

“FAST: Fortifying API Security Testing A framework for automated API security testing.”

MSc Research Project

Programme Name: 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

Supervisor: Rohit Verma

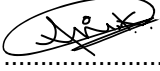
Submission Due Date: 12/10/2024

Project Title: FAST- Fortifying API Security Testing-A framework for automated API security testing.

Word Count: 7961 **Page Count:** 25

I hereby certify that the information contained in this (my submission) is information pertaining to research I conducted for this project. All information other than my own contribution will be fully referenced and listed in the relevant bibliography section at the rear of the project.

ALL internet material must be referenced in the bibliography section. Students are required to use the Referencing Standard specified in the report template. To use other author's written or electronic work is illegal (plagiarism) and may result in disciplinary action.

Signature: 

Date: 12/10/2024

PLEASE READ THE FOLLOWING INSTRUCTIONS AND CHECKLIST

Attach a completed copy of this sheet to each project (including multiple copies)	<input type="checkbox"/>
Attach a Moodle submission receipt of the online project submission, to each project (including multiple copies).	<input type="checkbox"/>
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.	<input type="checkbox"/>

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):	

FAST: Fortifying API Security Testing - A framework for automated API security testing

Aniket Kasturi

Student ID: 23136243

Abstract

The rise in demand for incorporating API (Application Programming Interface) in various software and systems has increased the use of enhanced security. API-integrated systems and software provide a wide range of facilities to the users but have a wide range of security gaps which are the potential paths for the attackers to gain malicious access. Manual form of testing offers a wide range of benefits and has the potential to detect weaknesses but has sets of drawbacks. Manual forms of testing practices sometimes are not efficient enough in modern applications and software which is continuously upgraded based on the needs and demands of the customers and also is time consuming on repetitive test cases. API Security testing tool is majorly used by different developers to develop and initiate testing of different APIs. The tool offers an advanced user-friendly platform which allows the developers to configure APIs and initiate testing phases for the APIs. On the other hand, different testing tools also have features that are used by the developers to achieve better security. Different security testing tool is generally used by the testers to develop different security test cases that can be further initiated. This helps the penetration testers to detect different sets of security flaws and vulnerabilities which are present within the system. This dissertation will mainly involve describing the wide range of roles and features of automation API security testing in the conduction of different ranges of security testing activities. It will majorly focus on the process of development of different test cases and delivering effective security activities. Moreover, the limitations of these are also discussed and further recommendations are also provided which can be used to enhance the security mechanism of the APIs.. Automated forms of security testing offer various sets of facilities which help in lowering the vulnerabilities and problems of manual testing. Automated forms of testing are associated with more scalable and enhanced security mechanisms. These sets of automated API security testing are incorporated with various sets of test cases which allows the security professionals to deal with the process of penetration testing. It also allows the manual testers to focus on more sensitive areas rather than dealing with sets of repetitive tasks.

1 Introduction

1.1 Background to the problem

The ongoing development of software has resulted in the formation and development of application programming interfaces. APIs are the main parts of developing different software applications (Manikas, K. and Manikas, K, 2016). This helps different systems to establish interaction and sharing of huge data sets without any delays. APIs majorly help different systems and applications to establish a connection and provide seamless communication among them. In the era of modernization, APIs are considered the heart of different sets of IoT devices and mobile applications as they help in initiating and controlling data flow.

Different organizations are readily accepting different APIs to gain seamless experiences in their work environment as well as provide support to the customers (De, 2023). The result of incorporating APIs in the internal systems increases the amount of exposure to huge datasets in the external environments. The exposure of large data sets in the external environments allows malicious attackers to facilitate attacks to gain unauthorized access to the systems. The main role of the APIs is to establish direct contact and connection between various internal systems and external environments which consists of different users and customers. These sets of interactions are associated with a huge number of sensitive data which are related to customer details, payment history, and other confidential information. The presence of sensitive information makes the systems the main target of attackers to gain malicious access into the system. This will finally result in disruption of the business environment as well as financial losses.

Using API security has no doubt enhanced the security levels of different systems and applications but there is a presence of different levels of vulnerabilities. There are wide ranges of vulnerability activities that emerged as the need to make enhancements in the security levels of different systems. Different sets of security flaws have been visualized which are related to insufficient authentication, validation of inputs, improper access control mechanisms and many more. These sets of high range of vulnerabilities are highlighted by the “OWASP API security Top 10” which states that there is a need to implement better security practices.

On the other hand, modern software is developed which readily depends on continuous integration and deployment methodologies. In this process, the sets of APIs are developed in the systems and software and continuously upgraded and the updated versions are released for use the customers or users. This is a challenging situation for the manual testers to design and initiate the testing process within a short time, resulting in the development of potential gaps and security flaws within the systems and the software. These sets of drawbacks of using manual testing methodologies have resulted in the sincere demand for automated testing practices which are readily scalable and efficient providing a high level of security in the absence of human testing interference. Automated forms of testing can automate different test cases in different environments and help in providing better sets of desired results (Paiva et al. 2022). It allows the security testers to provide a keen focus on other more difficult security activities instead of proving times on repetitive tasks.

1.2 Justification for the choice of the research

With the growing complexity of APIs and the need for continuous security testing, custom automation scripts provide a valuable tool for API penetration testing. This research is justified by the increasing demand for efficient, scalable, and real-time security assessments in modern environments.

Configuration and Setup: It involves the initiation of configuration phases for dealing with security tests. It also deals with integrating different sets of tools to facilitate the testing phase and development of test cases.

Custom Security Test Cases: The development of different test cases helps in determining different sets of potential risks and vulnerabilities that are present within the system.

Automation vs. Manual Testing: Detailed discussions are also made which focus on discussing the positive and negative side of manual testing over automated testing. It involves making comparisons by taking factors such as false and positive rates as well as testing rates.

Efficiency and Accuracy: It deals with discussing the efficiency of automated API security in providing an efficient platform to achieve more better level of security features for the systems and software.

Benefits and Limitations: Lastly it involves keeping a keen focus on the limitations that are present within the security testing tool majorly in the areas where high demand for API settings is required.

1.3 Gap in the literature it seeks to fill

This research fills the gap in evaluating the automation role as an **automated API security testing framework**. Existing literature focuses on tools like OWASP ZAP and Burp Suite, but lacks an in-depth examination of automation scripting capabilities and its integration within **CI/CD pipelines** for continuous testing.

1.4 Aim and Objective

Aim

The major aim associated with this research is to focus on the roles of automated API security which helps in automating penetrating testing for APIs and keeping a major focus on the development of different sets of test cases.

Objective

- To analyze the effectiveness of custom scripting in automating API security penetration testing by developing and executing tailored security test cases for identifying vulnerabilities such as Broken Object Level Authorization (BOLA) and Mass Assignment.
- To evaluate the benefits and limitations of automation scripts in API security testing compared to traditional manual testing methods, focusing on key factors such as time efficiency, accuracy, resource utilization, and scalability.

1.5 Research question

1. RQ1: How can custom scripting be leveraged as an automated tool for driving API security penetration testing, with a focus on creating custom security test cases?
2. RQ2: What are the benefits and limitations of using automation scripts for conducting API security penetration testing compared to traditional manual testing methods?

2 Related Work

2.1 API Security: An Evolving Threat Landscape

From the studies made by Vaghela *et al.*, (2024), APIs are considered as the major part of different software systems which helps in establishing communication between various sets of devices and platforms. With the increase in the demand for using APIs, it has been a major target for malicious attackers to theft information and cause harm to the system. Nowadays, every advanced system is integrated with APIs, making them the prime targets for attackers to perform malicious activities. The Open Web Application Security Project Top Ten List is listed as the third most attacked and the riskiest one. This results in different malicious activities like SQL injections, code execution, and other vulnerable attacks. Moreover, there

are sets of applications that are developed on the cloud platforms in which APIs can be considered as the major targets for the attackers to plan for malicious activities. Cloud computing environments readily rely on different sets of APIs and user interfaces as these systems help in establishing contact with cloud computing services. These sets of APIs used in the cloud services generally provide security to the systems. On the other hand, different sets of web applications are also dependent on APIs which makes them vulnerable to various advanced attacks.

According to Rao et al. 2020, the major shipment from the traditional process to integration of APIs has no doubt made advancements in the process of communication and has increased the need to develop a more secure environment to deal with advanced attacks. Generally, APIs provide sets of sensitive information public which makes them the prime targets for attackers to gain malicious access.

2.2 Manual Penetration Testing: Strengths and Limitations

From the argument made by Shah et al. (2020), manual forms of testing are considered the standard form of process in detecting wide ranges of vulnerabilities across different APIs. In this process, there are sets of security testers who design different test cases and perform attacks to gain information about the weaknesses present within the system. Instead of various sets of advantages, it has different limitations majorly in the areas where there is the presence of a large number of systems. Moreover, in the process of manual form of testing the groups of experienced testers design advanced test cases which helps in the early detection of flaws and vulnerabilities which may be lacking in modern automated testing. The manual testers provide a high level of flexibility and adaptability which is one of the positive forms of manual testing.

According to Gupta and Singh (2019), there are certain ranges of drawbacks of manual testing in the areas of APIs as manual forms of testing are associated with the usage of an intense number of human labourers as well as time-consuming.

2.3 The Role of APIs in Modern Software Architecture

As studied by Efuntade *et al.*, (2023), API can be defined as a set of functions which helps the developers gain access to the data and features integrated inside an application or software. API is used in developing different power systems such as maps, e-commerce platforms, social media platforms and many more. These are the sets of routine protocols or guidelines that help in standardizing of developing software which are compatible to the database or programs. APIs play a major role as the pillar of modern tech architectures, this develops an environment of interoperability among different software which is beneficial for different consumers and enterprises. The shift from the Monolithic architect to microservices has increased the demand of the APIs. This increase in demand has resulted in the development of Rest and Graph QL automatic programming interfaces. According to Lancos, (2021) The APIs are nowadays prime targets of attackers as large amounts of data are transferred with the APIs. API is also associated with the transportation of different sets of sensitive data which are required to be protected with advanced security mechanisms.

2.4 Common Vulnerabilities in API Security

As mentioned by Basheer *et al.*, (2024), APIs are associated with diverse varieties of vulnerabilities which result in unauthorised exposure of sensitive information and data. The most common API security risks as categorised by the OWASP include Broken Object level authorisation, SQL Injection and many more. Patan et al., (2021) states that, Broken Object Level Authorisation can be defined as the security risks having a lower level of access control

mechanisms which allows the attacks to gain unauthorised access into the system. These sets of vulnerabilities are the prime targets for the attackers to gain malicious access to the system. Malicious commands are executed to gain unauthorised access to the systems. These sets of security vulnerabilities arise due to the lack of access control mechanisms, security barriers, insecure configurations and many more. API often deals with diverse varieties of data and the transportation of the data across different systems. On the other hand, security issues may arise due to the insertion of improper inputs which can lead to different attacks such as SQL injections. In this type of Attack sets of malicious SQL queries are injected which results in the modification, retrieving and deletion of various important information and data. The presence of these wide ranges of vulnerabilities across different systems underlines the requirements to develop and integrate advanced security mechanisms across different APIs. Manual forms of testing will not be able to provide advanced security mechanisms while protecting the endpoints across complex situations and security vulnerabilities. Whereas the API security testing tools will help in providing advanced security to the APIs.

2.5 Evaluation Metrics for API Security Testing

From the perspectives of Abdelfattah *et al.*, (2024), for dealing with the evaluation phase of API security testing development of metrics is required which will help in detaining the accuracy of the test cases. Various sets of metrics include False positives, coverage and many more which are used in assessing the effective working of the security tools. the rate of detection with the help of identified metrics will help in the frequency with which sets of vulnerabilities are detected. The delivery of high false negative rates will indicate that some sets of vulnerabilities are missed. The accuracy of the API security testing tools is essential in the detection of a wide range of unknown threats and vulnerabilities. According to Baniaş *et al.*, (2021), response time is also another key metrics which helps in evaluating the time required for commencing tests. In the CI/CD environments various sets of test cases need to be commenced in a short time or else delays can take place during the phases of deployments. The ability of the different security testing tool to commence test cases in a short period makes an effective choice for developers.

3 Research Methodology

The section will focus on the applied methods while dealing with the phase of development of the testing frameworks. The main aim associated with this is to deal with the objectives in the identification and mitigation wide range of API vulnerabilities with the help of an API security testing tool. Different diagrams will help in the illustration of the workings and different phases associated with the development and implementation.

3.1 Requirements and Contextual Analysis

Purpose: In the first phase which deals with the identification of a wide range of security requirements in the APIs. Various common vulnerabilities are specified such as BOLA, SQL Injection, Input validation and Mass assignment (Condal Fontanet, 2023). In the traditional methods are associated with the exploration of different sets of vulnerabilities manually. The manual phases are generally time-consuming and have risks associated with the development of manual errors. To address these sets of challenges automation methods are chosen or better solutions.

Methodology: This stage is associated with the collection of different requirements from the APIs, it includes the vulnerabilities which are listed in the OWASP API security top 10 list. This helps in the identification of risky areas related to vulnerability testing. These

frameworks are further reviewed to generate different security test cases that can be used in the automation processes.

Justification: The developed test cases help in dealing with overall requirements and commence with speed. This helps in dealing with the phases of checking all sets of vulnerabilities in less amount of time. The developed test cases help in targeting specific areas having higher risks.

3.2 API Documentation and Endpoint Identification

In the next phase various sets of API documentation are gathered which are required for a detailed understanding of the overall structure, and potential vulnerabilities present across different endpoints of the APIs. The process of data gathering is also an identification process which helps in designing and developing security test cases.

Different sets of API documentation are used to gather more details related to the parameters, and structures associated with the responses and requirements for the process of authorisation. To commence with the test phase of BOLA and Mass assignment scenario different sets of endpoints are identified. The testing of the APIs helps in the verification of unauthorised manipulation by different users.

The data collected from various API documentation helped in the development of different test cases (Viglianisi *et al.*, 2020). This helps in dealing with the higher accuracy of the automated test cases and lowers the rate of abnormalities.

3.3 Test Environment Configuration

The main purpose of this phase is to prepare data which are essential to deal with the phases of the configuration of different varieties of environmental variables. These ranges of variables have essential requirements for the development of different test case scenarios. In this phase, various object IDs are generated for dealing with the phases of tests and saved in environment file.

Environment Variables: It includes the setting of different sets of environmental variables such as API tokens, URLs and session cookies in the security testing tool. This helps in the effective execution of the security test cases without the need for manual modification.

Random ID Selection: In this phase different random IDs are automatically generated for simulating test cases like IDOR. These sets of object ID are generated automatically by the attackers as the valid IDs are not known to them.

Header Manipulation: Various ranges of custom headers are generated which helps in developing requests for testing different vulnerabilities across the APIs.

The preparation of different data and the overall process of standardisation help in the development of the test cases. This varied process will help in commencing the test cases across different environments. This will also reduce the need to incorporate manual changes.

3.4 Test Case Development

This phase is the core area which is related to the development of different test cases for detecting sets of vulnerabilities. These sets of developed test cases are conducted in API security testing tool.

Mass Assignment (Unauthorized Field Injection) Along with Broken Access Control: In this type of test case different fields such as in our case “*isAdmin*” parameter was modified to ‘*false*’ as initially it was set to ‘*true*’ check where unauthorised changes can be made or not. Unauthorized POST request was constructed by manipulating authentication credentials here in my case without the authentication password. This request was sent to restricted API endpoint to assess whether the system allows unauthorized users to create resource or not.

This process helps identify weaknesses in the access control mechanisms, ensuring that only authorized users can perform privileged operations.

SQL Injection: There are diverse varieties of SQL payloads that were integrated into the parameters to deal with the SQL injection attacking phases.

Broken Object Level Authorization (BOLA): Wide ranges of IDs were delivered with different a GET request which helps in analysing whether unauthorised access was provided or not (Ball, 2022).

Input Validation: A wide range of incorrect data was inserted into the requests to check the the APIs were able to validate the accurate inputs or not.

The developed test cases help in the analysis of the wide range of vulnerabilities that can be used by the attacker to gain unauthorised access. The phase of automation helps in the early detection of vulnerabilities. The different varieties of test cases are developed with accuracy which focuses on testing of different endpoints.

3.5 Implementation Using Postman

This phase deals with the implementation of the test cases with the help of API security testing tool. It provided various sets of core tools which helped in the development and execution of the planned test cases.

Different API security testing collections are developed to deal with each of the test case scenarios. The requests are organised with the help of different headers, scripts and payloads. JavaScript codes are generated to adjust the sets of requests and help in the execution of the test cases across different environments (Loikkanen, 2024).

The API security testing tool allows in execution of the test cases automatically. Moreover, new codes are implemented resulting in prodigy feedback related to errors or security issues.

3.6 Testing and Evaluation

This phase is associated with the commencing of the developed test cases and detailed evaluations are made based on the results gathered. Various sets of performance metrics are also visualised to detail the quality and effectiveness of the developed test cases.

Methodology:

Execution: The developed test cases are executed in the API security testing (Postman) tool environment and the responses are addressed in each of the scenarios.

Performance Metrics: Various key metrics are recorded which include false rates, deduction rates and many more. For example, 200 OK status in the API security testing tool indicates the presence of BOLA vulnerabilities.

Response Analysis: The gathered responses are analysed to gather more information related to the identification of malicious attempts and blockage of the malicious inputs.

The evaluation of the performances helps in gathering more insights related to the effectiveness of automated testing as compared to the manual forms. The constant detection and reduced false positives are important for providing surety to the effectiveness of different test cases.

3.7 Pseudocode

A. Testing for Mass Assignment Along with Broken Access Control

// Endpoint: <https://restful-booker.herokuapp.com/booking> (POST)

// Input: Payload with restricted fields

// Output: Response indicating if unauthorized update was successful

START

1. Initialize request payload by changing an unauthorized field:

```
payload ← {  
  For eg. //In body  
  "isAdmin": true, // Unauthorized field change the value to 'false' for testing  
}
```

2. Send the POST request:

```
response ← POST("https://restful-booker.herokuapp.com/booking", payload)
```

3. Check the API response:

```
IF response.status == 200 then  
  PRINT "Request was successful."  
  IF "isAdmin" in response.body OR "role" in response.body then  
    PRINT "Vulnerability found: Unauthorized field modification detected."  
  ELSE  
    PRINT "No vulnerability: Unauthorized fields were ignored."  
  ELSE  
    PRINT "Request failed with status: ", response.status
```

4. Report the vulnerability:

```
IF unauthorized fields are accepted then  
  PRINT "Mass Assignment Vulnerability confirmed. Report this issue."
```

END

B. SQL Injection Vulnerability Detection

// Endpoint: <https://restful-booker.herokuapp.com/booking/{{parameters}}> (GET)

// Input: SQL payload strings injected into URL parameters

// Output: SQL errors or unexpected results indicating vulnerability

START

1. Initialize SQL injection payloads:

```
payloads ← ["Choose your own SQL payload/list"  
]
```

2. Identify vulnerable field:

```
target_field ← "id"
```

3. FOR each payload IN payloads DO

3.1. Construct API request:

```
url ← "https://restful-booker.herokuapp.com/booking/" + payload
```

3.2. Send GET request with payload:

```
response ← GET(url)
```

3.3. Monitor response:

```
IF response.status == 200 OR "SQL error" in response.body then  
  PRINT "SQLi Vulnerability found with payload: ", payload  
  EXIT  
ELSE
```

```
  PRINT "No vulnerability with payload: ", payload
```

4. IF no vulnerabilities found THEN

```
  PRINT "SQL Injection vulnerability not detected."
```

END

C. Input Validation Vulnerability Detection

// Endpoint: <https://restful-booker.herokuapp.com/booking/{{parameters}}> (PUT)

// Input: Invalid data payload with incorrect data types

// Output: Response indicating validation error or potential vulnerability

START

1. Prepare invalid data type and its value payload:

```
payload ← { for eg.  
  "firstname": "John",  
  "lastname": 12345, // Try to change the data type to invalid data type for 'lastname'  
}
```

2. Set up the API endpoint:

```
url ← "https://restful-booker.herokuapp.com/booking/{parameters}"
```

3. Send API request:

```
response ← PUT(url, payload)
```

4. Track the server's response:

```
IF response.status == 400 then  
  PRINT "Validation error detected: Input data rejected correctly."  
ELSE IF response.status == 500 then  
  PRINT "Internal Server Error: Validation logic may be broken."  
ELSE  
  PRINT "Unexpected behavior: Status - ", response.status
```

5. Assess vulnerability:

```
IF response.status == 500 OR input accepted unexpectedly then  
  PRINT "Validation vulnerability found: Input validation is not enforced."
```

END

D. Broken Object Level Authorization (BOLA) along with IDOR Vulnerability Detection

// Endpoint: <https://restful-booker.herokuapp.com/booking/{id}> (GET)

// Input: Random ID without authentication

// Output: Response indicating unauthorized access

START

1. Select a random object ID:

```
object_id ← random(1, 1000) //Choose/set any random id from 1-1000
```

2. Set up the API endpoint:

```
url ← "https://restful-booker.herokuapp.com/booking/" + object_id
```

3. Send the GET request without authentication:

```
response ← GET(url)
```

4. Track the server's response:

```
IF response.status == 200 then  
  PRINT "BOLA vulnerability found: Access granted to object ", object_id  
  PRINT "Response Data: ", response.body  
ELSE IF response.status == 401 OR response.status == 403 then  
  PRINT "Access denied: Proper authorization required."  
ELSE  
  PRINT "Unexpected behavior: Status - ", response.status
```

5. Assess vulnerability:

```
IF response.status == 200 then  
  PRINT "BOLA confirmed: Sensitive data accessible without authentication."
```

END

3.8 Automated API security Architectural Diagram

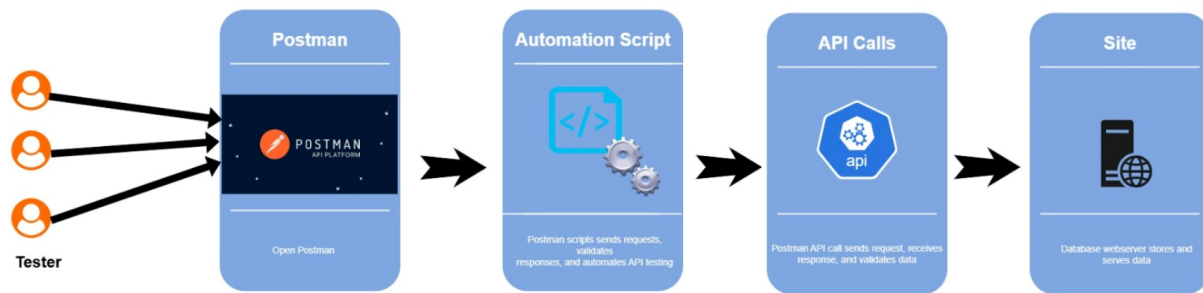


Figure 1: Architectural Diagram of API Security Automation

(Source: Self-Made)

As seen in the figure 1 the tester launches postman application and writes custom scripts for security testing according to the functionality of the API at collection level and individual API's Pre Request and Post response tab. The scripts are then run with collection runner tab giving results of PASS and FAILED (Detected Vulnerabilities) security cases.

3.9 Comparison, Benefits, and limitations (Manual Vs Automated testing)

Workflow Comparison	Manual Testing	Automation Testing
Setup and Initialization	Time-consuming and repetitive (van der Poel, 2022).	Dynamic with environment variables (Ranta, 2023)
Execution	Manual request crafting for each endpoint (van der Poel, 2022).	Automated execution across multiple endpoints
Analysis	Labor-intensive response analysis (van der Poel, 2022).	Automated logging and analysis
Documentation	Manual report generation (van der Poel, 2022).	Automated report generation
Scalability	Limited	Highly scalable

4 Design Specification

The designed code is about the automation of different detection techniques for vulnerabilities. The phase of implementation of this code is associated with API security testing tool which is an API testing interface (Westerveld, 2021), to develop a custom security test case java script is used. The below section will describe the techniques, architectures and other areas of the different test cases.

4.1 Techniques and Architecture

4.1.1 Mass Assignment with Broken Access Control:

Technique: In this phase, vulnerabilities are detected with the insertion of different unauthorised data into the POST method. In this process the system returns a payload which includes fields such as “role” and “isAdmin”, these roles cannot be modified by an unknown

user and later the responses of the API are recorded. Along with this requests was crafted without valid authentication credentials to determine if the API allows unauthorized users to create resource. The test evaluates whether the API correctly enforces access control by rejecting unauthorized requests.

Framework: To deal with this process API security testing tool is used for sending the requests and check the API allows unauthorised access or not. On the other hand, JavaScript is used for monitoring the gained responses. The response handling process assesses whether the API returns appropriate status codes (e.g., 401 Unauthorized or 403 Forbidden) to indicate proper access control enforcement, or if it erroneously processes the request, revealing a potential vulnerability. The flow is described visually in figure 2.

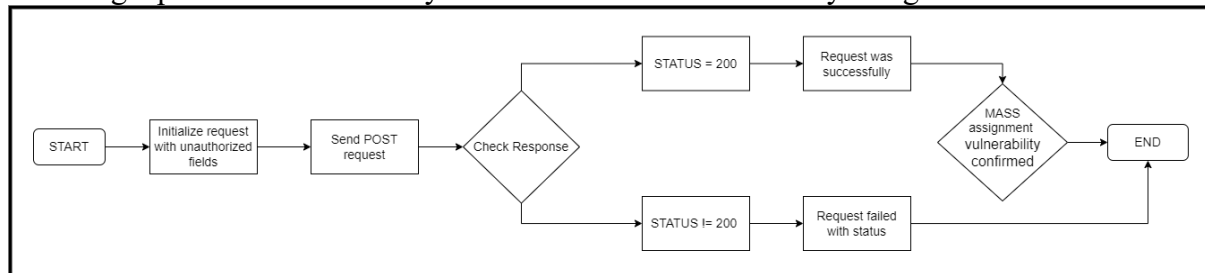


Figure 2: Flowchart – Mass Assignment with Broken access control

4.1.2 SQL Injection Vulnerability Detection:

Technique: In this testing phase different vulnerabilities are assessed which are related to the injection of malicious SQL codes. In this test phase payloads are crafted in the URL and further evaluations are made based on the response.

Framework: In this phase, API security testing tool is used for sending the unauthorised payloads and the role of JavaScript is to automate the process of construction of requests. Further, the responses are analysed for the SQL errors generated. The flow is described visually in figure 3.

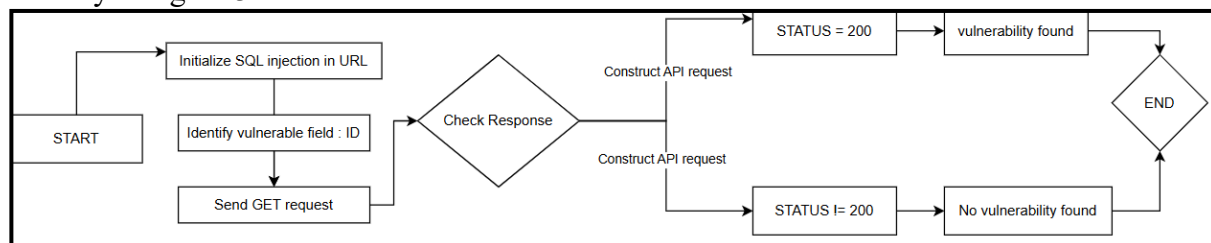


Figure 3: Flowchart – SQL Injection Vulnerability Detection

4.1.3 Input Validation Vulnerability Detection :

Technique: in this test phase codes are designed for checking the invalid inputs with the help of sending PUT requests in the security testing tool. The PUT request consists of malicious or unauthorised datatypes. In this scenario, integers are given in place of strings.

Framework: In this scenario JavaScript and the security testing tool send unauthorised or unknown statements into the API, it checks whether the API makes a response in rejecting the malicious data or not. The mechanism flow is shown in figure 4.

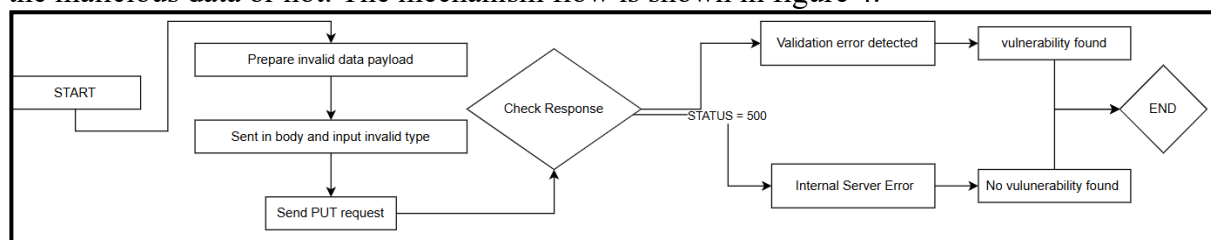


Figure 4: Flowchart – Input Validation Vulnerability Detection

4.1.4 Broken Object Level Authorization (BOLA) Vulnerability Detection:

Technique: In this type of vulnerability, API provides access to unauthorised objects without making detailed verification of the authorisation of the user. A random object ID is selected and attempts are made to gather information without any authorisation. In this testing scenario, various attempts are developed to gain access to as certain restricted resources with the help of different objects that are not authenticated.

Framework: As per figure 5 which shows flow of the mechanism of API security testing tool which sends requests to access randomly chosen objects. JavaScript analyses the responses associated with unknown access without proper authorisation.

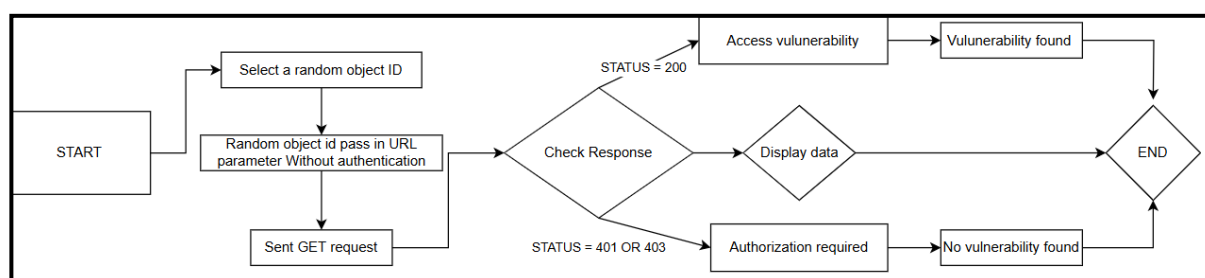


Figure 5: Flowchart – Broken Object Level Authorization Vulnerability Detection

4.2 Associated Requirements:

API Access: To deal with this task different endpoints are required along with an authorisation token to deal with authenticated testing.

Custom Test Cases: Proper test case definitions are required for each testing scenario to target vulnerabilities.

Automation: API security collections are made which helps in the proper organisation of different sets of test cases. Single collections made consist of different API endpoints that are further integrated with payloads, headers and many more.

Error Handling: The integration of JavaScript in the security testing tool helps handle and analyse responses made by API. Scripts are developed which is used in checking the HTTP status code for errors

5 Implementation

5.1 Tools and Languages:

Postman: API requests are developed and sent with the help of Postman. Moreover, different HTTP methods are also handled and responses are analysed.

JavaScript: It is used in Postman's scripting area to automate different requests. The responses gathered are further analysed for the presence of various security weaknesses.

JSON: This is the used format for developing payloads and processing various responses made by the API.

5.2 Outputs Produced:

1. Transformed Data: From the request generated from the API data are responses that consist of different status codes, messages and many more. The output data is further analysed for the detection of vulnerabilities. Vulnerabilities such as unauthorised access, malicious SQL code injection and others. In the case of the test scenario SQL injection errors or other responses are visualised which notifies that the vulnerabilities are present or not.

2. Code Written: For dealing with the phase of automation modifying API requests javascript is used. API requests such as injection of malicious fields, SQL queries, unauthorised data and many more. The scripts developed are integrated inside the Postman to handle different requests and validate the responses provided by the API. In the case of the first test case, it is used to detect unauthorised fields which were injected. In the case of the second scenario injection of malicious queries were used with results of SQL errors are present or not .

3. Test Cases Developed: Five test cases are designed which are unique in their types, these test cases are integrated with custom logic at their endpoints of the APIs.

4. Report Generation: Postman is used to generate reports with detail summaries associated with the developed test cases. The report contains a scenario of the execution of the tests.

5.3 Results

5.3.1 From Figure 6 which shows Mass Assignment Vulnerability Detection: By changing an unauthorized field here in our case “isAdmin” was set to true we changed to “false”

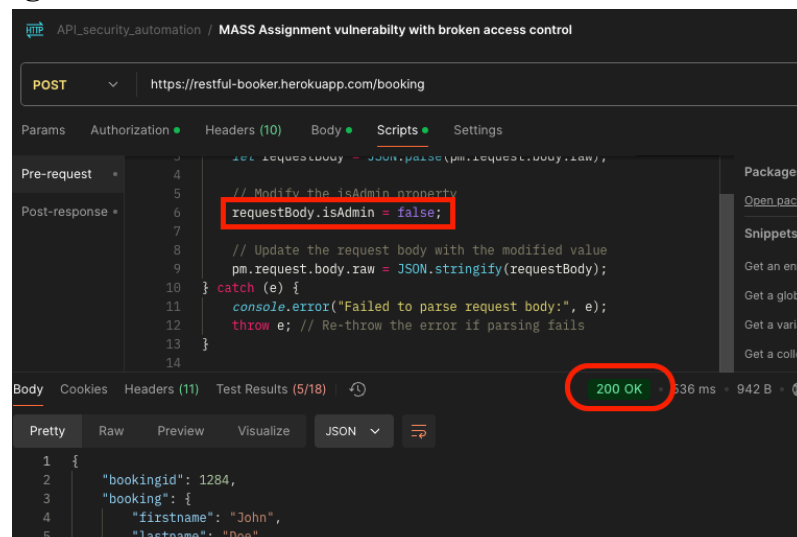


Figure 6: Mass Assignment vulnerability Detected as we got response as 200 ok

5.3.2 Figure 7 Demonstrating SQL Injection Vulnerability Detection:

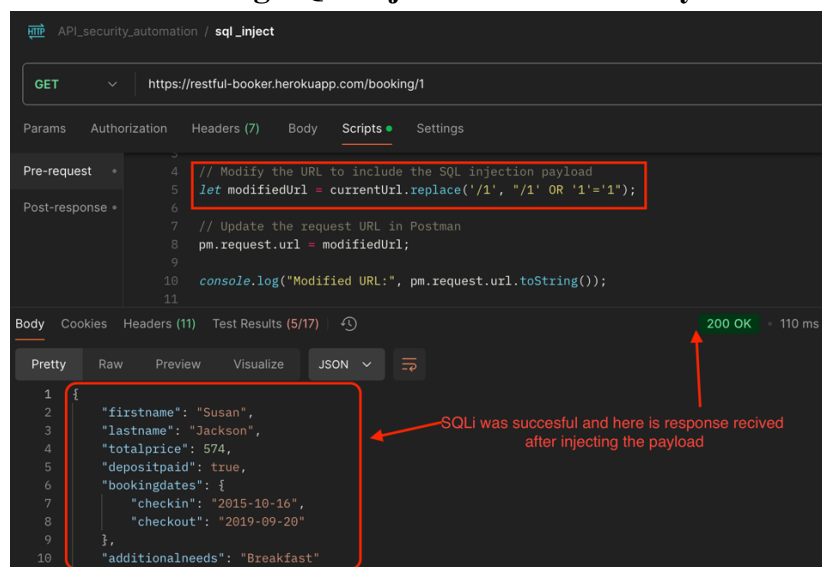


Figure 7: SQL Injection Vulnerability Detected

5.3.3 Figure 8 shows Input Validation Vulnerability detection: Changed the value of the 'lastname' parameter to integer value instead of string.

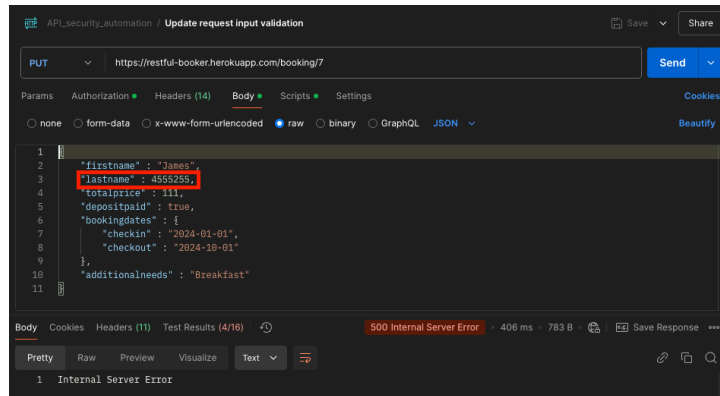


Figure 8: Input Validation vulnerability Not detected as response received was 500.

5.3.4 Figure 9 demonstrating Broken access control (Password kept blank) for authorization of creating a booking (POST request) still was send the request with authorization

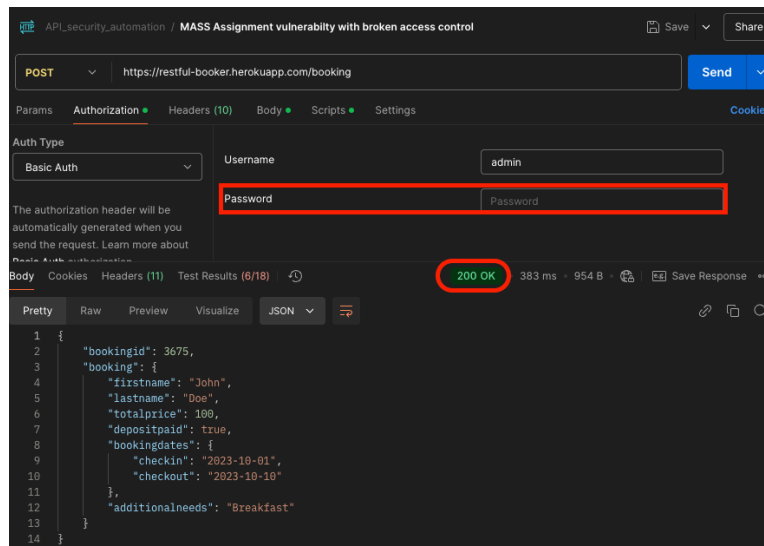


Figure 9: Broken Access Control Vulnerability Detected

5.3.5 From figure 10 which shows Broken Object Level Authorization was found as the booking ID's GET call's functionality should be with authorization but found to be without it.

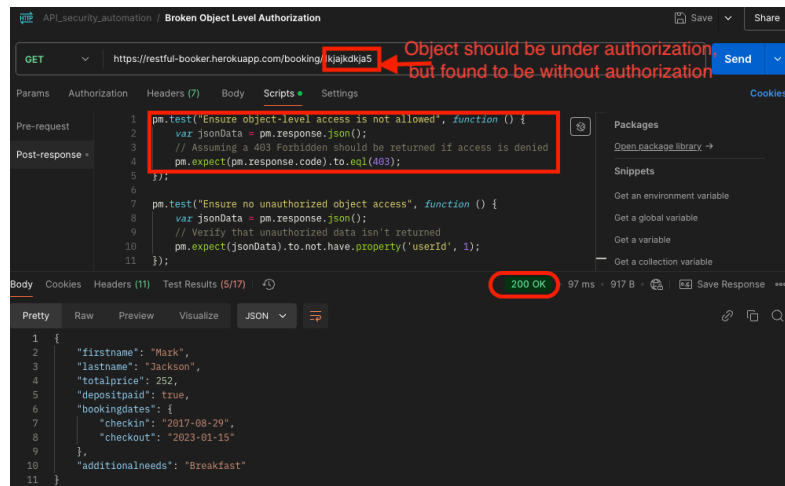


Figure 10: Broken Object level Authorization Vulnerability Detected

5.3.6 Figure 11 Describing Insecure Direct Object Reference (IDOR) Vulnerability Detection. By just changing any user's booking ID one can see his details such as 'totalprice', 'bookingdates'

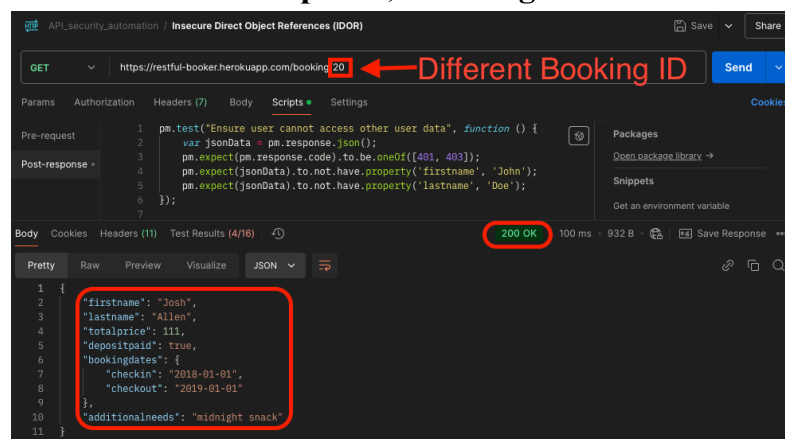


Figure 11: IDOR Vulnerability Detected

5.3.7 Figure 12 showing Runner Tab Collection Results which shows automated security test cases runs for each API call here in my case I have shown result of one API call.

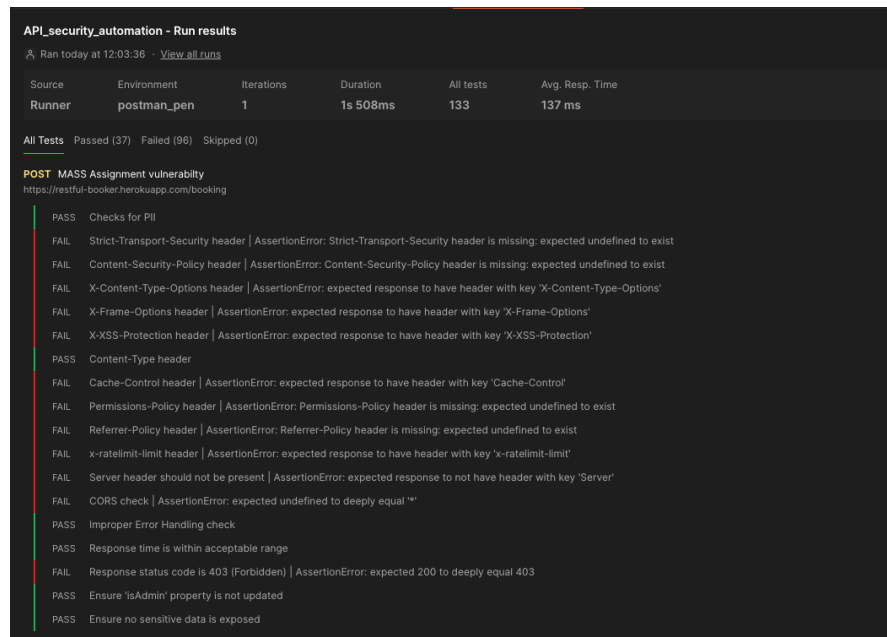


Figure 12: Postman Automation Runner Tab Results

6 Evaluation

6.1 Time Efficiency

From the Figure 13 an automated form of testing is considered effective due to the requirement of less time to commence with different test cases (Liu *et al.*, 2024). The manual form of testing requires time for inspecting requests and its responses across different endpoints. On the other hand automated form of testing helps ease the execution of various repetitive tasks that are present across different endpoints.

Comparison:

- **Manual Testing:** This form of testing process requires a high amount of time to commence with tests per endpoint (Yandrapally *et al.*, 2023). It requires an approximate amount of 30 to 60 minutes of time and this time varies based on the complexities.
- **Automated Testing:** This form of testing requires less amount of time due to the incorporation of automated scripts. It requires 5 to 10 minutes of the period to commence with test cases per endpoint.

Graph 1: Figure 13 representing -Time Taken per Endpoint (Manual vs. Automated Testing)

X-axis: Number of Endpoints

Y-axis: Time (minutes)

End Points	Time for manual testing	Automated testing time
1	5.38	0.016
4	30	0.033
7	45	0.066

Graph Representation:

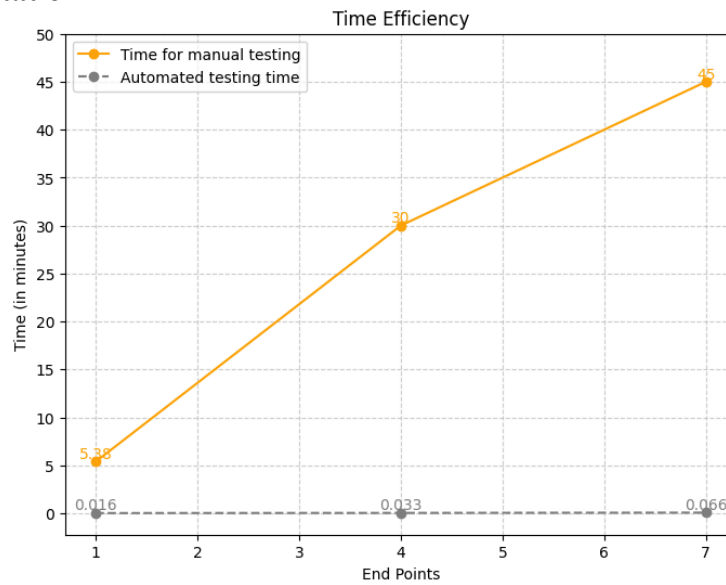


Figure 13: Time Efficiency Graph

6.2 Attack time consumed Manual VS Automated testing.

From Figure 14 it is clear that **Automated testing** significantly reduces the time required to detect common vulnerabilities. In contrast, **manual testing** relies on human expertise, making it a labour-intensive process that takes considerably longer to execute and analyse. Manual testing is often slower due to the need to manually create requests, analyse responses, and document findings as show in figure 14.

Comparison:

- **Manual Testing:** Manual testing is time-consuming, requiring an average of 30–45 minutes per endpoint (Average Self tested time for 5 iterations) depending on the complexity of the API security test cases to be performed. The process involves manually crafting and sending requests, monitoring responses, and logging results. development environments.
- **Automated Testing:** Automated testing significantly reduces attack time, with each endpoint tested in a short amount of time. Automated scripts can run multiple tests concurrently and analyse results automatically, enabling faster identification of vulnerabilities. This efficiency makes automated testing well-suited for integration into CI/CD pipelines.

Graph 3: Figure 14 showing Attack time consumed Manual VS automated testing

Attacks	Manual	Automated
Security Headers	1.5	0.0083
Mass assignment	2	0.033
Broke Object Level Authorization	5	0.033
Input (I/P) validation	2	0.016
Broke Access Control	5	0.025

Graph Representation:

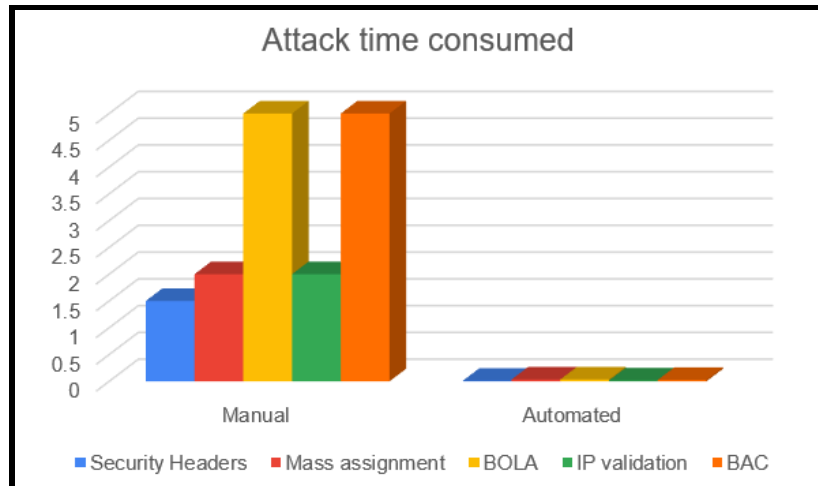


Figure 14: Manual VS Automated Attack time consumption

6.3 Cost Implications

An automated form of testing requires extensive costing and investments in the prior phases due to the complexities of various processes like configuration and setup (Moseh *et al.*, 2024). This form of testing provides long-term effectiveness in providing better testing results. Manual form of testing in the initial phases requires a lower amount of costs but after a certain period it increases and requires extensive costs and it is dependent on human resources.

Comparison:

- **Manual Testing:** This type of testing has the requirement of long-term investments because of the involvement of a high number of human resources to commence with various testing phases.
- **Automated Testing:** It requires less long-term costs due to automated configuration and reduction in the requirement of human resources.

6.4 Computational resources

Before Execution: Figure 15 showing utilization of resources.

Time	CPU(%)	Memory(mb)
0.01041865349	4.6	80.6
1.01658535	10	80.4
2.021544695	5.1	80.4
3.033410072	9.9	80.3
4.126435518	27	80.7
5.135279417	35.4	80.9
6.142272711	6.9	80.9
7.148488045	2.3	80.9
8.154577732	5.8	80.8
9.163725376	5.9	80.6

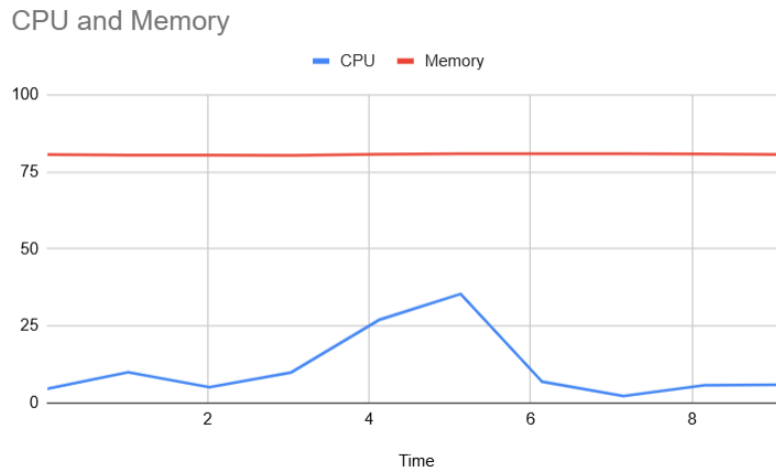


Figure 15: Before Execution CPU & Memory Consumption

After Execution: Figure 16 showing utilization of resources.

Time	CPU (%)	Memory(mb)
0.006520986557	17.9	80.3
1.089769363	38.5	80.8
2.09533906	41.5	81.2
3.104598761	20.3	81.3
4.115250349	56.1	81.4
5.125889063	55.8	81.4
6.136854649	41.2	81.5
7.14413166	30.4	81.7
8.155305386	36.7	81.9
9.162173271	11.5	81.9

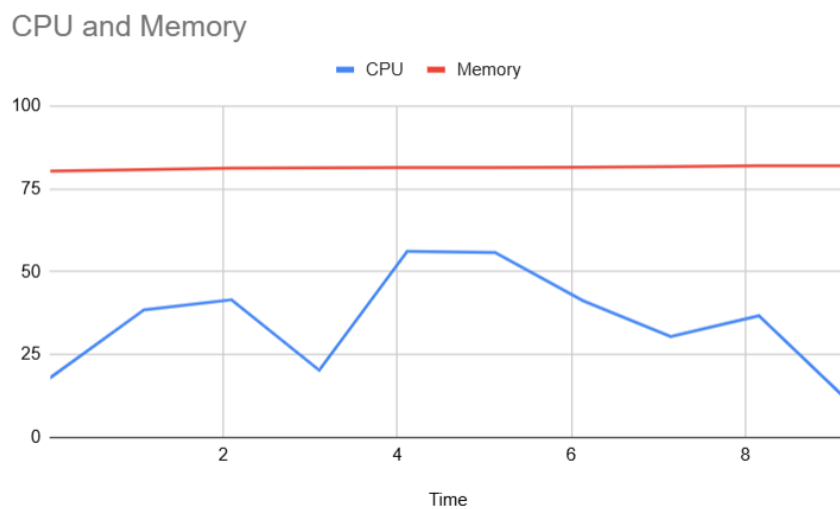


Figure 16: After Execution CPU & Memory Consumption

6.5 Discussion:

This research demonstrates the effectiveness of implementing automated API security testing frameworks in tackling the research questions. We were able to develop and run test cases for critical vulnerabilities such as Broken Object Level Authorization (BOLA), Mass Assignment along with broken access control, IDOR, and SQL Injection by using Postman with custom scripts. These direct implementations correspond to the stated objectives and illustrate how automation facilitates the process of vulnerability detection while being less resource-intensive and minimizing human mistakes.

Our solutions bring significant improvement in time efficiency, scalability, and accuracy over a manual testing method. Automating endpoint testing reduced its time from 30-45 minutes (from figure no. 13) to under 5 minutes, thus improving productivity. Moreover, the use of custom scripts has ensured high detection rates while minimizing false positives.

This approach well answers the research questions because it depicts how automation tackles the challenges of scalability and complexity in API security. The results underscore that automation security is a robust solution for modern fast-paced API development environments.

7 Conclusion and Future Work

7.1 Restating the Research Question and Objectives

Research Questions

- RQ1: How can custom scripting be leveraged as an automated tool for driving API security penetration testing, with a focus on creating custom security test cases?
- RQ2: What are the benefits and limitations of using automation scripts for conducting API security penetration testing compared to traditional manual testing methods?

Objectives

- To analyse the effectiveness of custom scripting in automating API security penetration testing by developing and executing tailored security test cases for identifying vulnerabilities such as Broken Object Level Authorization (BOLA), Mass Assignment, IDOR, Input Validation and SQLi.
- To evaluate the benefits and limitations of automation scripting in API security testing compared to traditional manual testing methods, focusing on key factors such as time efficiency, accuracy, resource utilization, and scalability.

Summary of Work Done

7.2 Key Findings and Implications

7.2.1 Key Findings

Efficiency: The use of API security testing tool to commence the process of automated testing helped in the reduction of time and lowered the number of human resources. Different sets of tests which are executed manually require extensive time and this automated form of testing requires less amount of time.

Coverage and Accuracy: In the case of automated testing, a higher percent accuracy and less time consumption is observed. On the other hand, in the case of the manual form of testing, the rate of time was higher and subject to human error (refer figure no. 13). Automated testing mechanisms helped in the reduction of false positive rates due to the incorporation of automated scripts.

Limitations: API security testing tool is useful in the detection of different sets of common vulnerabilities but the absence of different sets of advanced features which are essential for the detection of zero-day vulnerability and finding of advanced vulnerable vectors.

7.2.2 Implications

The research provided a detailed evaluation of the potentiality of automated tools which are associated with addressing different consistency and scalability challenges that are majorly faced by manual testing processes. From the findings, it can be ensured that there is a higher level of importance of integration of automated testing mechanisms for the achievement of better workflow. With the help of API security testing tool different organisations can able to make advancements across security tests.

7.3 Efficacy of the Research

The research helps in effectively answering different research questions and effective alignment with various objectives. In this case of automated testing, API security testing tool is one of the effective tools which helps with automated API security testing. The automated framework helps in the effective delivery of positive test cases and better results. There are certain limitations of the research which include

- Limited scripting processes across API security testing tool provided a high range of implications while dealing with complex vulnerabilities.
- Various sets of advanced scenarios are effectively addressed with the help of different sets of specialised tools and technologies.

7.4 Limitations

Tool-Specific Constraints: While API security testing tool supports scripting, it lacks advanced security features, such as traffic interception and active scanning, available in tools like Burp Suite.

Narrow Scope of Vulnerabilities: The study focused on a predefined set of vulnerabilities, leaving out more complex or lesser-known attack vectors.

Context-Specific Results: The framework was tested on specific APIs, and results may vary depending on the API architecture, complexity, and configuration.

7.5 Future Work

Advanced Vulnerability Detection: In the coming future other sets of advanced tools and technologies can be integrated with the API security testing tool for addressing different limitations. An effective combination of Burp Suite with API security testing tool will help in the development of an advanced framework which will help in the detection of advanced vulnerabilities.

Machine Learning for Dynamic Testing: Integration of advanced machine learning algorithms with API security testing will help in dealing with diverse varieties of threats (Chen and Babar, 2024). With the help of machine learning models patterns across traffic can be analysed for detection of various vulnerabilities.

Continuous Security testing in CI/CD Pipeline: Enhancing research for integrating this framework into CI/CD pipeline such as GitHub actions to automatically test the application every time when a release is pushed.

8 References

Abdelfattah, A.S., Cerny, T., Yero, J., Song, E. and Taibi, D., 2024. Test Coverage in Microservice Systems: An Automated Approach to E2E and API Test Coverage Metrics. *Electronics*, 13(10), p.1913.

Ball, C.J., 2022. *Hacking APIs: Breaking Web Application Programming Interfaces*. No Starch Press.

Baniaş, O., Florea, D., Gyalai, R. and Curiac, D.I., 2021. Automated specification-based testing of REST APIs. *Sensors*, 21(16), p.5375.

Basheer, N., Islam, S., Alwaheidi, M.K. and Papastergiou, S., 2024. Adoption of Deep-Learning Models for Managing Threat in API Calls with Transparency Obligation Practice for Overall Resilience. *Sensors*, 24(15), p.4859.

Basics of JavaScript and Modern Web App Development (pp. 223-258). Berkeley, CA: Condal Fontanet, J.O., 2023.

Analysis of web applications penetration testing and its realization (Master's thesis, Universitat Politècnica de Catalunya) creating, testing, and managing APIs for automated software testing. Packt Publishing Ltd.

De, B., 2023. Build APIs as a Product. In *API Management: An Architect's Guide to Developing and Managing APIs for Your Organization* (pp. 365-383).

Berkeley, CA: Apress. education: A state-of-the-art review. *ACM Transactions on Computing Education (TOCE)*, 22(3), pp.1-40.

Efuntade, O.O., Efuntade, A.O. and FCIB, F., 2023. Application Programming Interface (API) And Management of Web-Based Accounting Information System (AIS): Security of Transaction Processing System, General Ledger and Financial Reporting System. *Journal of Accounting and Financial Management*, 9(6), pp.1-18.

Lancos, P., 2021. Transforming the future of digital banking with APIs and DataSecOps. *Journal of Digital Banking*, 6(3), pp.270-276.

Loikkanen, I., 2024. Improving End to End Testing of a Complex Full Stack Software.

Paiva, J.C., Leal, J.P. and Figueira, Á., 2022. Automated assessment in computer science.

Patan, R. and Parizi, R.M., 2023, July. Automatic Detection of API Access Control Vulnerabilities in Decentralized Web3 Applications. In *2023 IEEE International Conference on Decentralized Applications and Infrastructures (DAPPS)* (pp. 76-85). IEEE.

Rao, S., Rajesh, M., & Singh, T. (2020). Microservices architecture and its impact on API security. *Journal of Cloud Computing and Security*, 17(2), 133-145.

Shah, M., Patel, J., & Mehta, A. (2020). Manual vs automated penetration testing: Evaluating the pros and cons in API security. *Cybersecurity Strategies*, 16(3), 100-112.

Vaghela, R.A., Solanki, K., Popat, R.R., Vaghela, I.R. and Chhangani, N., 2024. Usage of Modern API for Automation of Government Procedures. In *Transforming Public Services—Combining Data and Algorithms to Fulfil Citizen's Expectations* (pp. 131-150). Cham: Springer Nature Switzerland.

Viglianisi, E., Dallago, M. and Ceccato, M., 2020, October. Resttestgen: automated black-box testing of restful apis. In 2020 IEEE 13th International Conference on Software Testing, Validation and Verification (ICST) (pp. 142-152). IEEE.

Westerveld, D., 2021. API Testing and Development with Postman: A practical guide to creating, testing, and managing APIs for automated software testing. Packt Publishing Ltd.

Liu, J., Xia, C.S., Wang, Y. and Zhang, L., 2024. Is your code generated by chatgpt really correct? rigorous evaluation of large language models for code generation. *Advances in Neural Information Processing Systems*, 36.

Yandrapally, R., Sinha, S., Tzoref-Brill, R. and Mesbah, A., 2023, May. Carving ui tests to generate api tests and api specification. In 2023 IEEE/ACM 45th International Conference on Software Engineering (ICSE) (pp. 1971-1982). IEEE.

Chakraborty, S., Krishna, R., Ding, Y. and Ray, B., 2021. Deep learning based vulnerability detection: Are we there yet?. *IEEE Transactions on Software Engineering*, 48(9), pp.3280-3296.

Ranta, J., 2023. *Testing AWS Hosted RESTful APIs with Postman*.

Manikas, K. and Manikas, K., 2016. Revisiting software ecosystems research. *Journal of Systems and Software*. Available at: <https://dlnext.acm.org/doi/abs/10.1016/j.jss.2016.02.003> (Accessed: 12 December 2024).

Moseh, M.A., Al-Khulaidi, N.A., Gumaei, A.H., Alsabry, A. and Musleh, A.A., 2024, August. Classification and Evaluation Framework of Automated testing tools for agile software: Technical Review. In 2024 4th International Conference on Emerging Smart Technologies and Applications (eSmarTA) (pp. 1-12). IEEE.

Chen, H. and Babar, M.A., 2024. Security for Machine Learning-based Software Systems: A Survey of Threats, Practices, and Challenges. *ACM Computing Surveys*, 56(6), pp.1-38.

van der Poel, L., 2022. Towards automated discovery of access control vulnerabilities (Doctoral dissertation, Department of Software Technology Faculty EEMCS, Delft University of Technology).