# PROTOTYPE POLLUTION

22 • 05 • 22

# WHOAMI

- I am Godson
- Undergraduate Student
- CTF @tamilctf
- Interested in Web Security Research

Basic Javascript

Prototype and __proto__

Arrays and Objects

Prototype Chain

Class and Constructor

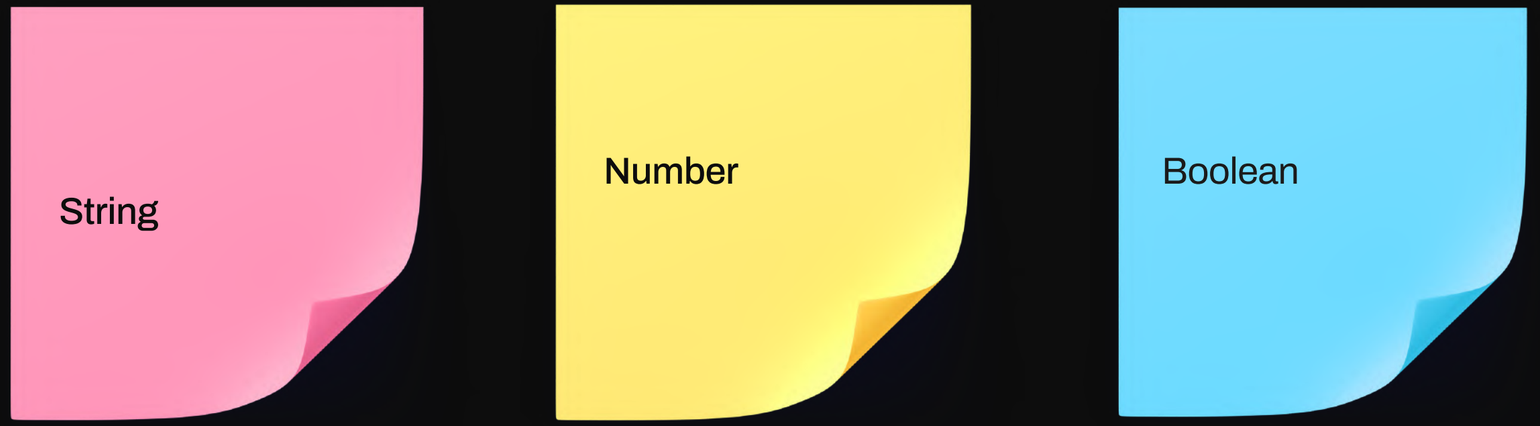
Prototype Pollution

# JAVASCRIPT

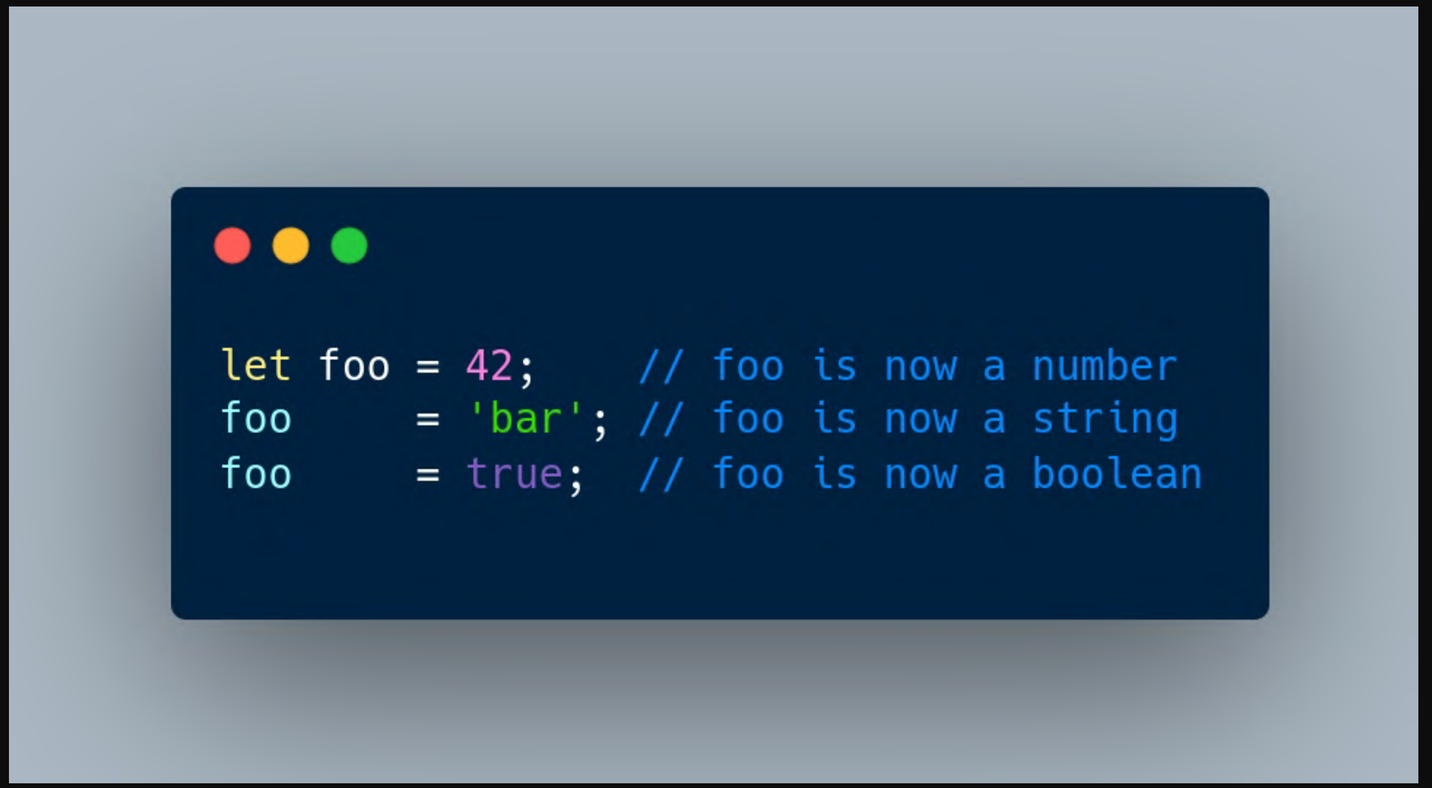
JavaScript is a text-based programming language used both on the client-side and server-side that allows you to make web pages interactive. Mainly Used for Front-End Development, Back-End Development and Game development.

# DATA TYPES

JavaScript is a *loosely typed* and *dynamic* language. Variables in JavaScript are not directly associated with any particular value type, and any variable can be assigned (and re-assigned) values of all types:



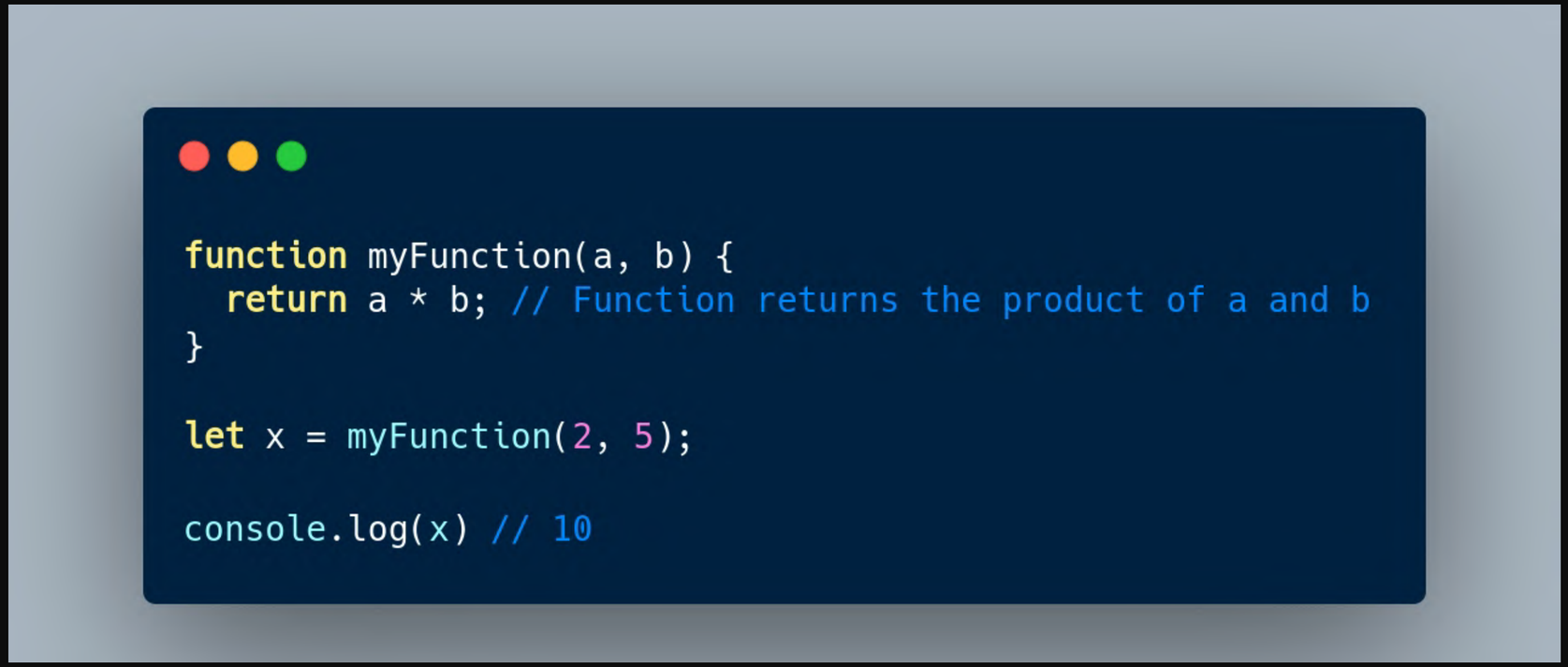
# DATA TYPES



# FUNCTIONS

*A JAVASCRIPT FUNCTION IS A BLOCK OF CODE DESIGNED TO PERFORM A PARTICULAR TASK.*

# FUNCTION



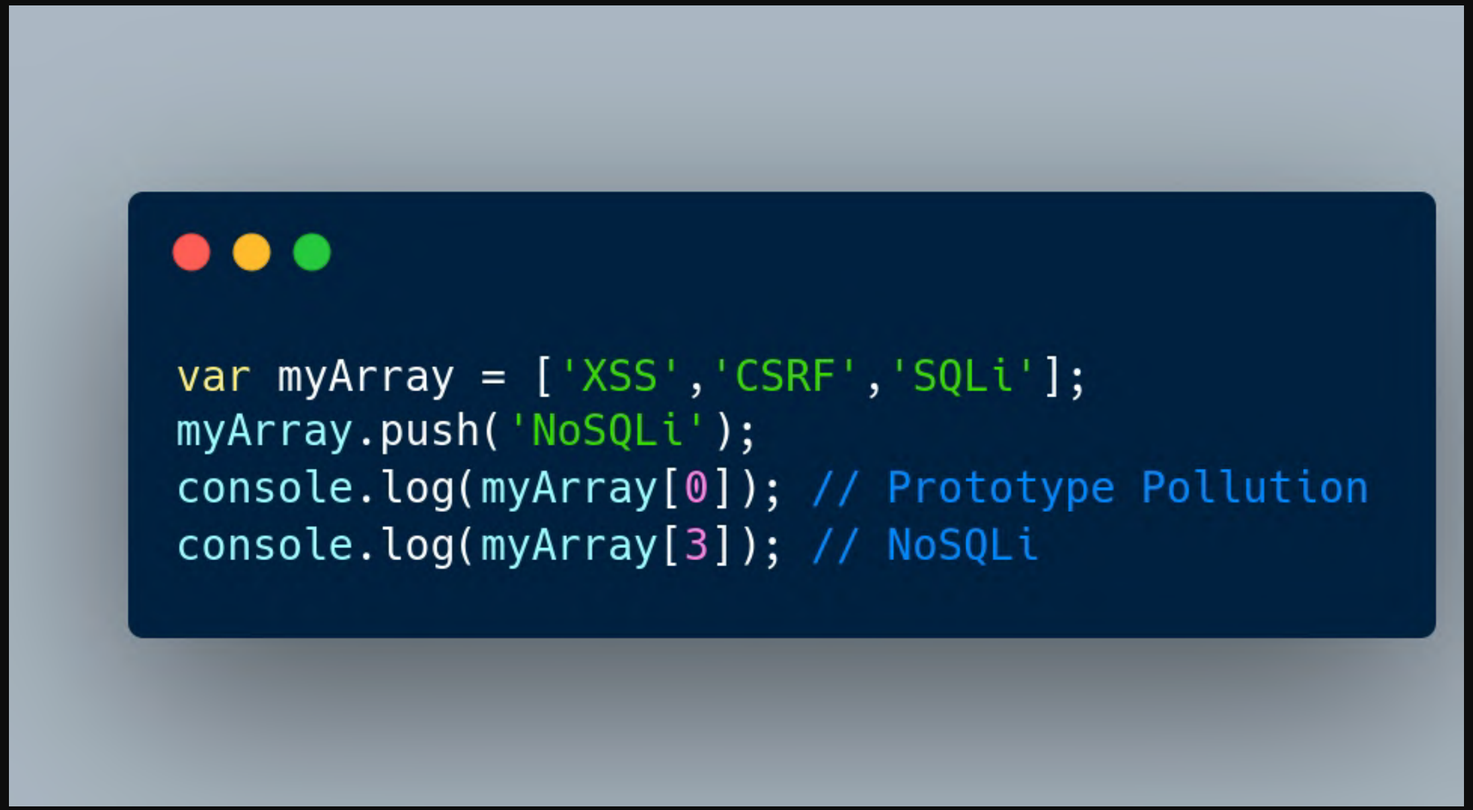
```
function myFunction(a, b) {
  return a * b; // Function returns the product of a and b
}

let x = myFunction(2, 5);

console.log(x) // 10
```

# ARRAYS AND OBJECTS

Both objects and arrays are considered “special” in JavaScript. Objects represent a special data type that is mutable and can be used to store a collection of data (rather than just a single value). Arrays are a special type of variable that is *also* mutable and can *also* be used to store a list of values.

# ARRAY

We use arrays whenever we want to create and store a list of multiple items in a single variable. Arrays are especially useful when creating **ORDERED COLLECTIONS** where items in the collection can be accessed by their numerical position in the list.



```
var myArray = ['XSS','CSRF','SQLi'];
myArray.push('NoSQLi');
console.log(myArray[0]); // Prototype Pollution
console.log(myArray[3]); // NoSQLi
```

# OBJECTS

Objects are used to represent a “thing” in your code. That could be a person, a car, a building, a book, a character in a game – basically anything that is made up or can be defined by a set of characteristics. In objects, these characteristics are called **PROPERTIES** that consist of a **KEY** and a **VALUE**.

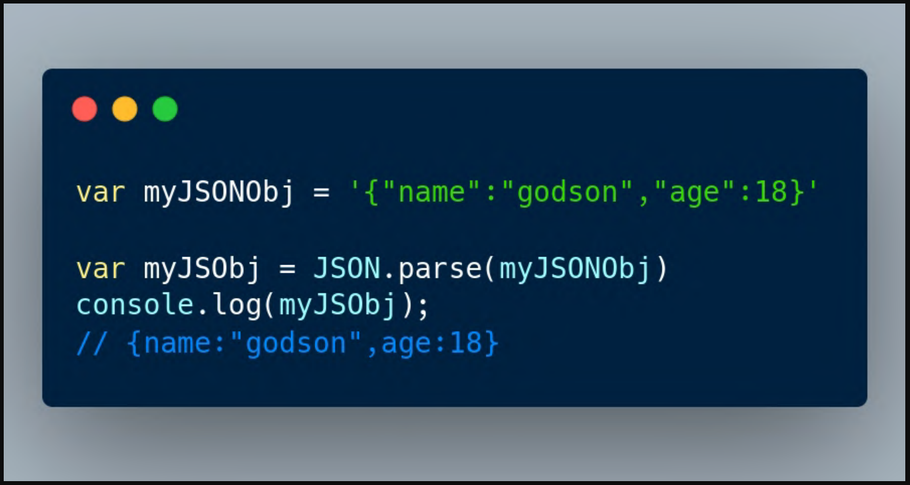
# JSON OBJECTS VS JAVASCRIPT OBJECTS

# DIFFERENCE


A Javascript object has a similar syntax to JSON, it uses curly braces and key/value pairs. The main difference in syntax is that **IN A JSON OBJECT THE KEYS MUST BE A STRING WRITTEN WITH DOUBLE QUOTES**

# JSON.PARSE()

JSON.PARSE() CAN BE USE TO CONVERT A JSON OBJECT INTO JS OBJECT



```
var myJSONObj = '{"name":"godson","age":18}'

var myJSObj = JSON.parse(myJSONObj)
console.log(myJSObj);
// {name:"godson",age:18}
```

# JSON.STRINGIFY()

JSON.STRINGIFY() CAN BE USED TO CONVERT A JS OBJECT TO JSON OBJECT

```
var myJSObj = {
  name:"godson",
  age:18
}

var myJSONObj = JSON.stringify(myJSObj)
console.log(myJSONObj);
// {"name":"godson","age":18}
```

# CONSTRUCTOR AND CLASS

The constructor() method is a special method for creating and initializing objects created within a class.

# CONSTRUCTOR

A constructor is **A SPECIAL METHOD OF A CLASS OR STRUCTURE IN OBJECT-ORIENTED PROGRAMMING THAT INITIALIZES A NEWLY CREATED OBJECT OF THAT TYPE**. Whenever an object is created, the constructor is called automatically.

```
function Person(name,age){
  this.name = name,
  this.age = age
}
var mySelf = new Person('Godson',18)
console.log(mySelf)
// Person {name: 'Godson', age: 18}
```

# CLASS

Classes are a template for creating objects. If Constructor Function is Not defined inside the Class, then Javascript will automatically Declare an empty Constructor function.

```
class Rectangle {
  constructor(height, width) {
    this.height = height;
    this.width = width;
  }
}

var rec = new Rectangle(10,20)
console.log(rec)
// Rectangle {height: 10, width: 20}
```

# PROBLEM

**Now we have a Rectangle Class which builds an Object with 2 keys 'height', and 'width'. Now, How to calculate the area of the rectangle, or how to write a method or function to return the area of the rectangle with was created by the Rectangle Class?**

## Possible Solutions?


Manually redefining the Rectangle Class with a pre-defined method inside the Rectangle Class


Manually Adding a key and value as a Call back function to an Object created with Rectangle Class. [Not applicable for Objects Created with Class Constructor]


Injecting the function to the Object Prototype which returns the area of the Rectangle

# SOLUTIONS

# PROTOTYPE AND __PROTO__

A prototype is **an object that is associated with every functions and objects by default in JavaScript**, where the function's prototype property is accessible and modifiable and the object's prototype property (aka attribute) is not visible. Every function includes prototype object by default.

# PROTOTYPE

Every object in JavaScript has a built-in property, which is called its **PROTOTYPE**. The prototype is itself an object, so the prototype will have its own prototype, making what's called a **PROTOTYPE CHAIN**. The chain ends when we reach a prototype that has **NULL** for its own prototype.

```
var myObj = {
  name:"Godson",
  sayHi: function(){console.log("Hi "+ this.name)}
}

myObj.name // Godson
myObj.sayHi() // Hi Godson
myObj.toString() // [object Object]

/*
Where the toString Function comes from??
*/
```

# __PROTO__

The **__PROTO__** function or Hidden property exposes the value of the internal **[[PROTOTYPE]]** of an object. Every Object in the Javascript has the hidden property __PROTO__

```
var myObj = {
  name:"Godson",
  sayHi: function(){console.log("Hi "+ this.name)}
}

console.log(myObj.__proto__)
// exposes the value of the internal Prototype of an object.
```

# PROTOTYPE CHAIN

**Each object has a private property which holds a link to another object called its prototype**. That prototype object has a prototype of its own, and so on until an object is reached with null as its prototype. By definition, null has no prototype, and acts as the final link in this prototype chain.

# CHAIN

```
var myObj1 = {
  name:"Godson"
}
console.log(myObj1.toString()) // [object Object]

var myObj2 = {
  name:"Godson",
  toString: function(){console.log("I am Here!")}
}
console.log(myObj2.toString()) // I am Here!

/*
In myObj2, are we Overwriting the toString? No
*/
```

FOR BETTER UNDERSTANDING, LETS SEE SOME PRACTICAL EXAMPLES

# POLLUTING THE PROTOTYPE

Prototype pollution is an injection attack that targets JavaScript runtimes. With prototype pollution, an attacker might control the default values of an object's properties. This allows the attacker to tamper with the logic of the application and can also lead to denial of service or, in extreme cases, remote code execution.

# WHY PROTOTYPE POLLUTION OCCUR

- Basically Prototype pollution occur when merging the user input with another Object.
- Merging the User Input in Unsafe Method → prototype Pollution

# BASIC EXPLOITATION


INJECTION ISADMIN = TRUE, INSIDE THE PROTOTYPE OF THE OBJECT. AS I SAID BEFORE, WE CAN ACCESS THE HIDDEN PROTOTYPE VALUES WITH THE __PROTO__ KEYWORD.

HERE, WE SET ISADMIN = TRUE, WITH THE __PROTO__ KEYWORD WHICH CREATES A NEW KEY-VALUE PAIR INSIDE THE PROTOTYPE.

# PROTOTYPE POLLUTION

## CLIENT SIDE

- PROTOTYPE POLLUTION ON THE CLIENT-SIDE CAN CHAIN TO GAIN XSS IN THAT DOMAIN. 😉

## SERVER SIDE

- PROTOTYPE POLLUTION ON THE SERVER-SIDE MAY CAUSE AUTH-BYPASS, LFI, RFI AND EVEN RCE IN SOME CASES. 😍

# LABS

# FIX?

# FIX

| | create(null) | sanitize | hasOwnProperty |
|---|---|---|---|
| Possible Fix | ✅ | ✅ | ✅ |

ALWAYS MAKE SURE TO USE POPULAR LIBRARY

# LET'S CONNECT

- 0xGodson_ 
- Godson Bastin 
- 0xGodson#0831 

# THANK YOU

# ANY QUESTIONS?