prototype of "Ty" has a new property called "admin"

```
> Ty
<· ▼{name: "Ty"} i
     name: "Ty"
   ▼__proto__:
     admin: true
```

```
> Jinny.admin
<· true
> var Shaunak = {}
<· undefined
> Shaunak.admin
<· true
```

However since we infected the prototype, any and all objects that inherits from that prototype will have the "admin" property.

We have essentially polluted the prototype which every JS object inherits.

THE EXPLOIT IN DETAIL

__proto__ "__proto__" is a property, which, when invoked, accesses the prototype of the object. __proto__ points to the prototype, which is an object.

Why does using __proto__ affect all objects?
__proto__ is the "installed" prototype of the base object, which every other object inherits from.
Since the base prototype is affected, every object will also have all of the properties even if those properties aren't explicitly defined in the inheriting object. This is due to "prototypal inheritance"

- Checks for the requested property in the object
- If the property isn't present in the obj, it checks the obj's prototype.
- If the prototype doesn't have it, the internal [[prototype]] is checked (what __proto__ returns).

PROTOTYPE POLLUTION IN PRACTISE

EXPLOIT IDENTIFICATION proto pollution is an exploit unique to JS, and often occurs with object manipulation with JSON capabilities.

What to look out for:
- JS infrastructure
- Wherever JSON files are being handled.
  - ↳Specifically, JSON merge, JSON property def by path, or object extension.
- Certain vulnerable libraries
  - ↳lodash, deepMerge

JSON merge functions
↳lodash, deepMerge
Insecure JSON merge operations generally involve copying properties from a source object into a target object, in a JSON context

```
var merge = function(target, source) {
    for(var attr in source) {
        if(typeof(target[attr]) === "object" && typeof(source[attr]) === "object") {
            merge(target[attr], source[attr]);
        } else {
            target[attr] = source[attr];
        }
    }
    return target;
};
```

Unsafe merge functions will iterate through the source object's properties, and copy those properties onto the target object in a recursive manner.

```
var APIdata = JSON.parse('{"hello": "world", "__proto__": {"pollution":"yep"}}');
undefined
var obj = merge({foo:"bar"}, APIdata)
```

User input will usually be treated as a JSON string, which is made into a JS object via a call to JSON.parse().
If a key in the JSON string is "__proto__", JSON.parse() will correspond it to the internal __proto__ property.
The var APIdata will pollute the object wl a var called "pollution". When it's merged into the var "obj", the __proto__ property will be applied and pollution will occur.

```
obj.pollution
"yep"
```

JSON property definition by path
↳fast-json-patch
Defining property values on an object based on a given path.
Usually has a function signature like:
Function(object, path, value)
Typically, the variable 'path' will be split at "/" and the values read as object properties. If an attacker controls the path val, they can inject something like "__proto__/myVal" to perform proto pollution.

EXPLOIT USE-CASES

Proto pollution to lead to AST injections
- AST (Abstract Syntax Tree) is used in nodeJS applications with template engines. We can leverage proto pollution to inject rogue AST into the template engine, allowing for RCE.

RedpwnCTF 2021: pastebin 2
- Ability to make pastes but they're passed by DOMpurify, so we can't make any conventional XSS payload
- In the source lies an interesting function called 'parseform' which makes a JS object out of a given HTML form's properties.
  - ↳JS object and properties defined by JSON input = vulnerable to prototype pollution

Pwn2Win: Illusion
- Suspicious library: fast-json-patch
- All user-input is treated in JSON
  - fast-json-patch has a prototype pollution vuln
  - combine with ejs AST template injection (pollute with 'outputFunctionName' to achieve RCE)
- proto pollution unsafe method: property definition by path

Proto pollution poisoning environment variables to achieve RCE

If a new process using node is spawned and you can use proto pollution to poison environment variables, you can achieve RCE

It's also possible to poison environment variables by setting the env property in some JS object

This is a potential proto pollution gadget.

When a new node process is spawned, env vars will be passed to it.

Additionally, when a new node process is spawned, it may be possible to pass command-line arguments using `NODE-OPTIONS`.

You can specify a cli arg "--require <file path>" and specify proc/self/environ to load in env vars

↳ If you can control env vars using proto pollution, you can control the contents of proc/self/environment

All in all, you can set 2 env variables: one to set an env var to execute code (like open a reverse-shell) and another to set NODE-OPTIONS to load proc/self/environ.

↳ proto pollute "env" to include a variable to execute code

→ proto pollute "env" to include NODE-OPTIONS, the value of which is to "--require /proc/self/environ"

CLOSING FACTS

- Detecting proto pollution:
  - look for where user input is being passed as JSON files. Look to see if its passed to JSON.parse()
  - look for suspicious libraries being used, like lodash
  - unsafe JSON functions: object merge, path definition
  - look for where objects are being made out of JSON-treated input

- Using prototype pollution involves some property to pollute the prototype.

- proto pollution defines a property in an obj's internal prototype, polluting all objects that inherits from the prototype.

- you can use __proto__ to modify any object property.

NOTES