Configuration Manual

MSc Research Project
Cybersecurity

Aniket Kasturi
Student ID: X23136243

School of Computing
National College of Ireland

Supervisor: Rohit Verma

National College of Ireland

MSc Project Submission Sheet

School of Computing

Student Name:	Aniket Kasturi		
Student ID:	X23136243		
Programme:	MSc. Cybersecurity	**Year:**	2024-25
Module:	Practicum Part 2		
Lecturer:	Rohit Verma		
Submission Due Date:	12/10/2024		
Project Title:	FAST- Fortifying API Security Testing-A framework for automated API security testing.		
Word Count:	………………………………	**Page Count:**	11

Signature: [illegible]

Date: 12/10/2024

PLEASE READ THE FOLLOWING INSTRUCTIONS AND CHECKLIST

Attach a completed copy of this sheet to each project (including multiple copies)	☐
Attach a Moodle submission receipt of the online project submission, to each project (including multiple copies).	☐
You must ensure that you retain a HARD COPY of the project, both for your own reference and in case a project is lost or mislaid. It is not sufficient to keep a copy on computer.	☐

Assignments that are submitted to the Programme Coordinator Office must be placed into the assignment box located outside the office.

Office Use Only	
Signature:	
Date:	
Penalty Applied (if applicable):	

Configuration Manual

Aniket Kasturi
X23136243

1.1 Introduction

This manual is designed for new users to help them configure and test Automated API security using any API testing tool. As an example, here is this project implementation using Postman.

Workflow of the system goes as:

- Create a collection file.
- Set up an environment file.
- Verify the API collection.
- Write scripts to test API vulnerabilities.
- Run the tests using the Runner tab.

System Requirements:

- Operating System : Windows/Linux/MacOS , In my case – Mac OS version 14.2.1 (23C71)

- Postman Community Edition installed on your system. In my case the version is *'11.20.0'*.
- API collection from the target i.e. in my case https://restful-booker.herokuapp.com/apidoc/index.html and self-created environment file.

- Minimum Specifications: 8gb RAM, 50 Gb Disk Space.

Here is the full process describing this project process

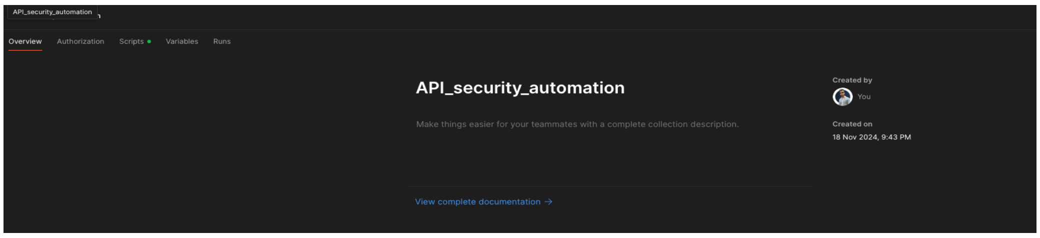
1.2 Importing the Collection

1.2.1 Step 1: Open Postman

Launch Postman Application on your system once downloaded.

1.2.2 Step 2: Create a Collection

1. Click on the **New** button in the top-left corner and then select 'Collection' and name it , here in my case I have named it as "API_security_automation".

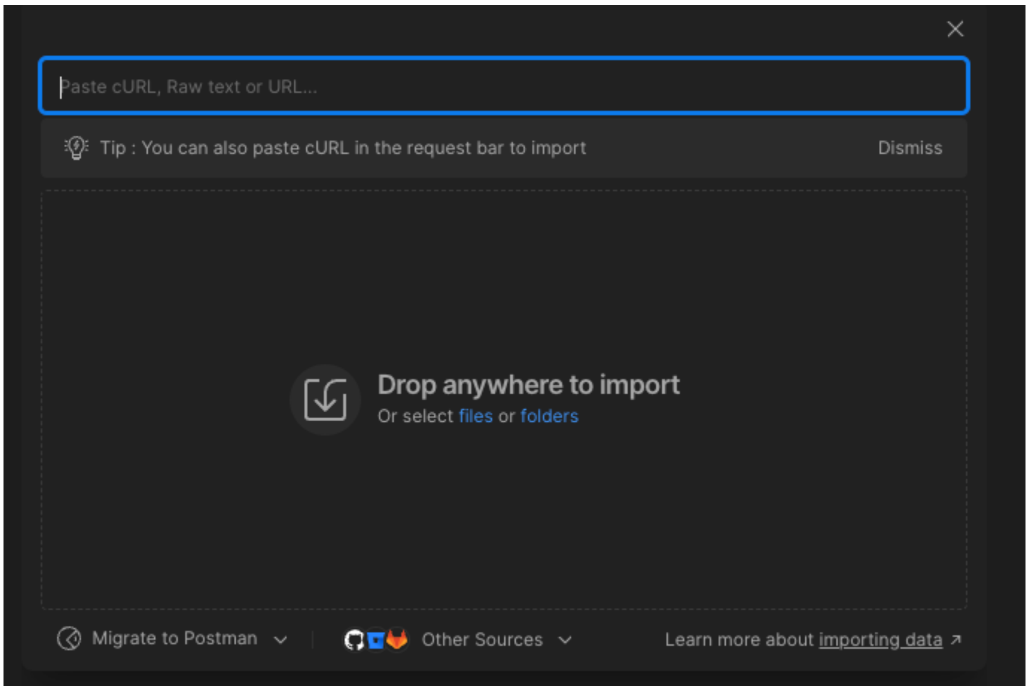

API_security_automation

Make things easier for your teammates with a complete collection description.

Created by You

Created on 18 Nov 2024, 9:43 PM

2. Now from the website copy all the curl requests for each API calls and paste them by clicking on "**import**" button in the file option and upload your collection file.


Paste cURL, Raw text or URL...
Tip : You can also paste cURL in the request bar to import
Dismiss
Drop anywhere to import
Or select files or folders
Migrate to Postman
Other Sources
Learn more about importing data

3. Now Each request will open and make sure to save and move them under the collection which you made.

1.2.3 Step 3: Verify Import

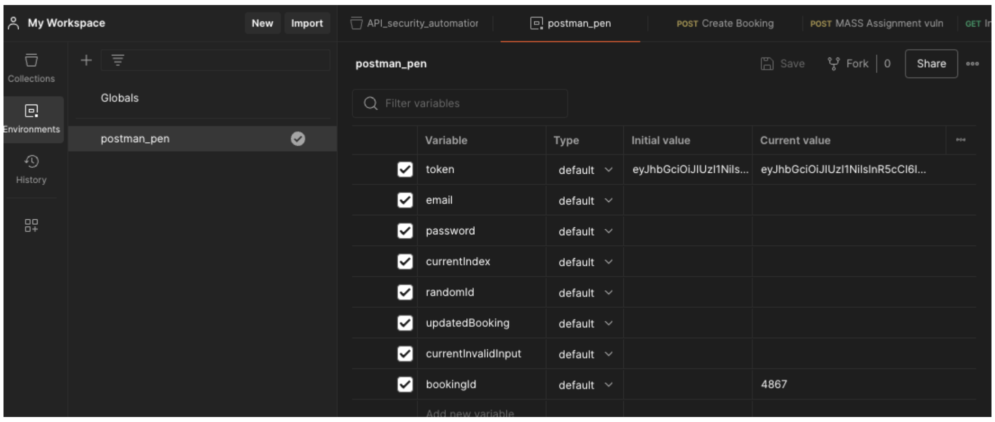
Check if the collection appears under **Collections** in the sidebar along with the API calls imported from Curl.

1.3 Setting Up the Environment

1.3.1 Step 1: Import Environment File

1. Go to the **Environments** tab in Postman.

2. Click **New** and create your environment file having these details.



postman_pen

	Variable	Type	Initial value	Current value
✓	token	default	eyJhbGciOiJIUzI1NiIs...	eyJhbGciOiJIUzI1NiIsInR5cCI6I...
✓	email	default		
✓	password	default		
✓	currentIndex	default		
✓	randomId	default		
✓	updatedBooking	default		
✓	currentInvalidInput	default		
✓	bookingId	default		4867

1.3.2 Step 2: Select Environment

From the dropdown in the top-right corner, select the imported environment as it is necessary when we run the collection under the runner tab.

1.3.3 Step 3: Verify Environment

Ensure the environment variables are loaded correctly by checking the **Variables** tab.

1.4 Testing API Vulnerabilities

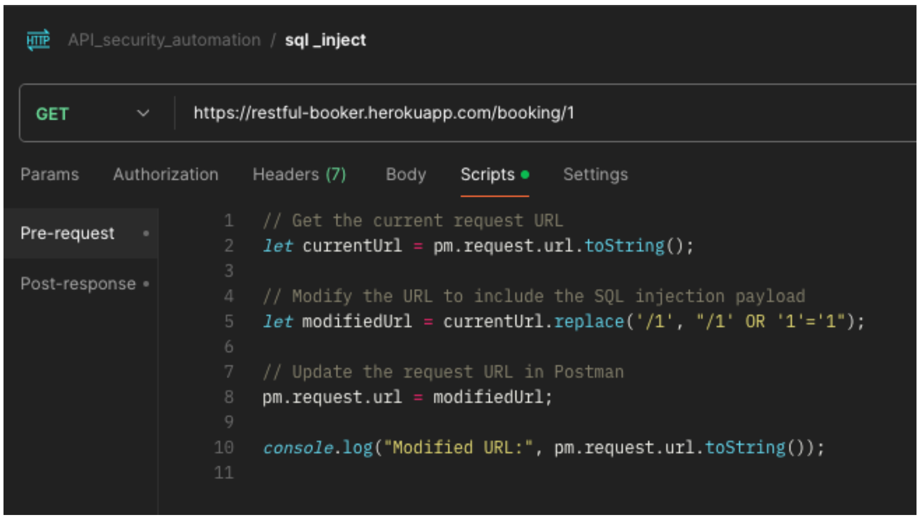
1.4.1 Step 1: Open a Request

Click on any request in your imported collection.

1.4.2 Step 2: Custom script development

In Postman, we add scripts for vulnerability testing under the “Pre-request” and “post-response” of individual and collection level Script tab and to check whether vulnerabilities.

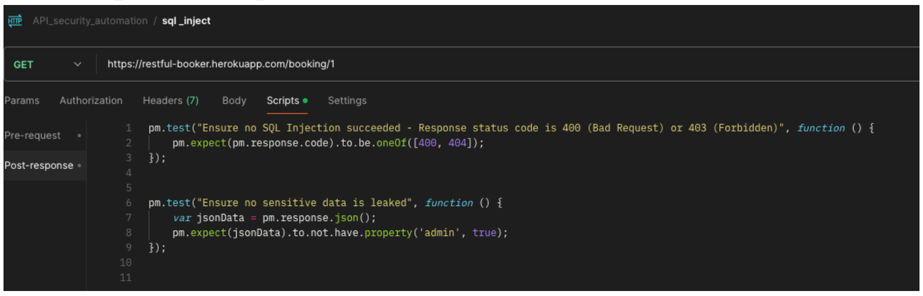
1. Navigate to the **Scripts** tab for each request.
2. Here scripts to check for potential vulnerabilities. Examples include:
   - **SQL Injection**: Send payloads like `'1 OR '1'='1` in query parameters and validate responses for errors or unintended data.
     Pre-request Script tab:


API_security_automation / sql _inject

GET https://restful-booker.herokuapp.com/booking/1

```
// Get the current request URL
let currentUrl = pm.request.url.toString();

// Modify the URL to include the SQL injection payload
let modifiedUrl = currentUrl.replace('/1', "/1' OR '1'='1");

// Update the request URL in Postman
pm.request.url = modifiedUrl;

console.log("Modified URL:", pm.request.url.toString());
```

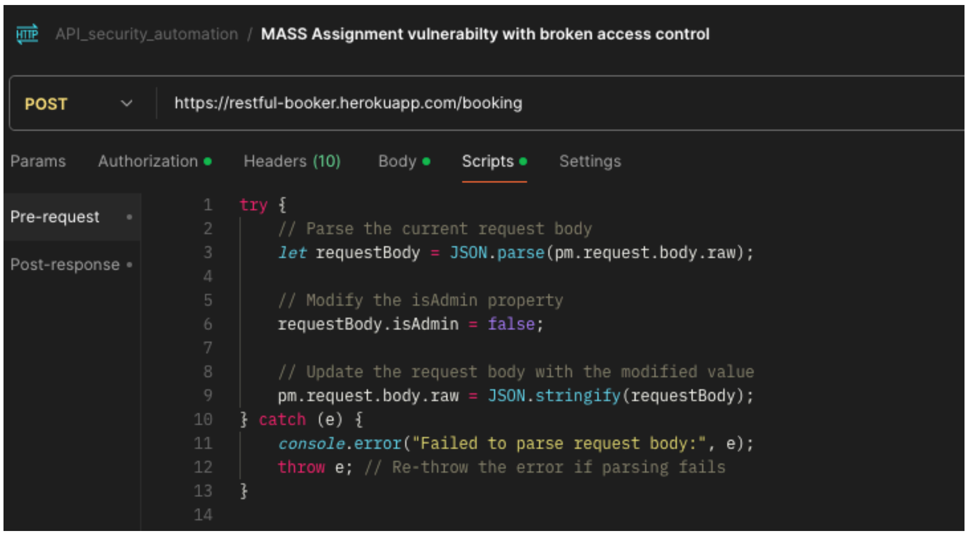
Post-response Script tab:


API_security_automation / sql _inject

GET https://restful-booker.herokuapp.com/booking/1

```
pm.test("Ensure no SQL Injection succeeded - Response status code is 400 (Bad Request) or 403 (Forbidden)", function () {
    pm.expect(pm.response.code).to.be.oneOf([400, 404]);
});


pm.test("Ensure no sensitive data is leaked", function () {
    var jsonData = pm.response.json();
    pm.expect(jsonData).to.not.have.property('admin', true);
});
```

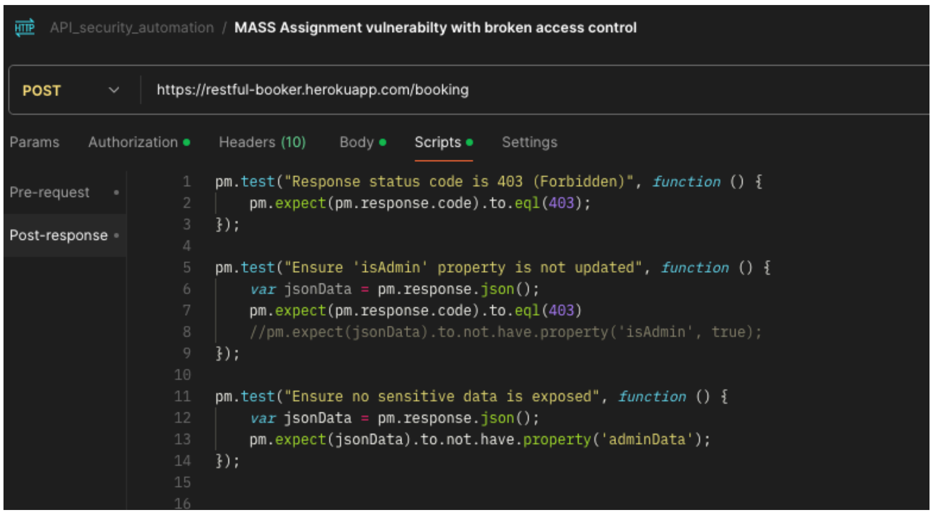
- **Mass Assignment with broken access control**: Develop custom scripts here in my case in mass assignment I am changing an unauthorized field "isAdmin" param which is set to be true, so in the pre-request script I have written code to set it to 'false'.


API_security_automation / MASS Assignment vulnerabilty with broken access control

POST https://restful-booker.herokuapp.com/booking

```
try {
    // Parse the current request body
    let requestBody = JSON.parse(pm.request.body.raw);

    // Modify the isAdmin property
    requestBody.isAdmin = false;

    // Update the request body with the modified value
    pm.request.body.raw = JSON.stringify(requestBody);
} catch (e) {
    console.error("Failed to parse request body:", e);
    throw e; // Re-throw the error if parsing fails
}
```

Post-response tab script: To check for responses which will flag for vulnerability detected or not.

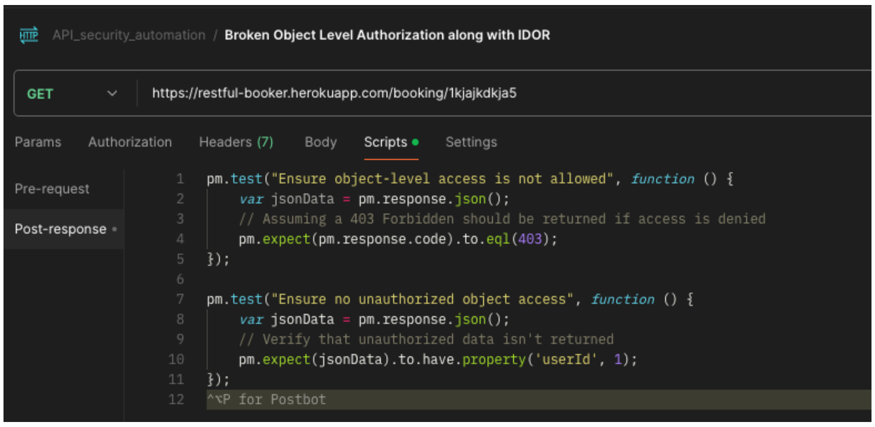

API_security_automation / MASS Assignment vulnerabilty with broken access control
POST
Params
Authorization
Headers (10)
Body
Scripts
Settings
Pre-request
Post-response

https://restful-booker.herokuapp.com/booking

```
pm.test("Response status code is 403 (Forbidden)", function () {
    pm.expect(pm.response.code).to.eql(403);
});

pm.test("Ensure 'isAdmin' property is not updated", function () {
    var jsonData = pm.response.json();
    pm.expect(pm.response.code).to.eql(403)
    //pm.expect(jsonData).to.not.have.property('isAdmin', true);
});

pm.test("Ensure no sensitive data is exposed", function () {
    var jsonData = pm.response.json();
    pm.expect(jsonData).to.not.have.property('adminData');
});
```

- **Broken Object Level Authorization (BOLA)**:
  Pre-request Script: Here in my case we have manually changed the /{objected} present at the last url endpoint. To check whether a user is able to see details of another user or not, and as this API call should have authorization based upon so we found BOLA vulnerability as well.

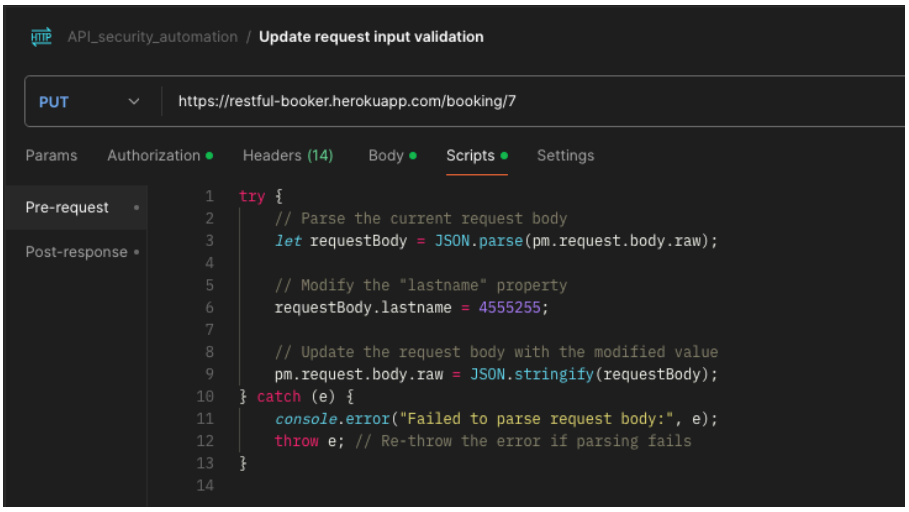
  Post-response Script:



https://restful-booker.herokuapp.com/booking/1kjajkdkja5

```
pm.test("Ensure object-level access is not allowed", function () {
    var jsonData = pm.response.json();
    // Assuming a 403 Forbidden should be returned if access is denied
    pm.expect(pm.response.code).to.eql(403);
});

pm.test("Ensure no unauthorized object access", function () {
    var jsonData = pm.response.json();
    // Verify that unauthorized data isn't returned
    pm.expect(jsonData).to.have.property('userId', 1);
});
^⌥P for Postbot
```

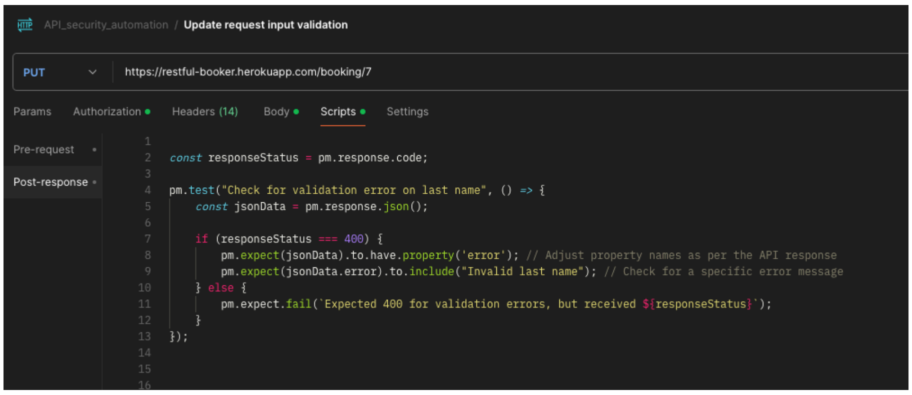
- **Input Validation Vulnerability Detection**:
  Pre-request Script: Here I have tried to change the data type in the body which had "*lastname*" as string, so for testing I changed it '*4555255'* and integer value to check for input validation vulnerability.



API_security_automation / Update request input validation
PUT
https://restful-booker.herokuapp.com/booking/7
Params
Authorization
Headers (14)
Body
Scripts
Settings
Pre-request
Post-response

```
try {
    // Parse the current request body
    let requestBody = JSON.parse(pm.request.body.raw);

    // Modify the "lastname" property
    requestBody.lastname = 4555255;

    // Update the request body with the modified value
    pm.request.body.raw = JSON.stringify(requestBody);
} catch (e) {
    console.error("Failed to parse request body:", e);
    throw e; // Re-throw the error if parsing fails
}
```

  Post-Response Script:

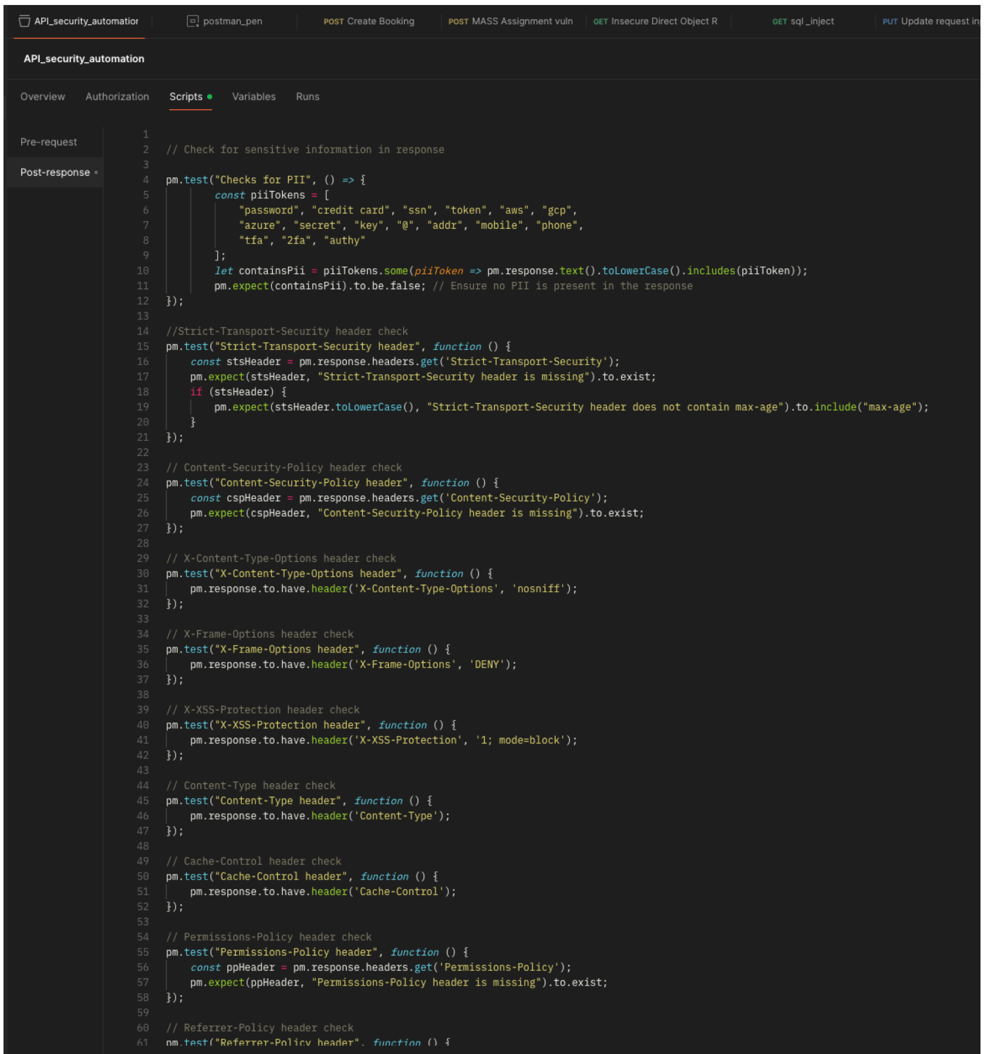


API_security_automation / Update request input validation
PUT
https://restful-booker.herokuapp.com/booking/7
Params
Authorization
Headers (14)
Body
Scripts
Settings
Pre-request
Post-response

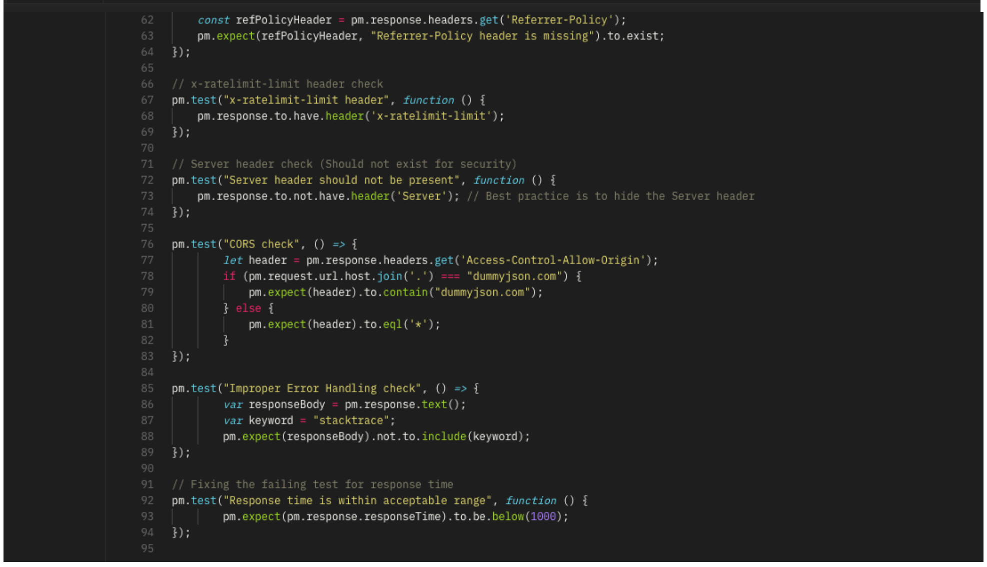
```

const responseStatus = pm.response.code;

pm.test("Check for validation error on last name", () => {
    const jsonData = pm.response.json();

    if (responseStatus === 400) {
        pm.expect(jsonData).to.have.property('error'); // Adjust property names as per the API response
        pm.expect(jsonData.error).to.include("Invalid last name"); // Check for a specific error message
    } else {
        pm.expect.fail(`Expected 400 for validation errors, but received ${responseStatus}`);
    }
});
```

- **Security Headers Check**: To test whether each and every API call has security headers or not, I have written a script that checks for each and every security header mentioned in the figure below at the collection level's post response tab.

Post – Response script:



```javascript

// Check for sensitive information in response

pm.test("Checks for PII", () => {
        const piiTokens = [
            "password", "credit card", "ssn", "token", "aws", "gcp",
            "azure", "secret", "key", "@", "addr", "mobile", "phone",
            "tfa", "2fa", "authy"
        ];
        let containsPii = piiTokens.some(piiToken => pm.response.text().toLowerCase().includes(piiToken));
        pm.expect(containsPii).to.be.false; // Ensure no PII is present in the response
});

//Strict-Transport-Security header check
pm.test("Strict-Transport-Security header", function () {
    const stsHeader = pm.response.headers.get('Strict-Transport-Security');
    pm.expect(stsHeader, "Strict-Transport-Security header is missing").to.exist;
    if (stsHeader) {
        pm.expect(stsHeader.toLowerCase(), "Strict-Transport-Security header does not contain max-age").to.include("max-age");
    }
});

// Content-Security-Policy header check
pm.test("Content-Security-Policy header", function () {
    const cspHeader = pm.response.headers.get('Content-Security-Policy');
    pm.expect(cspHeader, "Content-Security-Policy header is missing").to.exist;
});

// X-Content-Type-Options header check
pm.test("X-Content-Type-Options header", function () {
    pm.response.to.have.header('X-Content-Type-Options', 'nosniff');
});

// X-Frame-Options header check
pm.test("X-Frame-Options header", function () {
    pm.response.to.have.header('X-Frame-Options', 'DENY');
});

// X-XSS-Protection header check
pm.test("X-XSS-Protection header", function () {
    pm.response.to.have.header('X-XSS-Protection', '1; mode=block');
});

// Content-Type header check
pm.test("Content-Type header", function () {
    pm.response.to.have.header('Content-Type');
});

// Cache-Control header check
pm.test("Cache-Control header", function () {
    pm.response.to.have.header('Cache-Control');
});

// Permissions-Policy header check
pm.test("Permissions-Policy header", function () {
    const ppHeader = pm.response.headers.get('Permissions-Policy');
    pm.expect(ppHeader, "Permissions-Policy header is missing").to.exist;
});

// Referrer-Policy header check
pm.test("Referrer-Policy header", function () {
```

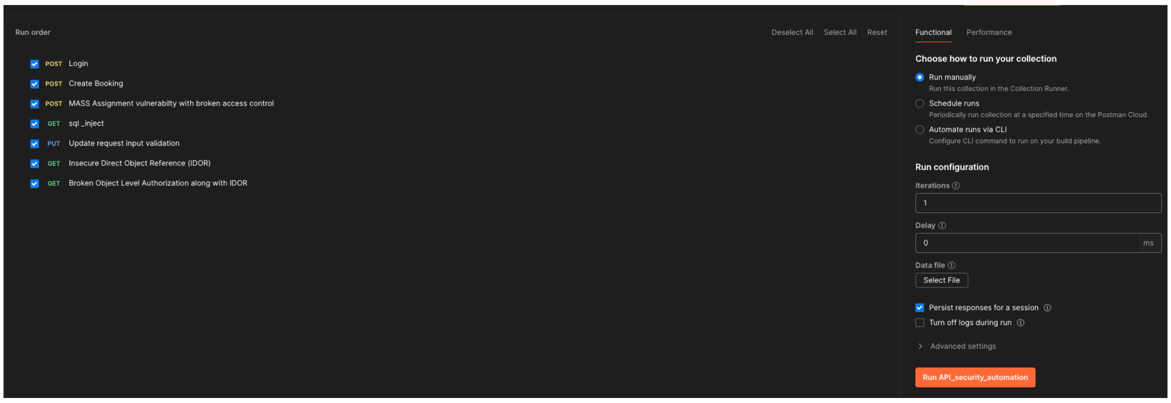


```javascript
    const refPolicyHeader = pm.response.headers.get('Referrer-Policy');
    pm.expect(refPolicyHeader, "Referrer-Policy header is missing").to.exist;
});

// x-ratelimit-limit header check
pm.test("x-ratelimit-limit header", function () {
    pm.response.to.have.header('x-ratelimit-limit');
});

// Server header check (Should not exist for security)
pm.test("Server header should not be present", function () {
    pm.response.to.not.have.header('Server'); // Best practice is to hide the Server header
});

pm.test("CORS check", () => {
        let header = pm.response.headers.get('Access-Control-Allow-Origin');
        if (pm.request.url.host.join('.') === "dummyjson.com") {
            pm.expect(header).to.contain("dummyjson.com");
        } else {
            pm.expect(header).to.eql('*');
        }
});

pm.test("Improper Error Handling check", () => {
        var responseBody = pm.response.text();
        var keyword = "stacktrace";
        pm.expect(responseBody).not.to.include(keyword);
});

// Fixing the failing test for response time
pm.test("Response time is within acceptable range", function () {
        pm.expect(pm.response.responseTime).to.be.below(1000);
});

```

1.5 Running Tests

1.5.1 Step 1: Access the Runner tab

Click the "*triple dot option*" beside the chosen collection and click the "*run collection*" button and review the response for anomalies or vulnerabilities.



Run order

- [✓] POST Login
- [✓] POST Create Booking
- [✓] POST MASS Assignment vulnerabilty with broken access control
- [✓] GET sql _inject
- [✓] PUT Update request input validation
- [✓] GET Insecure Direct Object Reference (IDOR)
- [✓] GET Broken Object Level Authorization along with IDOR

Choose how to run your collection

- [● Run manually] Run this collection in the Collection Runner.
- Schedule runs: Periodically run collection at a specified time on the Postman Cloud.
- Automate runs via CLI: Configure CLI command to run on your build pipeline.

Run configuration

Iterations: 1

Delay: 0 ms

Data file: Select File

- [✓] Persist responses for a session
- [ ] Turn off logs during run

1.5.2 Step 2: Configure Runner

1. Select your collection from the list, choose all the APIs which you want to run tests on.
2. Choose your environment file is correct.
3. Set the number of iterations and delay, if needed.
4. Check the box of 'Persists responses for a session' for checking the API calls in the console tab for future debugging or logging.

1.5.3 Step 3: Run Tests

Click the **Run** button to execute all requests in the collection.

1.5.4 Reviewing Results Under Postman Runner Tab.

Inspect the results to identify any issues or vulnerabilities. All the failed test cases will flag the vulnerabilities detected
Pay attention to:

- Status codes indicating errors (e.g., 500 Internal Server Error).
- Unexpected data in responses- improper error handling cases.
- Unauthorized access to restricted endpoints and unauthorized data type modification.

- Also Check for security headers.