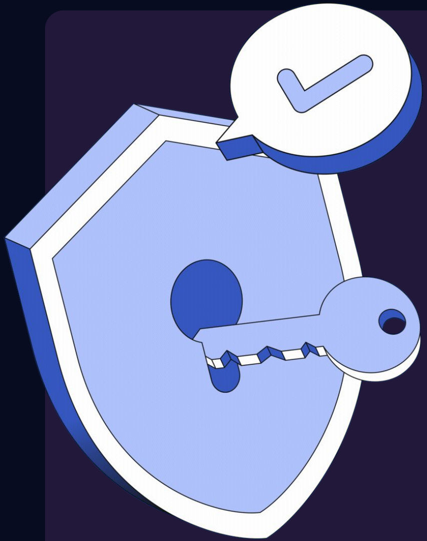
Secure Coding:

Code With Security In Mind

Session 2

Presented by: Hassan Al Achek



Who am i?

Hassan Al Achek

- Red Teamer
- CyberSecurity Researcher
- Offensive Cyber Security Engineer, Passionate about the inner workings of computers ;)
- Focused on web security, network security, reverse engineering, and malware analysis
- Attracted by state-sponsored threat actors 😉
- Moderator of Lebanon's largest cybersecurity community
- Instructor and penetration tester at Semicolon Academy
- Former Red Teamer at Covéa
- Ingénieur Civil des Mines (ICM) - École des Mines de Saint-Étienne
- Electrical and Telecommunications Engineer - Lebanese University

Agenda

01 Broken Access Control

Broken Access Control

- Within the application’s core security mechanisms, access controls are logically built on **authentication** and **session management**.

- **Authentication:** verify a user’s identity.

- **Session management:** confirm that a particular sequence of requests that the web application receives originated from the same user.

Broken Access Control |Continued

- The reason that the application needs to do these things is because it needs a way to decide whether it should **permit** a given request to perform its attempted action or access the resources it is requesting.

- **Access controls** (or **authorization**) are responsible for making these key decisions.

- Access control vulnerabilities are conceptually simple: **The application lets you do something you shouldn’t be able to**.

Broken Access Control |Continued

Access controls can be divided into three broad categories:

- ❖ **Vertical:** allow different types of users to access different parts of the application’s functionality. **Example:** division between normal users and administrators.

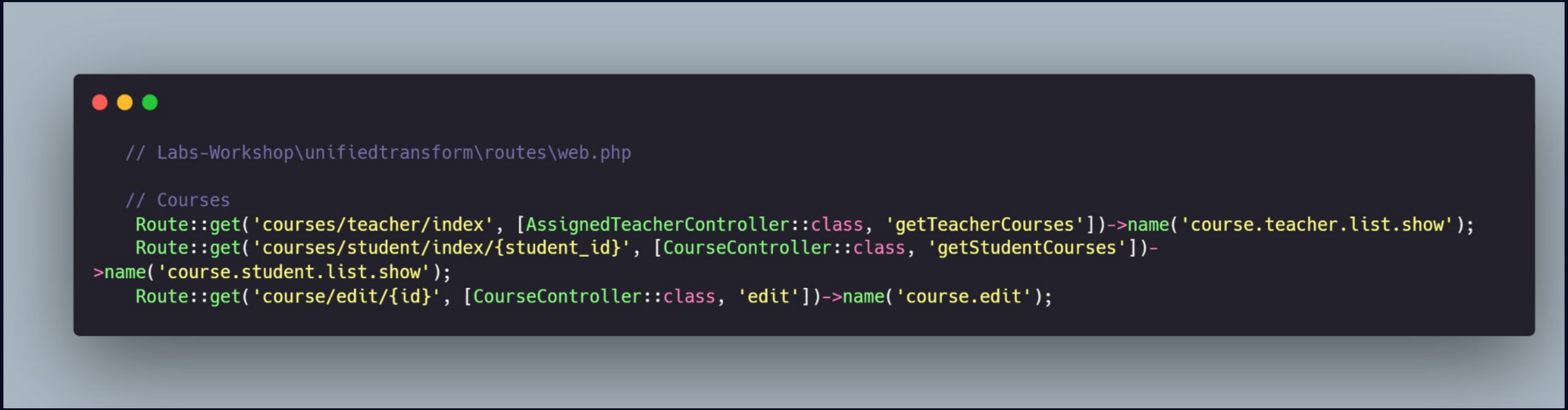
- ❖ **Horizontal:** allow users to access a certain subset of a wider range of resources of the same type. **Example:** a web mail application may allow you to read your email but no one else’s.

Scenario #1 - Real World Scenario

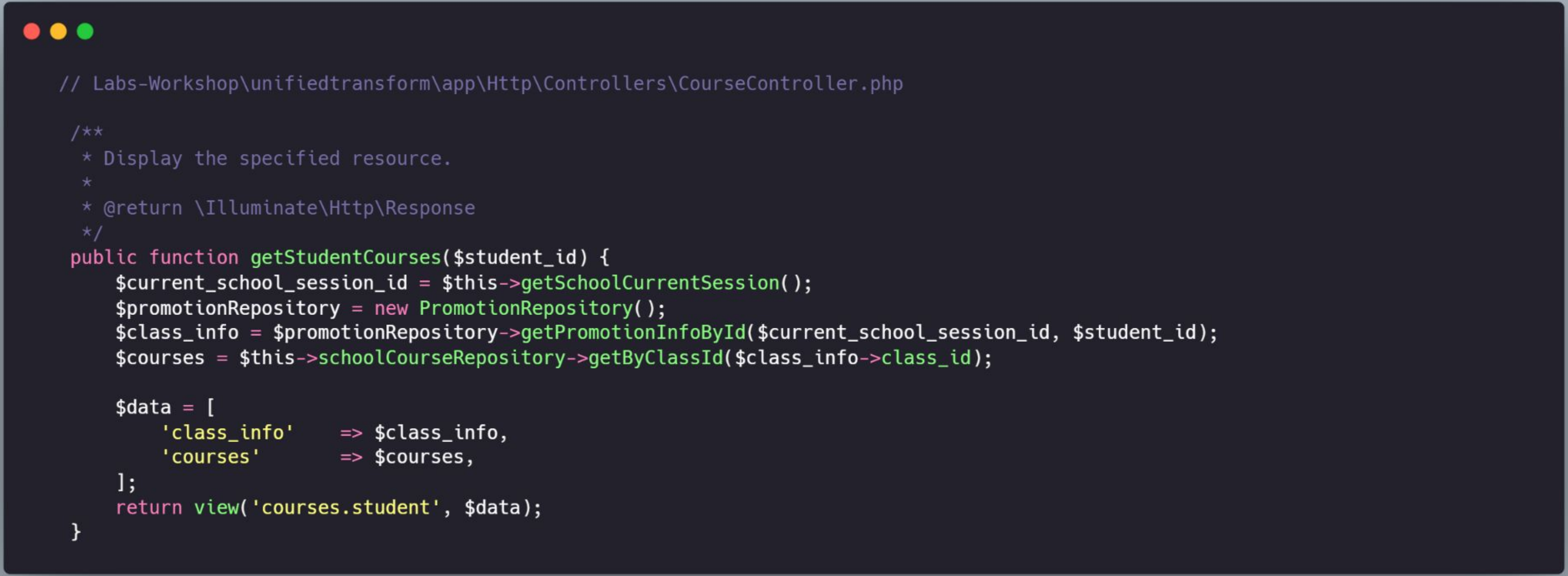
<u>Maybe I am crazy. Let’s introduce it directly.</u>

;)

Scenario #2 - unifiedtransform | CVE-2024-2292

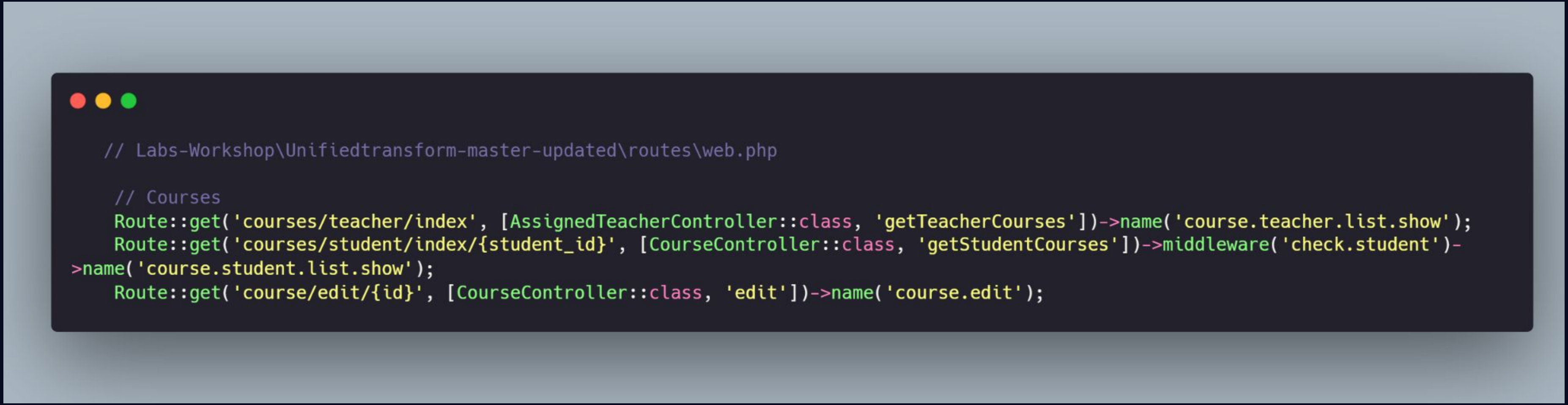

```
// Labs-Workshop\unifiedtransform\routes\web.php

// Courses
 Route::get('courses/teacher/index', [AssignedTeacherController::class, 'getTeacherCourses'])->name('course.teacher.list.show');
 Route::get('courses/student/index/{student_id}', [CourseController::class, 'getStudentCourses'])-
>name('course.student.list.show');
 Route::get('course/edit/{id}', [CourseController::class, 'edit'])->name('course.edit');
```

Scenario #2 - unifiedtransform | CVE-2024-2292



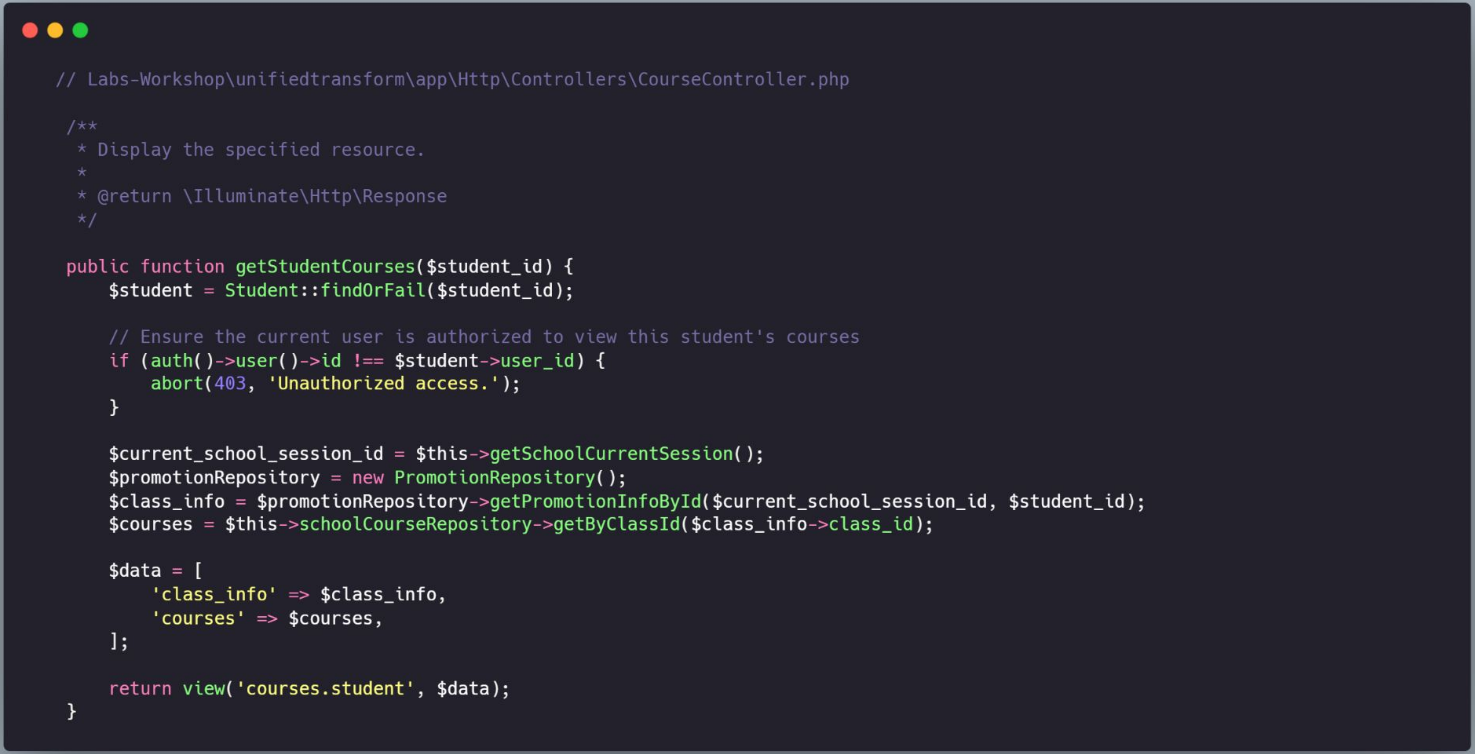
```php
// Labs-Workshop\unifiedtransform\app\Http\Controllers\CourseController.php

 /**
  * Display the specified resource.
  *
  * @return \Illuminate\Http\Response
  */
 public function getStudentCourses($student_id) {
     $current_school_session_id = $this->getSchoolCurrentSession();
     $promotionRepository = new PromotionRepository();
     $class_info = $promotionRepository->getPromotionInfoById($current_school_session_id, $student_id);
     $courses = $this->schoolCourseRepository->getByClassId($class_info->class_id);

     $data = [
         'class_info'    => $class_info,
         'courses'       => $courses,
     ];
     return view('courses.student', $data);
 }
```

Scenario #2 - unifiedtransform | CVE-2024-2292 |Patch



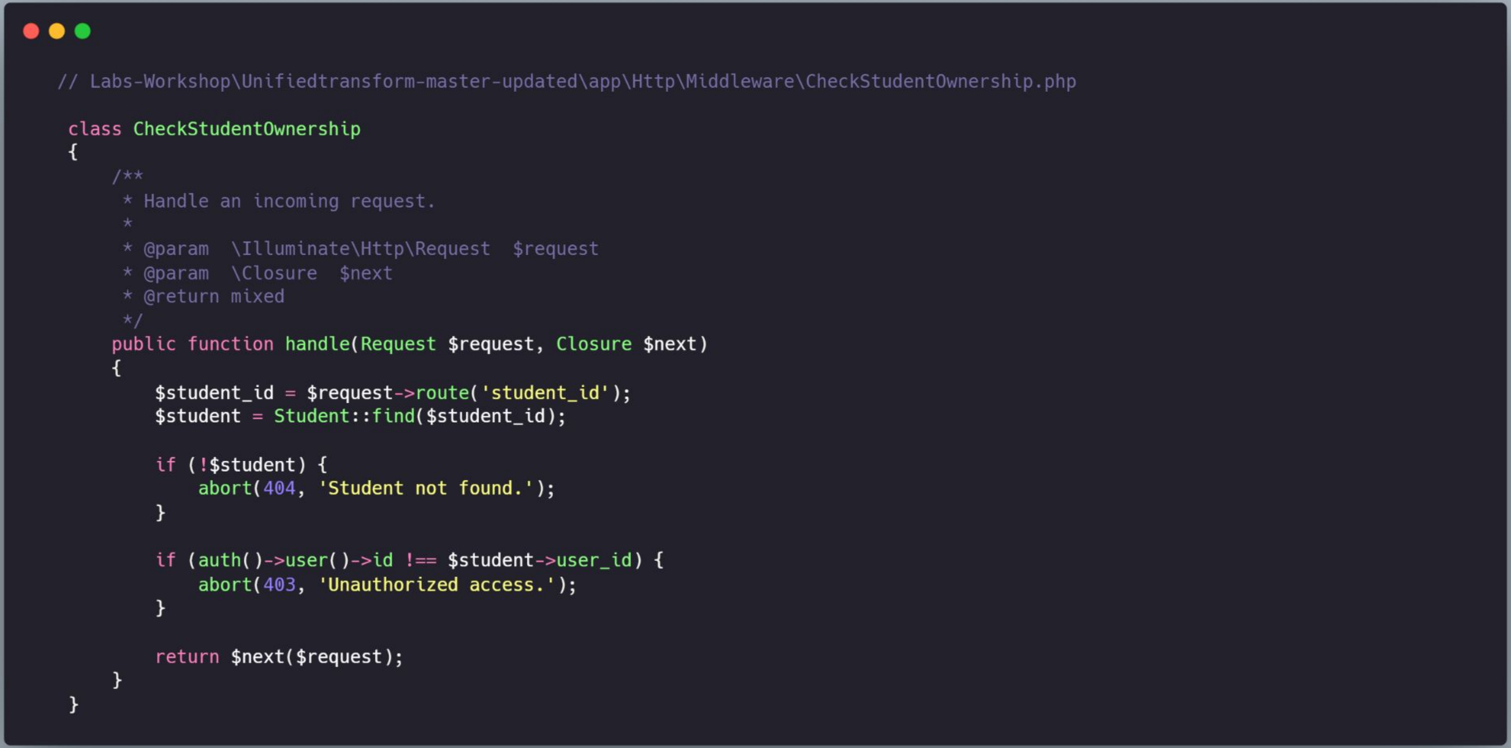
```
   // Labs-Workshop\Unifiedtransform-master-updated\routes\web.php

    // Courses
    Route::get('courses/teacher/index', [AssignedTeacherController::class, 'getTeacherCourses'])->name('course.teacher.list.show');
    Route::get('courses/student/index/{student_id}', [CourseController::class, 'getStudentCourses'])->middleware('check.student')-
>name('course.student.list.show');
    Route::get('course/edit/{id}', [CourseController::class, 'edit'])->name('course.edit');
```

Scenario #2 - unifiedtransform | CVE-2024-2292 |Patch



```
// Labs-Workshop\unifiedtransform\app\Http\Controllers\CourseController.php

 /**
  * Display the specified resource.
  *
  * @return \Illuminate\Http\Response
  */

 public function getStudentCourses($student_id) {
     $student = Student::findOrFail($student_id);

     // Ensure the current user is authorized to view this student's courses
     if (auth()->user()->id !== $student->user_id) {
         abort(403, 'Unauthorized access.');
     }

     $current_school_session_id = $this->getSchoolCurrentSession();
     $promotionRepository = new PromotionRepository();
     $class_info = $promotionRepository->getPromotionInfoById($current_school_session_id, $student_id);
     $courses = $this->schoolCourseRepository->getByClassId($class_info->class_id);

     $data = [
         'class_info' => $class_info,
         'courses' => $courses,
     ];

     return view('courses.student', $data);
 }
```

Scenario #2 - unifiedtransform | CVE-2024-2292 |Patch



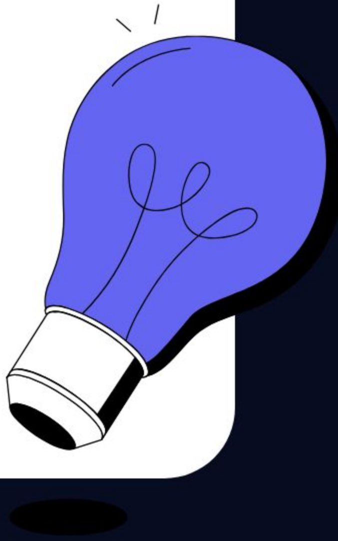
```
// Labs-Workshop\Unifiedtransform-master-updated\app\Http\Middleware\CheckStudentOwnership.php

 class CheckStudentOwnership
 {
     /**
      * Handle an incoming request.
      *
      * @param  \Illuminate\Http\Request  $request
      * @param  \Closure  $next
      * @return mixed
      */
     public function handle(Request $request, Closure $next)
     {
         $student_id = $request->route('student_id');
         $student = Student::find($student_id);

         if (!$student) {
             abort(404, 'Student not found.');
         }

         if (auth()->user()->id !== $student->user_id) {
             abort(403, 'Unauthorized access.');
         }

         return $next($request);
     }
 }
```

02 Mass Assignment

Mass Assignment

- ❖ **“**Software frameworks sometime allow developers to **automatically bind HTTP request parameters** into program **code variables or objects** to make using that framework easier on developers. This can sometimes cause harm.**” - OWASP**

- ❖ Alternative Names:
  - ➢ ***Mass Assignment:*** Ruby on Rails, NodeJS.
  - ➢ ***Autobinding:*** Spring MVC, ASP NET MVC.
  - ➢ ***Object injection:*** PHP.



Mass Assignment |Continued

Request Body

```
{
    "name": "User",
    "email": "user@google.com",
    "password": "pAssword123",
    "number": "12345678"
}
```

User object:

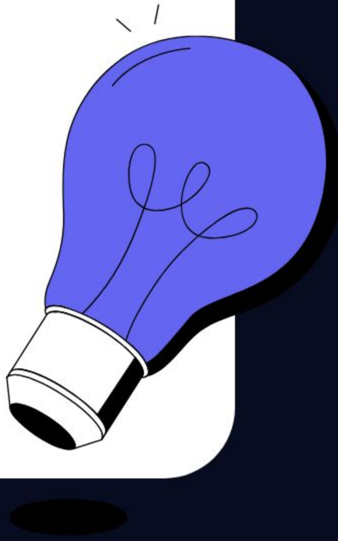
Name = x

Email = x

Password = x

Number = x

Role = x



Mass Assignment |Continued

Request Body

```
{
    "name": "User",
    "email": "user@google.com",
    "password": "pAssword123",
    "number": "12345678",
    "role": 0
}
```

User object:

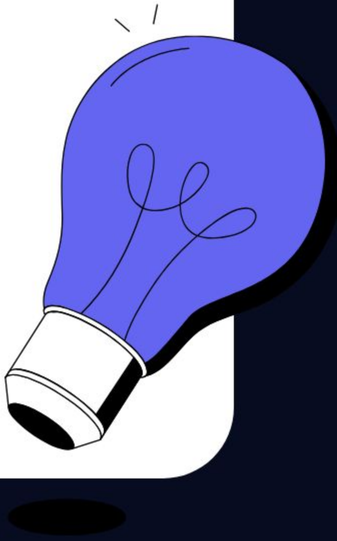
Name = x

Email = x

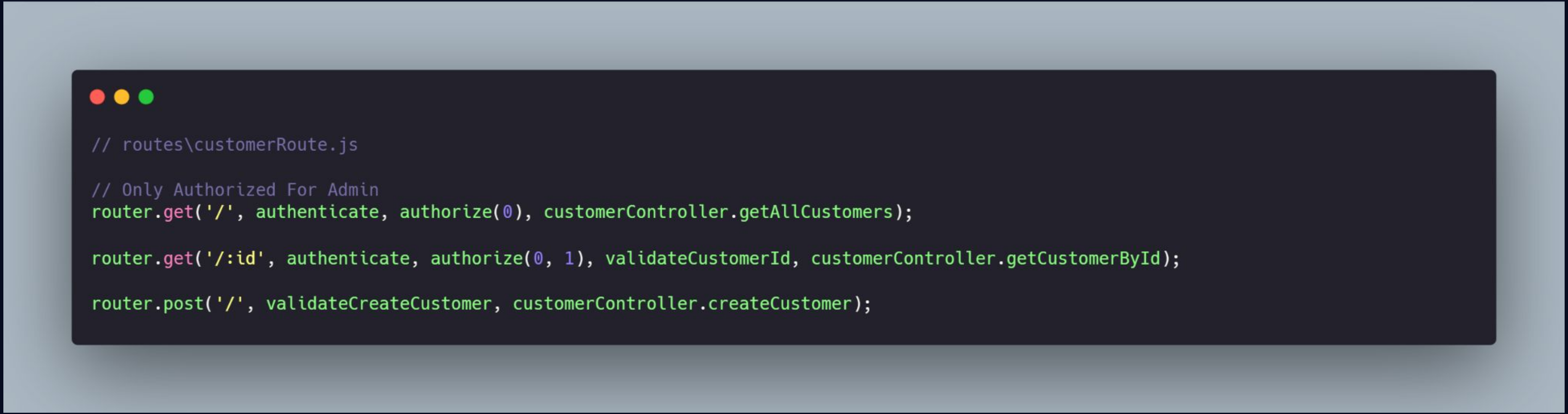
Password = x

Number = x

Role = x

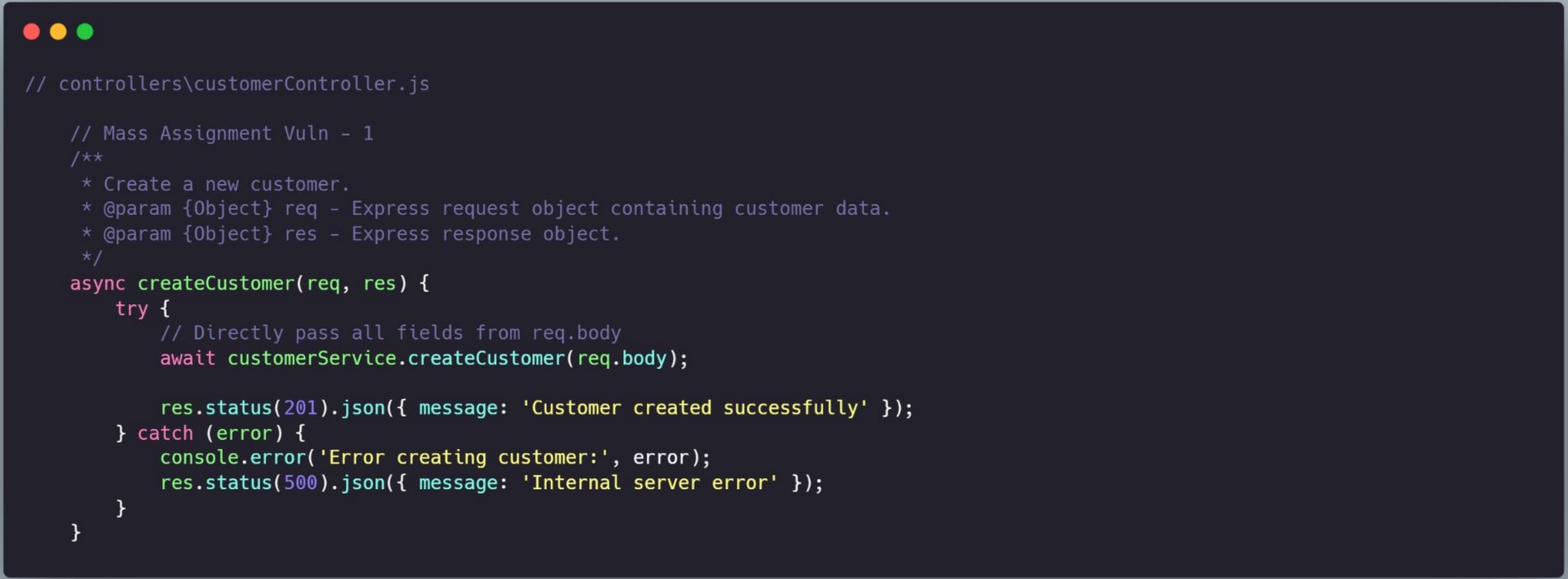


Scenario #1 - Real World Scenario



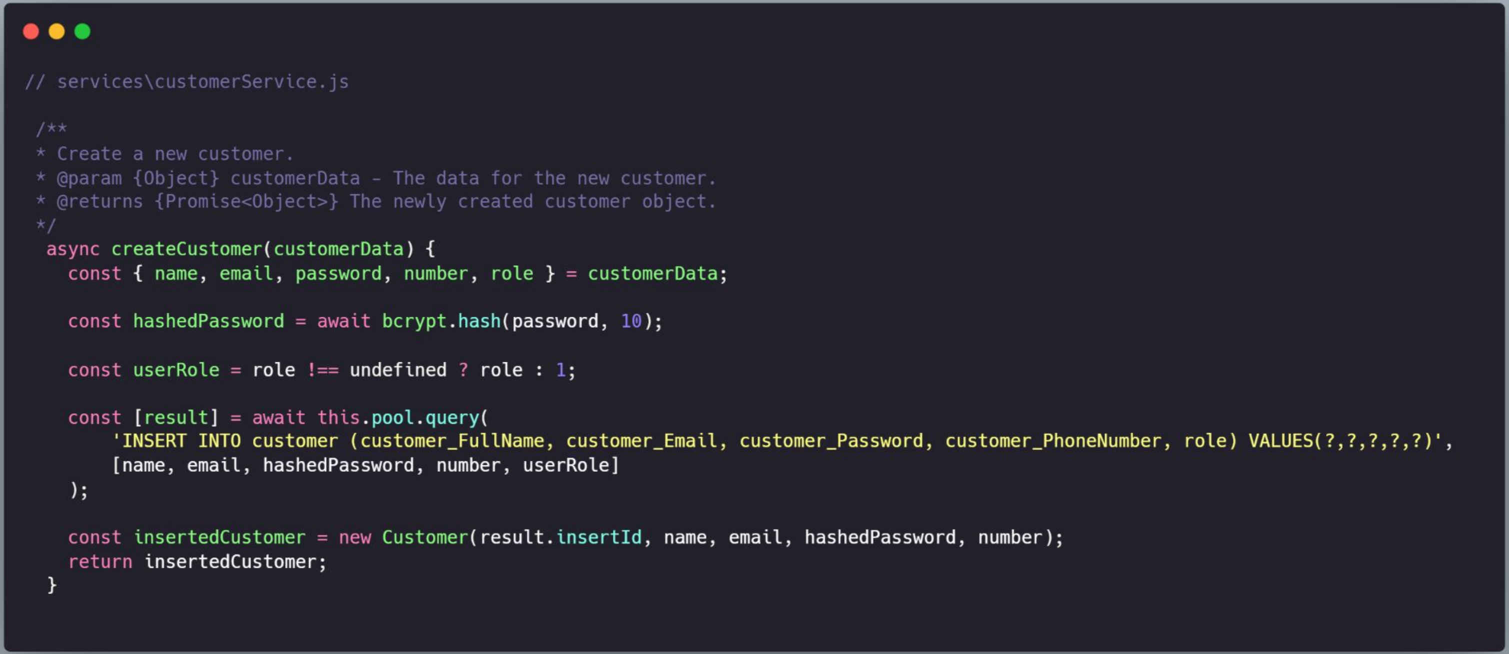
```js
// routes\customerRoute.js

// Only Authorized For Admin
router.get('/', authenticate, authorize(0), customerController.getAllCustomers);

router.get('/:id', authenticate, authorize(0, 1), validateCustomerId, customerController.getCustomerById);

router.post('/', validateCreateCustomer, customerController.createCustomer);
```

Scenario #1 - Real World Scenario



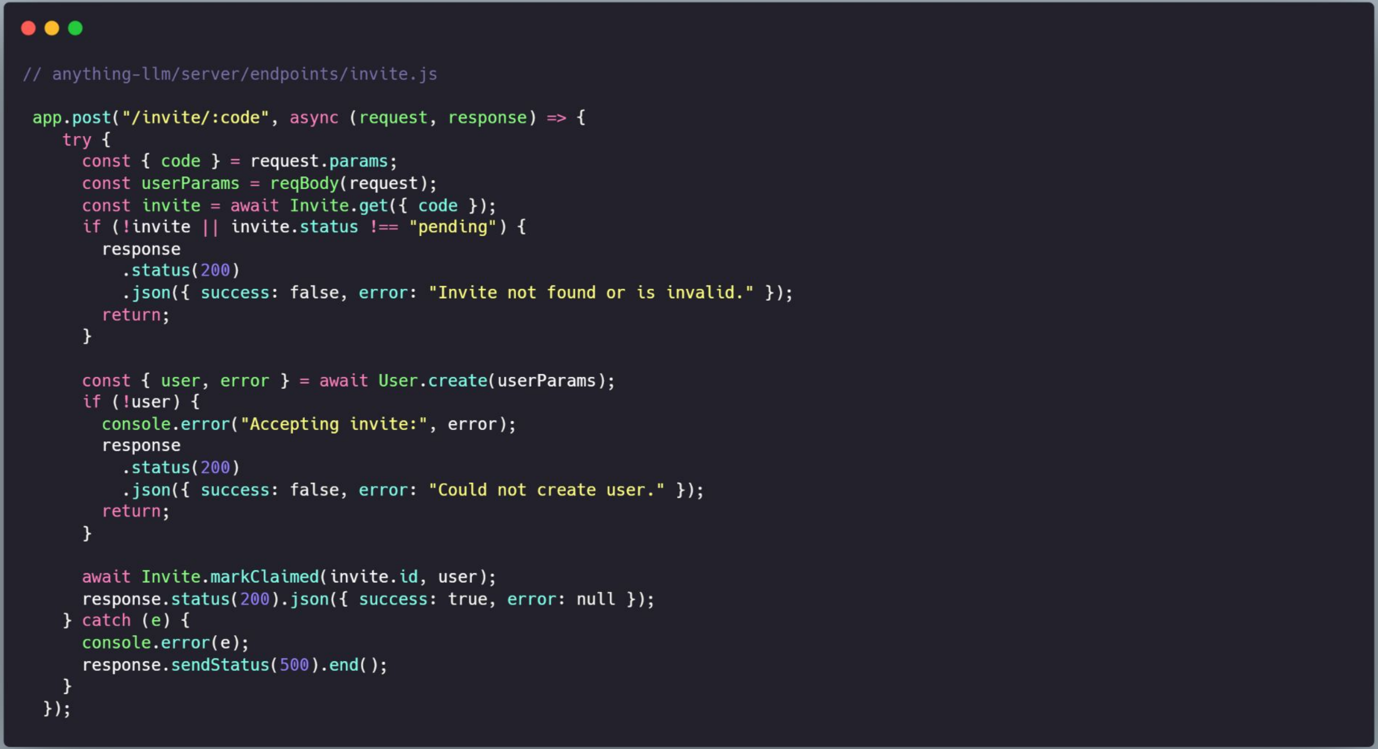
```
// controllers\customerController.js

    // Mass Assignment Vuln - 1
    /**
     * Create a new customer.
     * @param {Object} req - Express request object containing customer data.
     * @param {Object} res - Express response object.
     */
    async createCustomer(req, res) {
        try {
            // Directly pass all fields from req.body
            await customerService.createCustomer(req.body);

            res.status(201).json({ message: 'Customer created successfully' });
        } catch (error) {
            console.error('Error creating customer:', error);
            res.status(500).json({ message: 'Internal server error' });
        }
    }
```

Scenario #1 - Real World Scenario



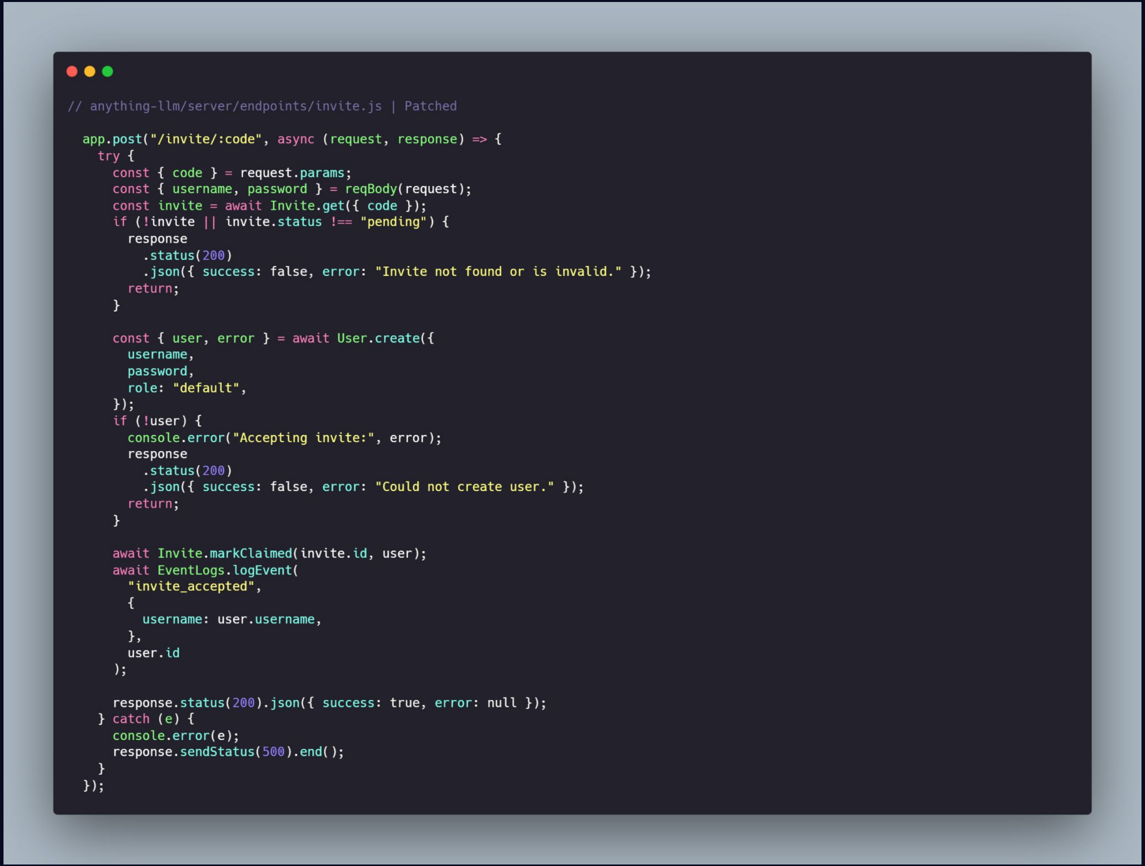
```javascript
// services\customerService.js

 /**
 * Create a new customer.
 * @param {Object} customerData - The data for the new customer.
 * @returns {Promise<Object>} The newly created customer object.
 */
  async createCustomer(customerData) {
    const { name, email, password, number, role } = customerData;

    const hashedPassword = await bcrypt.hash(password, 10);

    const userRole = role !== undefined ? role : 1;

    const [result] = await this.pool.query(
        'INSERT INTO customer (customer_FullName, customer_Email, customer_Password, customer_PhoneNumber, role) VALUES(?,?,?,?,?)',
        [name, email, hashedPassword, number, userRole]
    );

    const insertedCustomer = new Customer(result.insertId, name, email, hashedPassword, number);
    return insertedCustomer;
  }
```

Scenario #2 - anything-llm |CVE-2024-0404



```
// anything-llm/server/endpoints/invite.js

 app.post("/invite/:code", async (request, response) => {
    try {
      const { code } = request.params;
      const userParams = reqBody(request);
      const invite = await Invite.get({ code });
      if (!invite || invite.status !== "pending") {
        response
          .status(200)
          .json({ success: false, error: "Invite not found or is invalid." });
        return;
      }

      const { user, error } = await User.create(userParams);
      if (!user) {
        console.error("Accepting invite:", error);
        response
          .status(200)
          .json({ success: false, error: "Could not create user." });
        return;
      }

      await Invite.markClaimed(invite.id, user);
      response.status(200).json({ success: true, error: null });
    } catch (e) {
      console.error(e);
      response.sendStatus(500).end();
    }
  });
```

Scenario #2 - anything-llm |CVE-2024-0404

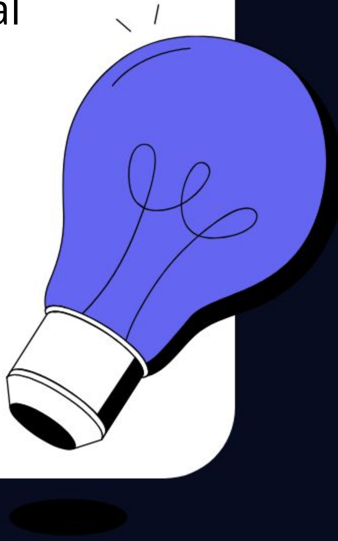


```
// anything-llm/server/endpoints/invite.js | Patched

  app.post("/invite/:code", async (request, response) => {
    try {
      const { code } = request.params;
      const { username, password } = reqBody(request);
      const invite = await Invite.get({ code });
      if (!invite || invite.status !== "pending") {
        response
          .status(200)
          .json({ success: false, error: "Invite not found or is invalid." });
        return;
      }

      const { user, error } = await User.create({
        username,
        password,
        role: "default",
      });
      if (!user) {
        console.error("Accepting invite:", error);
        response
          .status(200)
          .json({ success: false, error: "Could not create user." });
        return;
      }

      await Invite.markClaimed(invite.id, user);
      await EventLogs.logEvent(
        "invite_accepted",
        {
          username: user.username,
        },
        user.id
      );

      response.status(200).json({ success: true, error: null });
    } catch (e) {
      console.error(e);
      response.sendStatus(500).end();
    }
  });
```

03 Prototype Pollution

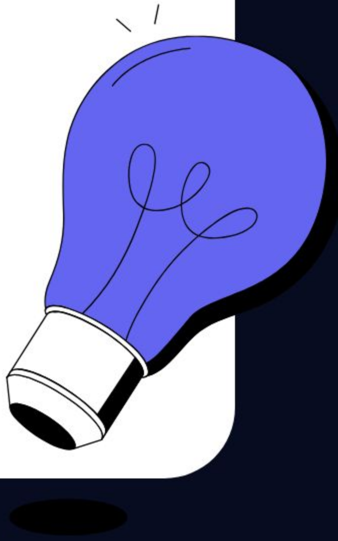
Prototype Pollution

❖ Prototype pollution is often described as a JavaScript vulnerability (though I don’t entirely agree with this characterization—I’ll explain why) that allows attackers to inject arbitrary properties into global object prototypes. These properties can then be inherited by user-defined objects.



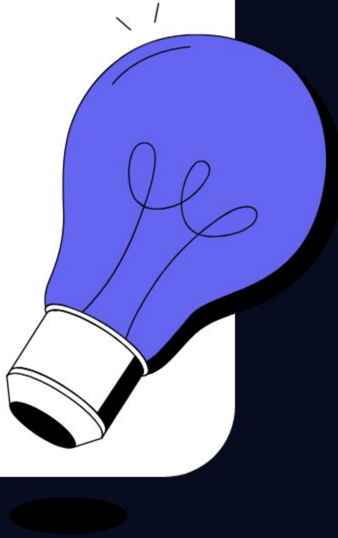
Prototype Pollution |Impact

❖ This vulnerability can result in significant security risks, including unauthorized data access, privilege escalation, and potentially even remote code execution.

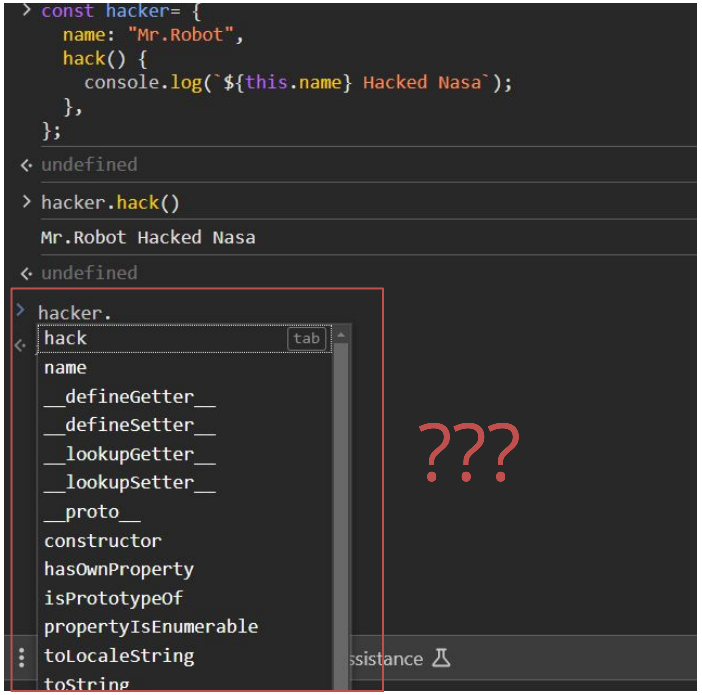


Prototype Pollution

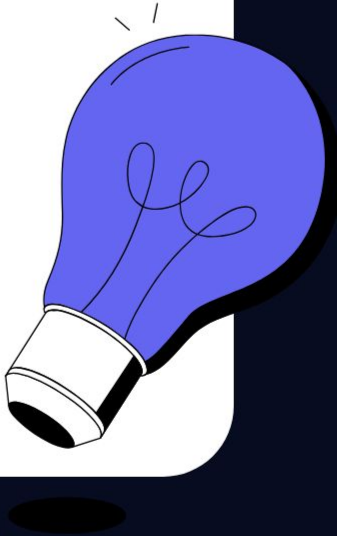
```
const hacker= {
  name: "Mr.Robot",
  hack() {
    console.log(`${this.name} Hacked Nasa`);
  },
};

hacker.hack(); // Mr.Robot Hacked Nasa
```



Prototype Pollution

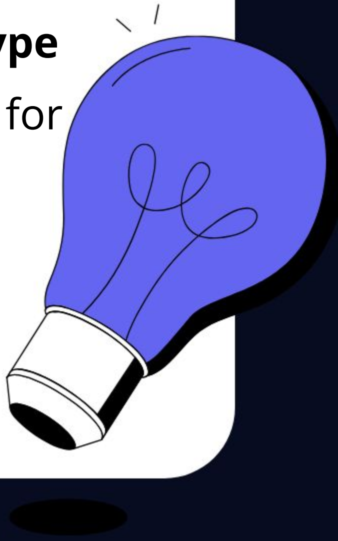


```
> const hacker= {
    name: "Mr.Robot",
    hack() {
      console.log(`${this.name} Hacked Nasa`);
    },
  };
<· undefined
> hacker.hack()
  Mr.Robot Hacked Nasa
<· undefined
> hacker.
  hack                tab
  name
  __defineGetter__
  __defineSetter__
  __lookupGetter__
  __lookupSetter__
  __proto__
  constructor
  hasOwnProperty
  isPrototypeOf
  propertyIsEnumerable
  toLocaleString
  toString
```



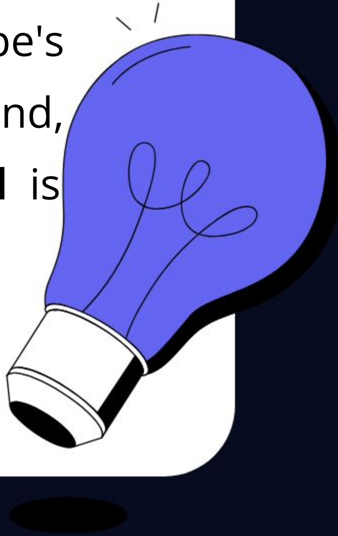
What are these extra properties, and where do they come from?

- “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.” - ***MDN Web Docs***



What are these extra properties, and where do they come from?

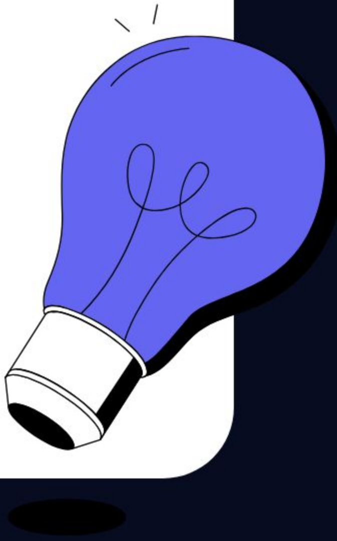
- ❖ “When you try to access a **property** of an **object**: if the property can't be found in the object itself, the prototype is searched for the property. If the property still can't be found, then the prototype's prototype is searched, and so on until either the property is found, or the end of the chain is reached, in which case **undefined** is returned” - ***MDN Web Docs***



What are these extra properties, and where do they come from?

❖ What is the prototype for ***hacker*** object?

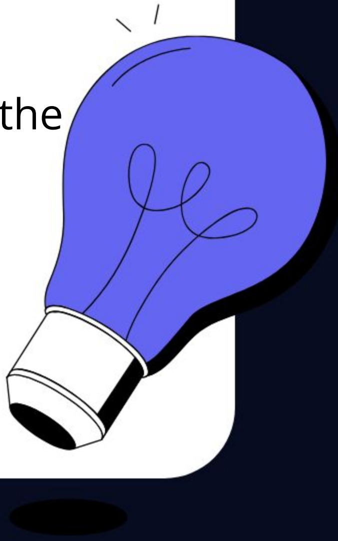
```
Object.getPrototypeOf(hacker);
```



What are these extra properties, and where do they come from?

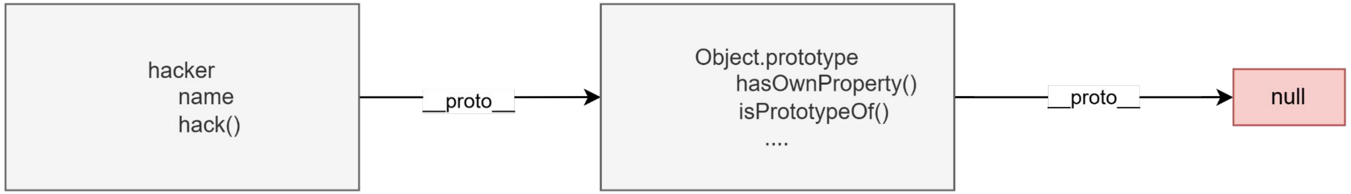
- ❖ ***Object.prototype***: is the most basic prototype, that all objects have by default.

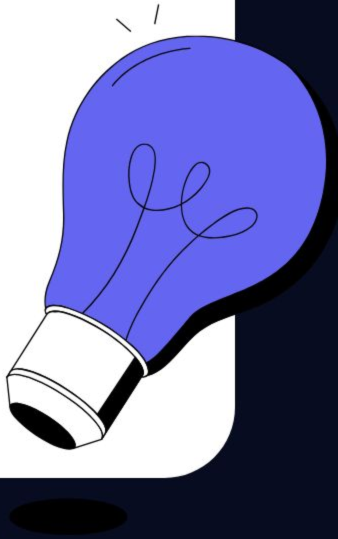
- ❖ The prototype of ***Object.prototype*** is ***null***, so it's at the **end** of the **prototype chain**.



What are these extra properties, and where do they come from?

- Prototype chain:

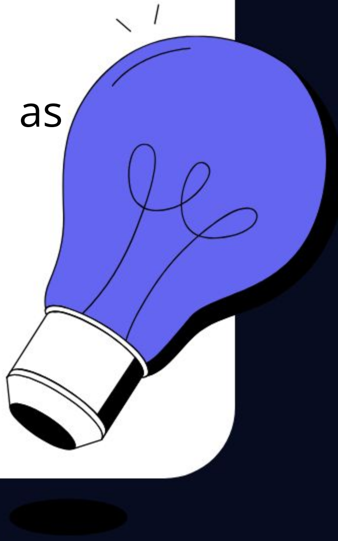




How Does Prototype Pollution Vulnerability Arise?

- ❖ Prototype pollution vulnerabilities generally occur when a JavaScript function **merges** an object with **user-controllable properties** into an existing object recursively, without properly **sanitizing the keys**.

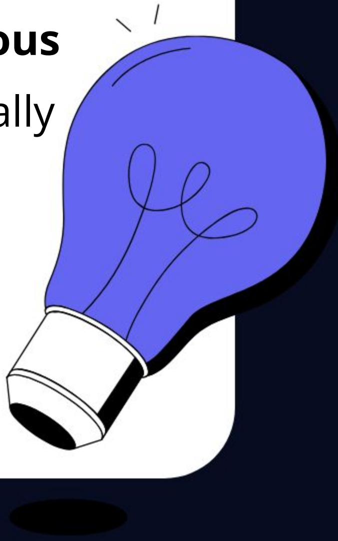
- ❖ This enables an attacker to inject a property with a key such as **__proto__**, along with arbitrary nested properties.



How Does Prototype Pollution Vulnerability Arise?

- ❖ The merge operation might assign nested properties to the **object's prototype** rather than to the **target object itself**.

- ❖ This allows an attacker to **pollute the prototype** with **malicious properties**, which the application may later use in potentially harmful ways.

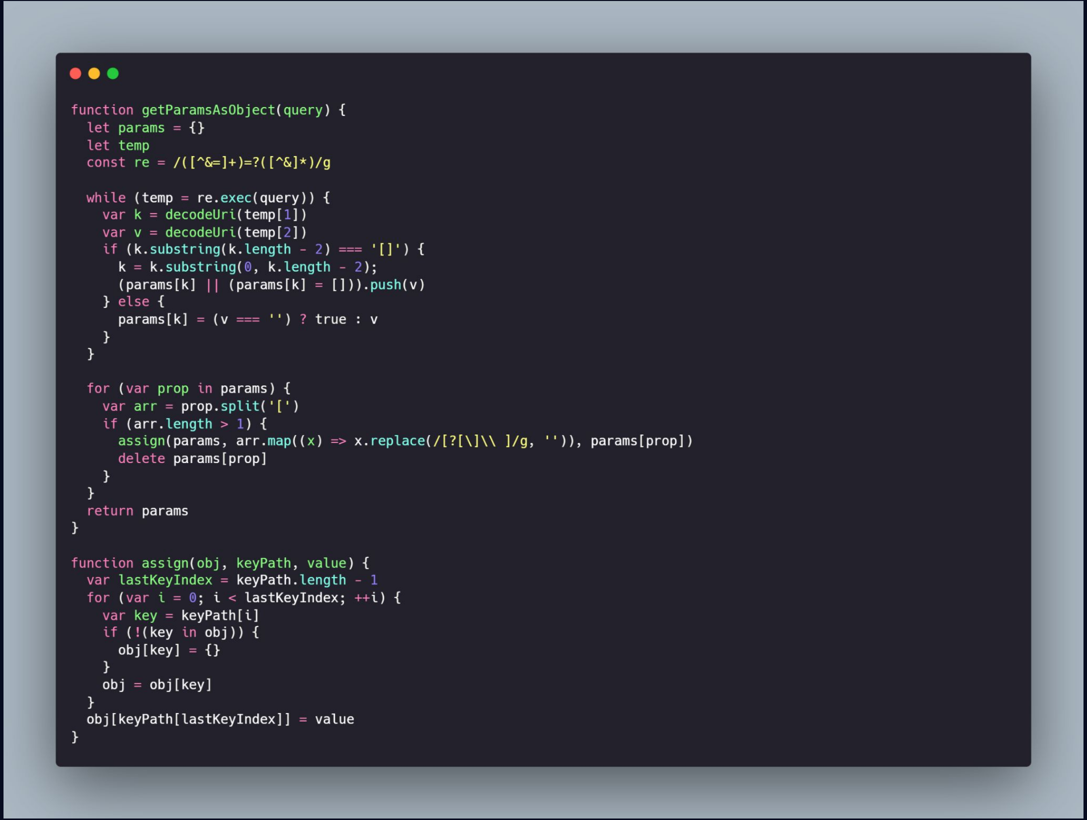


How Does Prototype Pollution Vulnerability Arise?

- ❖ What do we need to pollute the prototype?
  - ➢ **A prototype pollution source:** An input that enables you to poison prototype objects with arbitrary properties.
  - ➢ **A sink:** A javascript function or a DOM element that enable arbitrary code execution.
  - ➢ **An exploitable gadget:** This is any property that is passed into a sink without proper filtering or sanitization.

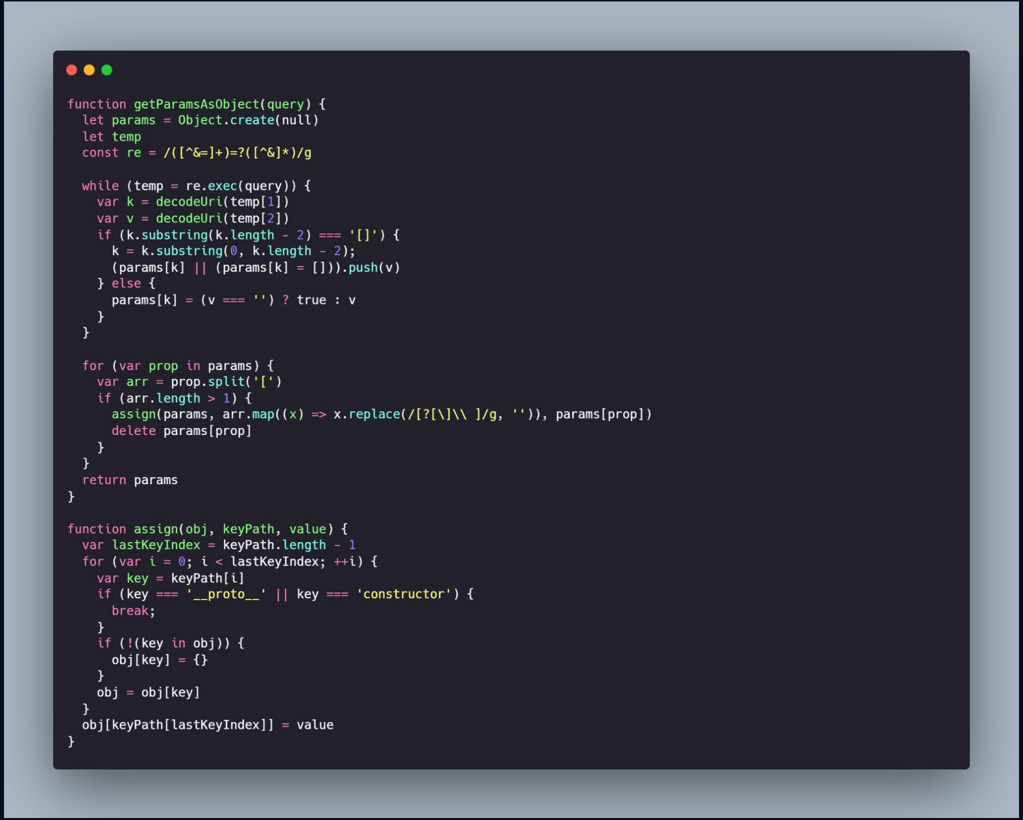
Scenario #1 - Real World Scenario

Scenario #2 - analytics-utils < 1.0.3



```javascript
function getParamsAsObject(query) {
  let params = {}
  let temp
  const re = /([^&=]+)=?([^&]*)/g

  while (temp = re.exec(query)) {
    var k = decodeUri(temp[1])
    var v = decodeUri(temp[2])
    if (k.substring(k.length - 2) === '[]') {
      k = k.substring(0, k.length - 2);
      (params[k] || (params[k] = [])).push(v)
    } else {
      params[k] = (v === '') ? true : v
    }
  }

  for (var prop in params) {
    var arr = prop.split('[')
    if (arr.length > 1) {
      assign(params, arr.map((x) => x.replace(/[?[\]\\ ]/g, '')), params[prop])
      delete params[prop]
    }
  }
  return params
}

function assign(obj, keyPath, value) {
  var lastKeyIndex = keyPath.length - 1
  for (var i = 0; i < lastKeyIndex; ++i) {
    var key = keyPath[i]
    if (!(key in obj)) {
      obj[key] = {}
    }
    obj = obj[key]
  }
  obj[keyPath[lastKeyIndex]] = value
}
```

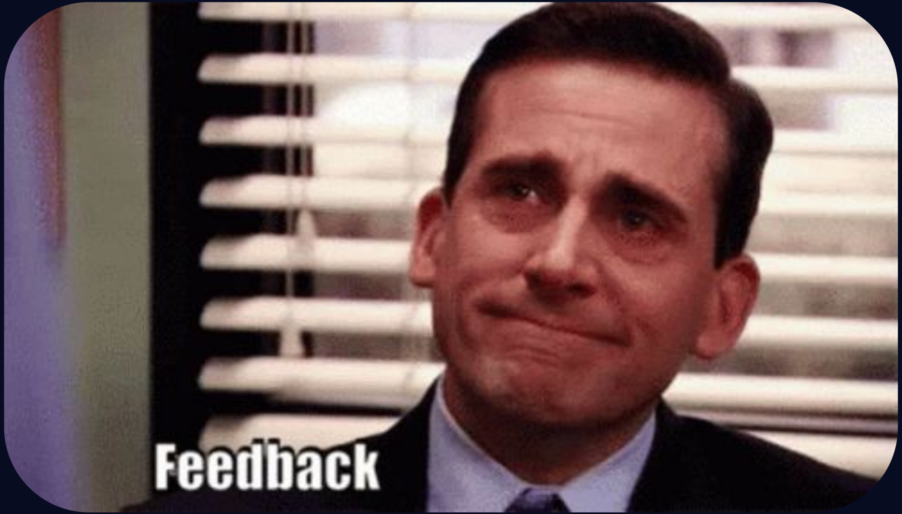
Scenario #2 - analytics-utils < 1.0.3 | Patch



```javascript
function getParamsAsObject(query) {
  let params = Object.create(null)
  let temp
  const re = /([^&=]+)=?([^&]*)/g

  while (temp = re.exec(query)) {
    var k = decodeUri(temp[1])
    var v = decodeUri(temp[2])
    if (k.substring(k.length - 2) === '[]') {
      k = k.substring(0, k.length - 2);
      (params[k] || (params[k] = [])).push(v)
    } else {
      params[k] = (v === '') ? true : v
    }
  }

  for (var prop in params) {
    var arr = prop.split('[')
    if (arr.length > 1) {
      assign(params, arr.map((x) => x.replace(/[?[\]\\ ]/g, '')), params[prop])
      delete params[prop]
    }
  }
  return params
}

function assign(obj, keyPath, value) {
  var lastKeyIndex = keyPath.length - 1
  for (var i = 0; i < lastKeyIndex; ++i) {
    var key = keyPath[i]
    if (key === '__proto__' || key === 'constructor') {
      break;
    }
    if (!(key in obj)) {
      obj[key] = {}
    }
    obj = obj[key]
  }
  obj[keyPath[lastKeyIndex]] = value
}
```

Questions?

Seeking Feedback



REACH ME!

Email: hassanalachek@hotmail.com

LinkedIn: in/hassan-al-achek

GitHub: @gokupwn

Social media: @hassanalachek

WTM Cyber Security Community