Prototype Pollution

@splitline

JavaScript OOP 101

```
1.  function Cat() {
2.    this.sound = 'meow!';
3.    this.meow = function () {
4.      alert(this.sound);
5.    }
6.  }
```

綁在 object 上

綁在 class 上

```
1.  function Cat() {
2.    this.sound = 'meow!';
3.  }
4.  Cat.prototype.meow = function () {
5.    alert(this.sound);
6.  }
```

JavaScript OOP 101

```
1.  function Cat() {
2.    this.sound = 'meow!';
3.  }
4.  Cat.prototype.meow = function () {
5.    alert(this.sound);
6.  }
7.  let kitten = new Cat();
```

- `Class.prototype`

  其 instance 都會有 prototype 裡面的屬性和方法。

- `instance.__proto__`

  指向所屬 class 的 prototype。

```
kitten.__proto__ === Cat.prototype
```

JavaScript OOP: 繼承

```
1.  function Animal() {
2.    this.cute = true;
3.  }
4.  function Cat() {
5.    this.sound = 'meow!';
6.  }
7.  Cat.prototype = new Animal();
8.  let kitten = new Cat();
9.  kitten.sound; // "meow!"
10. kitten.cute;  // true
```

- Class.prototype

  其 instance 都會有 prototype 裡面的屬性和方法。

- 繼承

  讓子 class 的 prototype 指向想繼承的父 class instance。

  標準做法應該是：
  object.create(Animal.prototype)

JavaScript OOP: 繼承

```
1.  function Animal() {
2.    this.cute = true;
3.  }
4.  function Cat() {
5.    this.sound = 'meow!';
6.  }
7.  Cat.prototype = new Animal();
8.  let kitten = new Cat();
9.  kitten.sound; // "meow!"
10. kitten.cute;  // true
```

JavaScript OOP: 繼承

```
1.  function Animal() {
2.    this.cute = true;
3.  }
4.  function Cat() {
5.    this.sound = 'meow!';
6.  }
7.  Cat.prototype = new Animal();
8.  let kitten = new Cat();
9.  kitten.sound; // "meow!"
10. kitten.cute;  // true
```

```
>> kitten
← ▶ { sound: "meow!" }
```

JavaScript OOP: 繼承

```
1.  function Animal() {
2.    this.cute = true;
3.  }
4.  function Cat() {
5.    this.sound = 'meow!';
6.  }
7.  Cat.prototype = new Animal();
8.  let kitten = new Cat();
9.  kitten.sound; // "meow!"
10. kitten.cute;  // true
```

```
>> kitten
← ▶ { sound: "meow!" }
>> kitten.__proto__
← ▶ { cute: true }
```

JavaScript OOP: 繼承

```
1.  function Animal() {
2.    this.cute = true;
3.  }
4.  function Cat() {
5.    this.sound = 'meow!';
6.  }
7.  Cat.prototype = new Animal();
8.  let kitten = new Cat();
9.  kitten.sound; // "meow!"
10. kitten.cute;  // true
```

```
>> kitten
← ▶ { sound: "meow!" }
>> kitten.__proto__
← ▶ { cute: true }
>> kitten.__proto__.__proto__
← ▶ Object { … }
```

JavaScript OOP: 繼承

```
1.  function Animal() {
2.    this.cute = true;
3.  }
4.  function Cat() {
5.    this.sound = 'meow!';
6.  }
7.  Cat.prototype = new Animal();
8.  let kitten = new Cat();
9.  kitten.sound; // "meow!"
10. kitten.cute;  // true
```

```
>> kitten
← ▶ { sound: "meow!" }
>> kitten.__proto__
← ▶ { cute: true }
>> kitten.__proto__.__proto__
← ▶ Object { … }
>> kitten.__proto__.__proto__.__proto__
← ▶ null
```

JavaScript OOP: 繼承

```
1.  function Animal() {
2.    this.cute = true;
3.  }
4.  function Cat() {
5.    this.sound = 'meow!';
6.  }
7.  Cat.prototype = new Animal();
8.  let kitten = new Cat();
9.  kitten.sound; // "meow!"
10. kitten.cute;  // true
```

修改它會怎樣呢？

```
>> kitten
← ▶ { sound: "meow!" }
>> kitten.__proto__
← ▶ { cute: true }
>> kitten.__proto__.__proto__
← ▶ Object { … }
>> kitten.__proto__.__proto__.__proto__
← ▶ null
```

Prototype Pollution

```
>> let user = { admin: false }
```

Prototype Pollution

```
>> let user = { admin: false }
>> user.__proto__.admin = true
>> user.admin
```

Prototype Pollution

```
>> let user = { admin: false }
>> user.__proto__.admin = true
>> user.admin
← ▶ false
```

user.admin → false
~~user.__proto__.admin → true~~

Prototype Pollution

```
>> let user = { admin: false }
>> user.__proto__.admin = true
>> user.admin
← ▶ false
>> let anotherUser = { }
>> anotherUser.admin
```

Prototype Pollution

```
>> let user = { admin: false }
>> user.__proto__.admin = true
>> user.admin
← ▶ false
>> let anotherUser = { }
>> anotherUser.admin
← ▶ true
```

```
user.admin              → undefined
user.__proto__.admin    → true
```

Prototype Pollution: 出現狀況

- 能任意操作 object 的 key & value → prototype pollution
- Set
  - Prototype Pollution in lodash (_.setWith, _.set)
  - e.g. .set('__proto__.x', 'polluted')
- Merge / Extend
  - CVE-2019-11358 (jQuery $.extend)
  - e.g. .merge({},JSON.parse('{"__proto__":{"x":"polluted"}}'))
- ......

Example: Backend

Frontend Scenario

BlackFan/client-side-prototype-pollution: Prototype Pollution and useful Script Gadgets

Realworld Cases

- HackerOne XSS (Bug Bounty)
  #986386 Reflected XSS on www.hackerone.com via Wistia embed code
- Kibana RCE (CVE-2019-7609)
  Prototype Pollution in Kibana