**COMP4901O Competitive Programming in Cybersecurity II**

# JavaScript Prototype Pollution

LI, Kwan To (STommydx)
March 26, 2021

>FIREBIRD CTF
TEAM

# Agenda

# Agenda (cont.)

# Introduction

# Introduction

## Before We Begin...

# Code of Ethics

- The exercises for the course should be attempted ONLY INSIDE THE SECLUDED LAB ENVIRONMENT documented or provided. Please note that most of the attacks described in the slides would be ILLEGAL if attempted on machines that you do not have explicit permission to test and attack. The university, course lecturer, lab instructors, teaching assistants and the Firebird CTF team assume no responsibility for any actions performed outside the secluded lab.
- The challenge server should be regarded as a hostile environment. You should not use your real information when attempting challenges.
- Do not intentionally disrupt other students who are working on the challenges or disclose private information you found on the challenge server (e.g. IP address of other students). Please let us know if you accidentally broke the challenge. While you may discuss with your friends about the challenges, you must complete all the exercises and homework by yourselves.

# Expectations

Hopefully most of you can learn something new...

- A guide for you to learn
- More on basics concepts
- Some easy hands-on examples
- Advanced concepts or application won't be covered (better to explore by yourself!)

In case you are familiar with JavaScript frameworks and/or the prototype pollution exploit, please help other fellow students in case they have problems. This presentation might be a little bit boring for you... (I apologize for that.)

# Resource Repository

All codes and the slides used today can be found in the following repository.

https://github.com/STommydx/comp4901o-presentation

Please clone the repository if you would like to follow the slides and run the codes locally.

# Introduction

## CTF in the Web

# The Web Category

You should have seen them before. Right?

- Appears in every CTF!
- Harder than pwn? (depends on people)
- Easy to get in. Difficult to master.

# Previously...

Did you solve some of them?

- Intro CTF
  - Baby Web Developer, QuestionAIR, Real Firebird Website, ...
- HKCERT CTF
  - Rickroll, Conversion Center, ...
- Internal CTF
  - token101, Boku no Firebird CTF, URL Shortener, ...
- Zer0pts CTF
  - Kantan Calc, Simple Blog, ...

# Weapons

We have learnt quite a number of web attacks in COMP 4901N.

**Example**

- XSS
- LFI
- XXE
- SQL Injection
- Command Injection
- and more...

Which one is your favourite? Type in the chatbox :)

# Introduction

## Why JavaScript?

# The Web Stack

In the old days...

- Frontend: Static HTML
- Styling: CSS
- Backend: PHP, JSP
- Communication: HTTP Form Data

# The Web 3.0

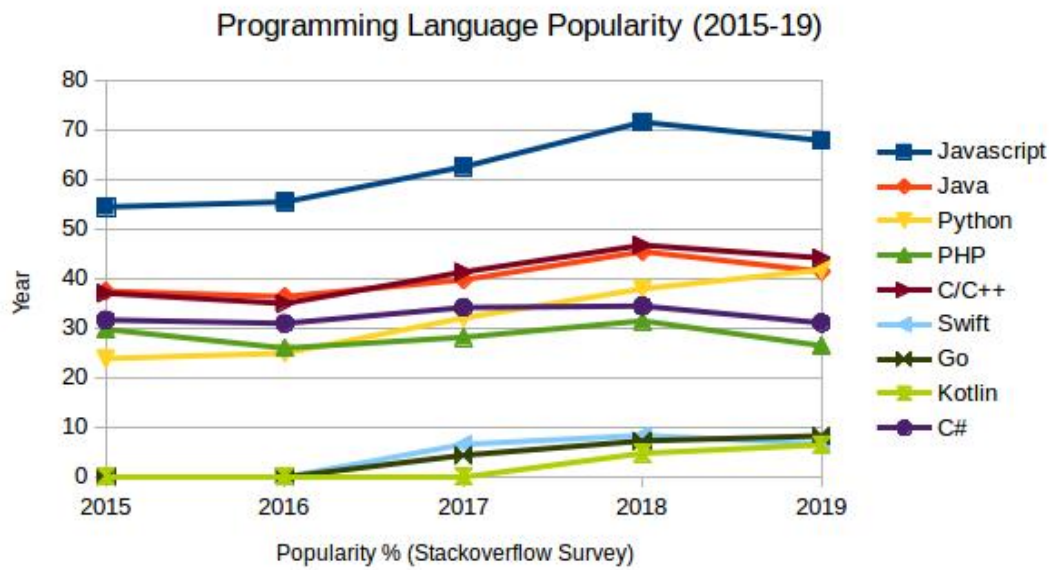
One language for everything!

- Frontend: DHTML with JavaScript
- Styling: dynamically styled components with JS, SASS compiled to CSS by JS
- Backend: Node.js
- Communication: JSON APIs

Both good news and bad news...

# JavaScript Popularity

Of course, there is no single language that is the most popular. Different websites already give different ranking results. However, JavaScript is in the top list in general.




https://duckduckgo.com/?q=programming+language+popularity+chart&t=h_&iax=images&ia=images

# JavaScript Attack Surface

JavaScript is too powerful!

- Access to local storage data (cookies)
- Network APIs
- Dynamically typed language
- Unsafe APIs and more...

# JavaScript in CTF

You have to face it...

**Example**

- Internal CTF: token101
  - A Angular + Express.js full-stack web app
- Zer0pts CTF: PDF Generator
  - A Vue.js + Express.js full-stack web app

No worries! We will cover some basics today. :)

# Introduction

## Tools & Environment

# The Browser

The developer tools in the browser is powerful. It is good for trying out things quickly.

- Browser JavaScript Console
  - Prototyping
  - Inspect variables
- Browser Network Tab
  - Inspect network traffic
  - Quick edit & resend request

# Node.js

There are a lot of JavaScript web frameworks.

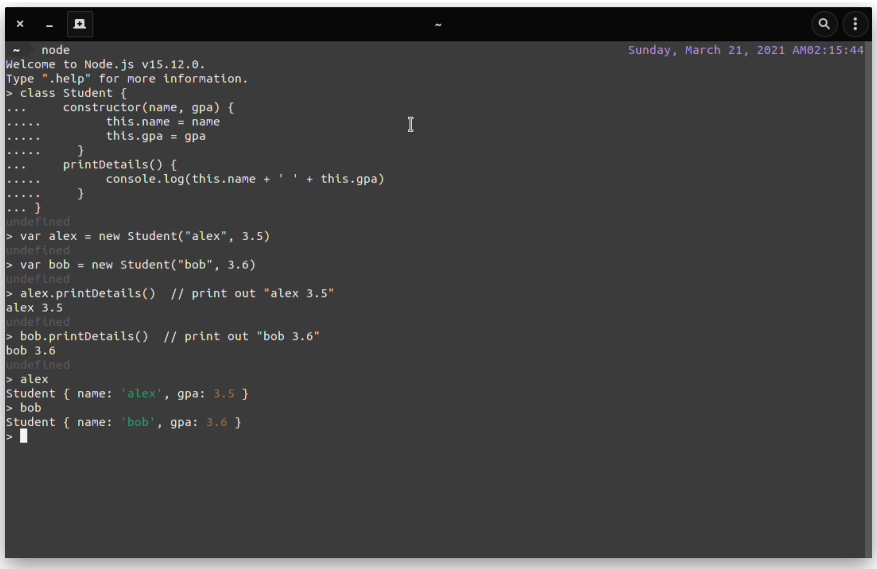
**Example**

- Frontend
  - React.js
  - Vue.js
  - Angular
- Backend
  - Express.js

You may want to deploy some of the web applications in CTF challenges. Let's have a short tutorial on how to run a Node.js server application.

# Server-side JavaScript

Node.js is a back-end JavaScript runtime environment that can be run without your browser.



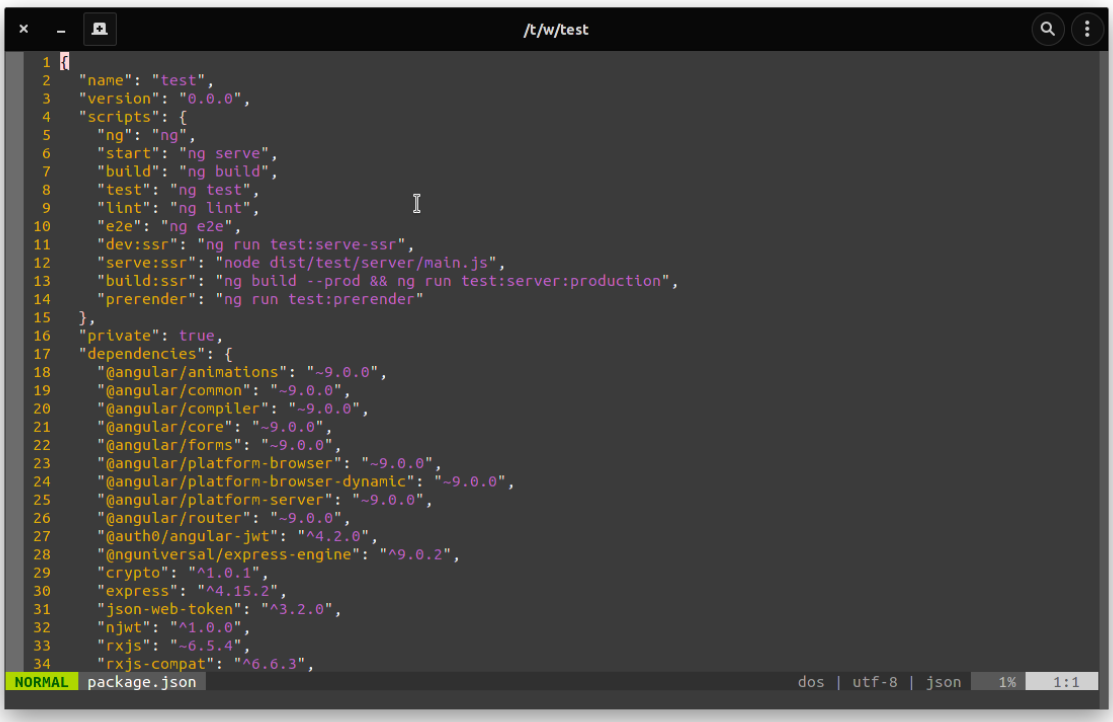

Please check the download page for the instruction of your Linux distribution. You can just use the package from offical Kali repo to install in Kali Linux[1].

https://nodejs.org/en/download/package-manager/

[1] `sudo apt install npm`

# Node Package Manager

A typical Node.js project requires library packages. The packages must be installed before running the application.

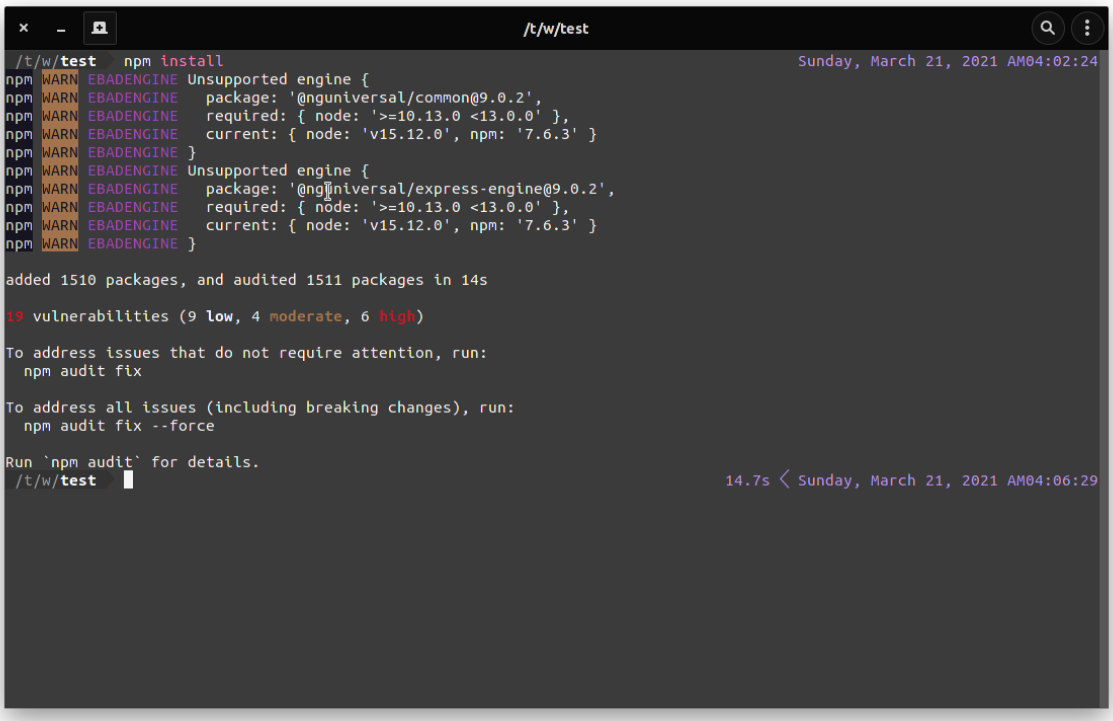



```
{
  "name": "test",
  "version": "0.0.0",
  "scripts": {
    "ng": "ng",
    "start": "ng serve",
    "build": "ng build",
    "test": "ng test",
    "lint": "ng lint",
    "e2e": "ng e2e",
    "dev:ssr": "ng run test:serve-ssr",
    "serve:ssr": "node dist/test/server/main.js",
    "build:ssr": "ng build --prod && ng run test:server:production",
    "prerender": "ng run test:prerender"
  },
  "private": true,
  "dependencies": {
    "@angular/animations": "~9.0.0",
    "@angular/common": "~9.0.0",
    "@angular/compiler": "~9.0.0",
    "@angular/core": "~9.0.0",
    "@angular/forms": "~9.0.0",
    "@angular/platform-browser": "~9.0.0",
    "@angular/platform-browser-dynamic": "~9.0.0",
    "@angular/platform-server": "~9.0.0",
    "@angular/router": "~9.0.0",
    "@auth0/angular-jwt": "^4.2.0",
    "@nguniversal/express-engine": "^9.0.2",
    "crypto": "^1.0.1",
    "express": "^4.15.2",
    "json-web-token": "^3.2.0",
    "njwt": "^1.0.0",
    "rxjs": "~6.5.4",
    "rxjs-compat": "^6.6.3",
```

The packages required are specified in `package.json`.

# Installing the Packages

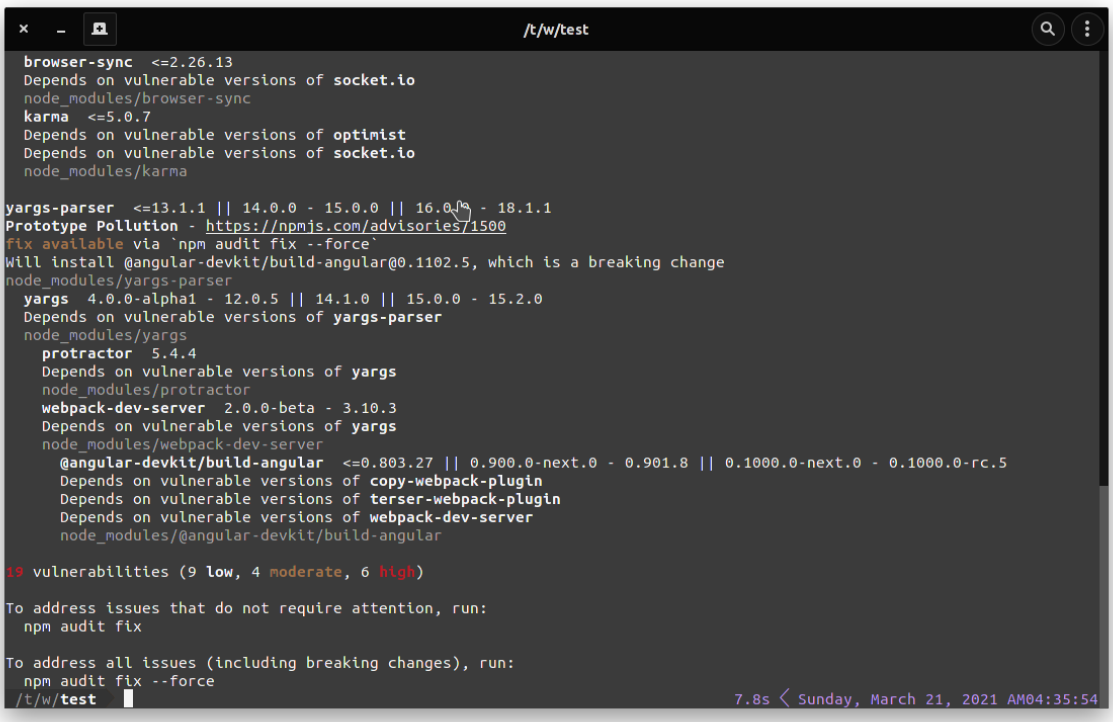
To install all dependencies required in `package.json`, run `npm install`.




If the file `yarn.lock` exists, the app uses the yarn package manager instead of npm. Then, you should run `yarn install` instead.

# NPM Audit

You may want to run `npm audit` to see the vulnerabilities that exist in the packages.



# Running the App

Read `package.json`! It varies between different apps. Common scripts include:

- `npm run start`
- `npm run dev`
- `npm run build`
- `npm run serve`

If a Dockerfile or a start-up script is available, do read it to see how the application should be spinned up.

# Burp Suite

Our good old friend Burp Suite. I assume you guys are familiar with that...

- Inspecting requests
- Intercept and modify requests
- Sending saved requests

# Object-oriented Programming in JavaScript

# Object-oriented Programming in JavaScript

## Preliminaries

# Preliminaries

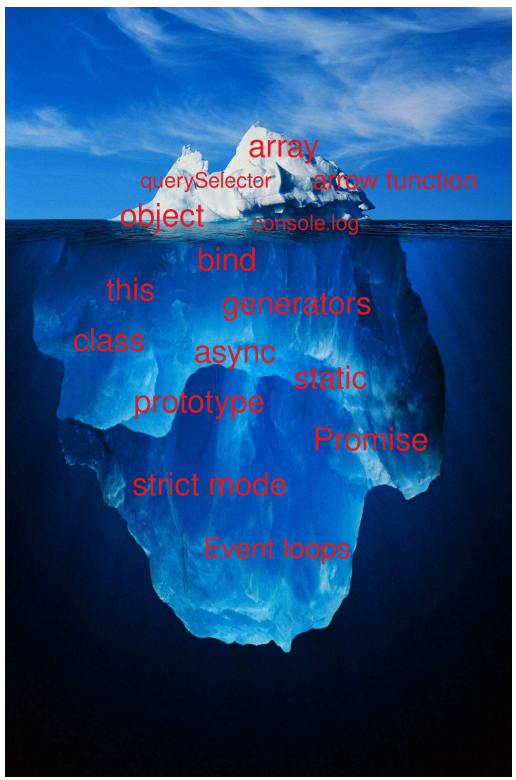
From COMP 2011/2012H and COMP 4901N, I assume you know the following...

- Understanding of Basic OOP Concepts
  - Classes & Objects
  - Inheritance
- Understanding of Simple JavaScript

Feel free to stop me if you got stuck on some basic concepts.

# Basics are important!

It is important to understand the JavaScript language before getting into the actual exploit. So please bear with me!



# Object-oriented Programming in JavaScript

## JavaScript Class

# Class Syntax of JavaScript

Have you ever tried using class in JavaScript? It is much simpler than you may think. :)

**Example (Class Definition)**

```
class Student {
    constructor(name, gpa) {
        this.name = name
        this.gpa = gpa
    }
    printDetails() {
        console.log(this.name + ' ' + this.gpa)
    }
}
```

# Class Syntax of JavaScript

To create an object with an specific class, you will use the `new` keyword like you do in most other OOP languages. The way to invoke a method is also similar.

**Example (Object Creation)**

```
var alex = new Student("alex", 3.8)
var bob = new Student("bob", 3.6)
alex.printDetails()  // print out "alex 3.8"
bob.printDetails()  // print out "bob 3.6"
```

# Class Getter and Setter

The getter and setter is a little bit similar to Kotlin. You do not need to create a method with a long name `getMyField()` and invoke it. Instead, you can access the field like a variable and the corresponding getter and setter will be called when you access or modify the variable.

## Example (Getter)

```
class Student {
    constructor(name, gpa) { this.name = name; this.gpa = gpa }
    get honor() {
        if (this.gpa >= 3.5) return "1st"
        if (this.gpa >= 2.15) return "2nd"
        if (this.gpa >= 1.5) return "3rd"
        return "pass"
    }
}
var alex = new Student("alex", 3.8)
console.log(alex.honor)  // print out "1st"
```

# Class Getter and Setter

## Example (Setter)

```
class Student {
    constructor(name, gpa) { this.name = name; this.gpa = gpa }
    set fullName(fullName) {
        this.name = fullName.split(' ')[0]
    }
}
var alex = new Student("alex", 3.5)
alex.fullName = "Alex Lee"
console.log(alex.name)  // print out "Alex"
```

One way to execute prototype pollution is through the `__proto__` getter and setter. We will discuss this later.

# Static Method and Properties

You should know `static` from Java or C++. One main difference between this and Java is static fields and method cannot be directly accessed from the object.

**Example (Static Properties)**

```
class Student {
    static maxGpa = 4.3
    constructor(name, gpa) { this.name = name; this.gpa = gpa }
    static printMax() { console.log(this.maxGpa) }
}
var alex = new Student('Alex', 3.8)
console.log(Student.maxGpa)  // print out 4.3
console.log(alex.maxGpa)  // print out undefined
Student.printMax()  // print out 4.3
// alex.printMax()  // fail to run
```

Be extra careful about the binding of `this`. It is binded to the class object for static methods.

# Inheritance

We will start with a simple example. Let's say hi to the `extends` keyword! :)

## Example (Simple Inheritance)

```
class Student {
    constructor(name, gpa) { this.name = name; this.gpa = gpa }
    printDetails() { console.log(this.name + ' ' + this.gpa) }
}
class UGStudent extends Student {
    constructor(name, gpa, fyp) { super(name, gpa); this.fyp = fyp }
    printFypDetails() { console.log(this.name + ' is doing ' + this.fyp) }
}
var alex = new UGStudent('Alex', 3.8, 'Barcode Scanner App')
alex.printDetails()  // print out "Alex 3.8"
alex.printFypDetails()  // print out "Alex is doing Barcode Scanner App"
```

Nothing really special. You can access all fields and methods of the parent class. (Note that JavaScript don't have private fields/methods[2].)

[2] Private field support is still in proposal. Support in browsers is limited but the feature can be used through polyfill or transpiler like Babel.

# Inheritance

We can use the `instanceof` operator to check whether an object is an instance of a class. Also, the `constructor` property of the object would return the class of the object.

## Example (Class Relations)

```
var a = new UGStudent('Alex', 3.8, 'Barcode Scanner App')
var b = new Student('Betty', 3.5)
console.log(a instanceof Object, b instanceof Object)  // true true
console.log(a instanceof Student, b instanceof Student)  // true true
console.log(a instanceof UGStudent, b instanceof UGStudent)  // true false
console.log(a.constructor == Object, b.constructor == Object)  // false false
console.log(a.constructor == Student, b.constructor == Student)  // false true
console.log(a.constructor == UGStudent, b.constructor == UGStudent)  // true false
```

# Inheritance

There is not really overridding in JavaScript. JavaScript will always lookup the method from the class object directly and only lookup the parent class in case the method doesn't exist.

## Example (Overridding)

```
class Student {
    constructor(name, gpa) { this.name = name; this.gpa = gpa }
    toString() { return this.name + ' ' + this.gpa }
}
class UGStudent extends Student {
    constructor(name, gpa, fyp) { super(name, gpa); this.fyp = fyp }
    toString() { return this.name + ' is doing ' + this.fyp }
}
var alex = new UGStudent('Alex', 3.8, 'Barcode Scanner App')
console.log(alex + '') // print out "Alex is doing Barcode Scanner App"
```

In the example, we have no simple way to invoke the Student class `toString()` method for alex. The `toString()` method in UGStudent will always stop the lookup process.

# Object-oriented Programming in JavaScript

## Function Object as Class

# Class or Function?

So, what is a class? Why the `constructor` property gives us back a class instead of a function?

## Example (Class Type Checking)

```
class Student {
    constructor(name, gpa) { this.name = name; this.gpa = gpa }
    printDetails() { console.log(this.name + ' ' + this.gpa) }
}
console.log(Student instanceof Function)  // true
console.log(Student.constructor == Function)  // true
var alex = new Student("alex", 3.5)
console.log(alex.constructor instanceof Function)  // true
console.log(alex.constructor.constructor === Function)  // true
```

It indeed gives us back a function. :)

# Function and Object Creation

Lets rewrite our class using Function!

## Example (Function as Classes)

```
function Student(name, gpa) {
    this.name = name; this.gpa = gpa
    this.printDetails = function() {
        console.log(this.name + ' ' + this.gpa)
    }
}
var alex = new Student("alex", 3.5)
console.log(alex instanceof Student)  // true
alex.printDetails()  // alex 3.5
```

Note that it is not exactly the same. Now, the method is copied to each of the Student object. We will mitigate this after introducing prototypes.

# The 'new' keyword

The `new` keyword should be used with a constructor function. It will do the following when invoked:

1. Create a new blank object.
2. Binds the newly created object instance as the `this` context.
3. Execute the constructor function with the specified arguments.
4. Return the object created as the result of the operator.

# Classes are new!

In fact, JavaScript classes are from ES6. Classes do not even exist in JavaScript in the old days!
To support old browser like IE, we can transpile the new class syntax to the old function syntax using Babel.

https://babeljs.io/repl

Let's take a look at the transpiled output for our Student class.

# Object-oriented Programming in JavaScript

## Prototype-based OOP

# The 'prototype' Property

To define common properties for a class (such as class methods), we would define the `prototype` property for the class. We would then be able to access the field from all objects of the class.

**Example (Prototype Property)**

```
function Student(name, gpa) { this.name = name; this.gpa = gpa }
Student.prototype.printDetails = function() {
    console.log(this.name + ' ' + this.gpa)
}
var alex = new Student("alex", 3.8)
var bob = new Student("bob", 3.6)
alex.printDetails()  // alex 3.8
bob.printDetails()  // bob 3.6
```

# The 'prototype' Property

When defining a class method for a class, it will be added to `prototype` property.

**Example (Prototype Property for Classes)**

```
class Student {
    constructor(name, gpa) { this.name = name; this.gpa = gpa }
    printDetails() { console.log(this.name + ' ' + this.gpa) }
}
console.log(Student.prototype) // {constructor: f, printDetails: f}
var alex = new Student("alex", 3.8)
console.log(alex.printDetails === Student.prototype.printDetails) // true
```

We can try to understand the transpiled code from Babel now.

# The [[prototype]] Property

But why setting the prototype of the constructor works? We can't see any of the shared properties in the objects... And JavaScript is not recording the type of the Object!

The magic is the hidden `[[prototype]]` property that exists in every object. When an object is created, it will automatically link the `[[prototype]]` property to the constructor's `prototype` property.

When the property (field or methods) is not found in the object, the `[[prototype]]` property will be looked up and the respective property in `[[prototype]]` will be used.

# The [[prototype]] Property

To get the `[[prototype]]` property of an object, one can use `Object.getPrototypeOf()` static method of the `Object` class.

## Example (getPrototypeOf Method)

```
class Student {
    constructor(name, gpa) { this.name = name; this.gpa = gpa }
    printDetails() { console.log(this.name + ' ' + this.gpa) }
}
var alex = new Student("alex", 3.8)
console.log(Object.getPrototypeOf(alex)) // {constructor: f, printDetails: f}
console.log(Object.getPrototypeOf(alex) === Student.prototype) // true
```

# The 'new' keyword (Revised)

Let's revise the previous slide.

1. Create a new blank object.
2. Binds the newly created object instance as the `this` context.
3. **Adds a property [[prototype]] that links to the constructor function's prototype object.**
4. Execute the constructor function with the specified arguments.
5. Return the object created as the result of the operator.

# Inheritance and The Prototype Chain

That's not the end of the story. To fully support inheritance, it not only look from the properties `[[prototype]]` but also `[[prototype]].[[prototype]]` and `[[prototype]].[[prototype]].[[prototype]]` (might be deeper too!).

## Example (Resolving the Chain)

```
class Student { constructor(name, gpa) { this.name = name; this.gpa = gpa } }
class UGStudent extends Student {
    constructor(name, gpa, fyp) { super(name, gpa); this.fyp = fyp } }
var alex = new UGStudent('Alex', 3.8, 'Barcode Scanner App')
console.log(Object.getPrototypeOf(alex) === UGStudent.prototype)  // true
console.log(Object.getPrototypeOf(Object.getPrototypeOf(alex))
            === Student.prototype)  // true
console.log(Object.getPrototypeOf(Object.getPrototypeOf(Object.getPrototypeOf(alex)))
            === Object.prototype) // true
console.log(alex.toString === Object.prototype.toString) // true
```

When invoking toString method from a `UGStudent` object, it finally resolved to `Object.prototype.toString` after finding no record in `Student.prototype.toString` and `UGStudent.prototype.toString`.

# Inheritance and The Prototype Chain

Sometimes, we won't be able to find the property we need even with a long chain. The search will stop when it reaches the end of the chain - `Object.prototype.[[prototype]]`.

## Example (End of Chain)

```
var alice = {}
console.log(alice.whatever)  // undefined
console.log(Object.getPrototypeOf(alice) === Object.prototype) // true
console.log(Object.getPrototypeOf(Object.prototype)) // null
```

The end of the chain would be `null`. It will return `undefined` for the properties not beingg found.

# Prototype Pollution

# Prototype Pollution

## The __proto__ Property

# The __proto__ Property

Normally, we will access the hidden `[[prototype]]` property by `Object.getPrototypeOf()`. However, in most JS engine implementations, a convenient getter and setter of the `[[prototype]]` property is implemented.

**Example (The __proto__ Property)**

```
class Student {
    constructor(name, gpa) { this.name = name; this.gpa = gpa }
    printDetails() { console.log(this.name + ' ' + this.gpa) }
}
var alex = new Student("alex", 3.8)
console.log(alex.__proto__ === Object.getPrototypeOf(alex)) // true
console.log(alex.__proto__ === Student.prototype) // true
```

# The __proto__ Property

The `__proto__` property allow the direct access of the `[[prototype]]` property for backward compatibility. The use of it is discouraged and the methods `Object.getPrototypeOf()` and `Object.setPrototypeOf()` should be used instead.

## Example (Modifying __proto__)

```
class Student { constructor(name, gpa) { this.name = name; this.gpa = gpa } }
var alex = new Student("alex", 3.8)
alex.__proto__.defer = true  // Student.prototype.defer = true
var bob = new Student("bob", 3.5)
console.log(alex.defer, bob.defer)  // true true
console.log(alex instanceof Student, bob instanceof Student) // true true
alex.__proto__ = {}
console.log(alex.defer, bob.defer)  // undefined true
console.log(alex instanceof Student, bob instanceof Student) // false true
bob.__proto__ = null
console.log(alex instanceof Object, bob instanceof Object) // true false
```

# Prototype Pollution

## Attack Surface

# Property Access

Most vulnerabilities are based on the fact that the user can arbitrarily control the `__proto__` property of an object. For example, a unpatched `JSON.parse()` may allow user to modify the prototype.

**Example (Accessing The __proto__ Property)**

```
const alice = { user: "alice", phone: { mobile: 98765432, home: 23456789 } }
const bob = { user: "bob", phone: { mobile: 98769786, home: 23452345 }, isAdmin: 1 }
function isAdmin(user) { return Boolean(user.isAdmin) }
function updateDetails(user, field, subfield, value) {
    if (field != "isAdmin") user[field][subfield] = value; }
console.log(isAdmin(alice), isAdmin(bob))  // false true
updateDetails(alice, "phone", "mobile", 22334455)  // normal user input
updateDetails(alice, "__proto__", "isAdmin", 22334455)  // malicious user input
console.log(isAdmin(alice), isAdmin(bob))  // true true
const cathy = { user: "cathy", phone: { mobile: 98766666, home: 23455555 } }
console.log(isAdmin(cathy))  // true
```

If we can control `Object.prototype`, we can arbitrarily give default values of any field for all objects!

# Unsafe Merge

When dealing with merge, we need to be aware when the `__proto__` property is explicitly defined.

## Example (Unsafe Merge)

```
function merge(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;
};
```

The example will copy each of the property to the target object recursively. If the target object in an direct instance of Object, that is, its `__prototype__` is Object.prototype, we would be able to pollute almost every possible object!

# Unsafe Merge

Let's see the merge in action.

**Example (Unsafe Merge with Malicious JSON)**

```
const user_input = '{ "course": "COMP4901O", "__proto__": { "isAdmin": true } }'
const alice_info = { user: "alice", phone: { mobile: 98765432, home: 23456789 } }
const alice_academics = { gpa: 3.2 }
merge(alice_academics, JSON.parse(user_input))
console.log(alice_info.isAdmin)  // true
```

Remember to keep your merge library up to date! Anyway, it is quite interesting people use a library for few lines of code (just JavaScript stuffs...).

# Prototype Pollution

## Demo

# Be ethical!

Before the demo, I would like to re-emphasize....

**Please DO NOT attempt to attack others without permission!**

The demo setup is conducted in a controlled environment in my local computer.

# Simple Privilege Escalation

I cloned a random web application from GitHub. Let's attack it!


https://github.com/Kirill89/prototype-pollution-explained

# Ideas

Some ideas to get you started (just in case you got stuck).

- express will parse JSON body and store in `req.body`
- unsafe `_.merge` from lodash@4.17.4
- the delete access is controlled by `user.canDelete`

# Prototype Pollution

## Countermeasure

# Prototype Freezing

The `Object.freeze()` method freezes an object. A frozen object can no longer be changed. In this case, we may freeze `Object.freeze(Object.prototype)` to mitigate the input.

## Example (Prototype Freezing)

Let's rerun the example above with `Object.freeze(Object.prototype)`.

```
Object.freeze(Object.prototype)
const alice = { user: "alice", phone: { mobile: 98765432, home: 23456789 } }
const bob = { user: "bob", phone: { mobile: 98769786, home: 23452345 }, isAdmin: 1 }
function isAdmin(user) { return Boolean(user.isAdmin) }
function updateDetails(user, field, subfield, value) {
    if (field != "isAdmin") user[field][subfield] = value; }
console.log(isAdmin(alice), isAdmin(bob))  // false true
updateDetails(alice, "phone", "mobile", 22334455)  // normal user input
updateDetails(alice, "__proto__", "isAdmin", 22334455)  // malicious user input
console.log(isAdmin(alice), isAdmin(bob))  // false true
const cathy = { user: "cathy", phone: { mobile: 98766666, home: 23455555 } }
console.log(isAdmin(cathy))  // false
```

# Object.create(null)

The magic function `Object.create(null)` will create an object without any `[[prototype]]`!

## Example (Object Without Prototype)

```
var haha = Object.create(null)
console.log(haha.__proto__)  // undefined
console.log(haha.constructor)  // undefined
```

## Example (Unsafe Merge Maybe Okay?)

```
const user_input = '{ "course": "COMP4901O", "__proto__": { "isAdmin": true } }'
const alice_info = { user: "alice", phone: { mobile: 98765432, home: 23456789 } }
const alice_academics = Object.create(null)
merge(alice_academics, { gpa: 3.2 })
merge(alice_academics, JSON.parse(user_input))
console.log(alice_info.isAdmin)  // undefined
```

For normal object, the prototype chain would be `obj -> Object.prototype -> null`. The magic would have the prototype chain as `obj -> null` where `[[prototype]]` is directly pointing to `null`.

# Schema Validation

For your server-side program, make sure you validate user's input!

- Users are always evil!
- Filter out any unneeded properties.
- Most recommended way to fix (IMO).

You may want to use a specialized library for that.

# Prototype Pollution in Practice

# Prototype Pollution in Practice

## Prototype Pollution in Real World

# Types of Exploits

The most common type of prototype pollution exploits include the following:

- Bypass sanitization
- XSS
- RCE
- and more...

Script gadgets:
https://github.com/BlackFan/client-side-prototype-pollution

# Exploits in the Wild

It indeed exists in real-world applications! Do update your packages regularly to patch those nesty vulnerabilities.[3]

- Lodash Library
  - Highly popular utils library!
  - Unsafe merge operations
  - https://snyk.io/vuln/SNYK-JS-LODASH-73638
- Kibana
  - You may have tried to use this in 3632/4632 before...
  - https://research.securitum.com/prototype-pollution-rce-kibana-cve-2019-7609/
- You can search more from here
  - https://github.com/advisories?page=1&query=Prototype+Pollution

[3] Please ask the author to update instead of attacking in case you find some apps that is using the affected version!

# Exercise: Bypassing Sanitizers

Let's have an exercise so you guys won't get bored...

https://securitymb.github.io/xss/4

Of course, there won't be enough time for you to complete in class time... (maybe not?)

# Exercise: Bypassing Sanitizers

You can refer to the below URLs for some of the XSS payloads.

- https://portswigger.net/web-security/cross-site-scripting/cheat-sheet
- https://research.securitum.com/prototype-pollution-and-bypassing-client-side-html-sanitizers/

Let me demonstrate how to bypass one of the sanitizer. :)

# Frame Title

You may read other's payload to learn more.

https://twitter.com/SecurityMB/status/1291283381439913984

Good luck and have fun! And... be ethical of course.

# Prototype Pollution in Practice

## Prototype Pollution in CTF

# zer0pts CTF 2021 - PDF Generator

- Offical Writeup
  - https://blog.s1r1us.ninja/CTF/zer0ptsctf2021-challenges
- Writeup of an Unintended Solution
  - https://github.com/aszx87410/ctf-writeups/issues/23

# Related Attacks

# Related Attacks

## Sandbox Escape

# The vm Node.js Module

The Node.js vm module allow executing scripts in a separate context (unlike eval).
https://nodejs.org/api/vm.html

# Escaping the VM

With the constructor property, we may get the Function object outside the vm and execute outside the vm context.
https://gist.github.com/jcreedcmu/4f6e6d4a649405a9c86bb076905696af

# Not really Sandbox Escape

I would like to share a CTF challenge that sandbox escape may not work.

- zer0pts CTF 2021 - Kantan Calc
- Node.js vm module
- Our old friend `Object.create(null)` as context
- Writeup: https://github.com/aszx87410/ctf-writeups/issues/22

# Conclusion

# Conclusion

## Final Words

# Final Words

I know this presentation is getting long...

- **Be ethical!**
- You can't master a language in a single day...
- Knowing where to start is important!

Good luck on future hacking!

# Thank You

# Thank you!

Any questions?

# Conclusion

## Resources

# Documentation

Reading language and API references is a great way to learn a language.

- MDN
  - https://developer.mozilla.org/en-US/docs/Web/JavaScript
- Node.js Doc
  - https://nodejs.org/api/

# Supplementary Resources

This two articles explained JavaScript classes and the prototype pollution exploit with quite a lot of details.

- https://github.com/HoLyVieR/prototype-pollution-nsec18/blob/master/paper/JavaScript_prototype_pollution_attack_in_NodeJS.pdf
- https://www.exploit-db.com/docs/49328