API Security Testing: The Challenges of Security Testing for Restful APIs

Sattam J Alharbi[1], Tarek Moulahi[2]
Department of Information Technology,
College of Computer, Qassim University, Saudi Arabia

Abstract:- Modern web applications and software systems have shifted to relying on RESTful APIs, which are more susceptible to security threats such as injection attacks, authentication attacks, and data breaches. This article discusses the difficulties of performing security testing on RESTful APIs, such as input validation, authentication, and authorisation. It has been identified that vulnerabilities that affect security configuration include insufficient logging, faulty object-level authorisation, asset management, faulty function-level authorisation, and mass assignment. It concludes by summarising the findings and offering suggestions for maintaining the security of RESTful APIs using previous research studies.

Keywords:- *API security testing; RESTful APIs; Security challenges; API security vulnerabilities; Security testing techniques; API security practices.*

I. INTRODUCTION

Security has emerged as a significant worry due to the extensive use of RESTful APIs in contemporary software development. Many web and mobile apps depend on RESTful APIs to facilitate seamless data interchange and communication between platforms (Carlos Rodrguez et al., 2016). These APIs are, however, susceptible to several security risks, such as problems with authentication and authorisation, injection attacks, and data leakage. As a result, securing modern software applications now requires vulnerabilities in RESTful APIs to be found and mitigated.

The goal is to provide a thorough overview of RESTful API security testing, emphasising identifying and mitigating common vulnerabilities (Ehsan et al., 2022). The main objective is to list the many vulnerabilities that RESTful APIs can experience and to analyse the tools and methods that can be used to find and fix those issues. This article will also look at integrating security testing into the software development lifecycle and the recommended practices for protecting RESTful APIs.

There is a growing need to maintain the security of RESTful APIs as they become more widely used in contemporary software development. Data breaches, system outages, and reputational harm can result from failing to identify and fix RESTful API vulnerabilities, which can have serious repercussions. For any organisation that uses RESTful APIs, it is crucial to comprehend the vulnerabilities they are subject to and to establish efficient security testing techniques (Sean B. Cleveland et al., 2020).

A. Motivation

Checking API security has become a crucial part of developing applications. To find and fix these flaws and guarantee compliance with applicable security standards and best practices, security testing for RESTful APIs is required. The difficulties of security testing for RESTful APIs, the many kinds of security testing, and typical flaws that can be fixed through efficient security testing are all covered in this article.

B. Contribution

The article explains the difficulties in performing security testing on RESTful APIs. starting by briefly explaining RESTful APIs and their importance in the current software development environment. And discuss the security issues that come up while testing RESTful APIs, like input validation, authentication, and authorization.From the security testing and mitigation of RESTful APIs, it has been identified that vulnerabilities that affect security configuration include insufficient logging, faulty object-level authorisation, asset management, faulty function-level authorisation, and mass assignment. It concludes by summarising the findings and offering suggestions for maintaining the security of RESTful APIs using previous research studies.

C. Paper organization

This paper is organized as follows: after the introduction, section 2 Discuss RESTful API security testing aspects and approaches. Section 3 cover the role of RESTful API in modern software development. Section 4 discuss the importance of security testing for RESTful APIs. Section 5 review the Different types of security testing for RESTful APIs. Section 6 show the challenges of security testing for RESTful APIs. Section 7 discusses the common vulnerabilities in RESTful APIs and with approaches, models and tools.Finally section 8 present the conclusion and future directions.

II. RESTFUL API SECURITY TESTING

Developing web-based applications, especially web services, has made Representational State Transfer (REST) a prominent architectural approach. RESTful APIs are widely used to facilitate communication between different software applications(Costa et al., 2014). However, the increased use of RESTful APIs has also made them an attractive target for hackers. As a result, security testing has become a critical aspect of the development process to ensure that RESTful APIs are secure and safe from vulnerabilities(Yahya et al., 2014). This section provides an overview of RESTful API security testing, its importance, and the different types of security testing that can be used.

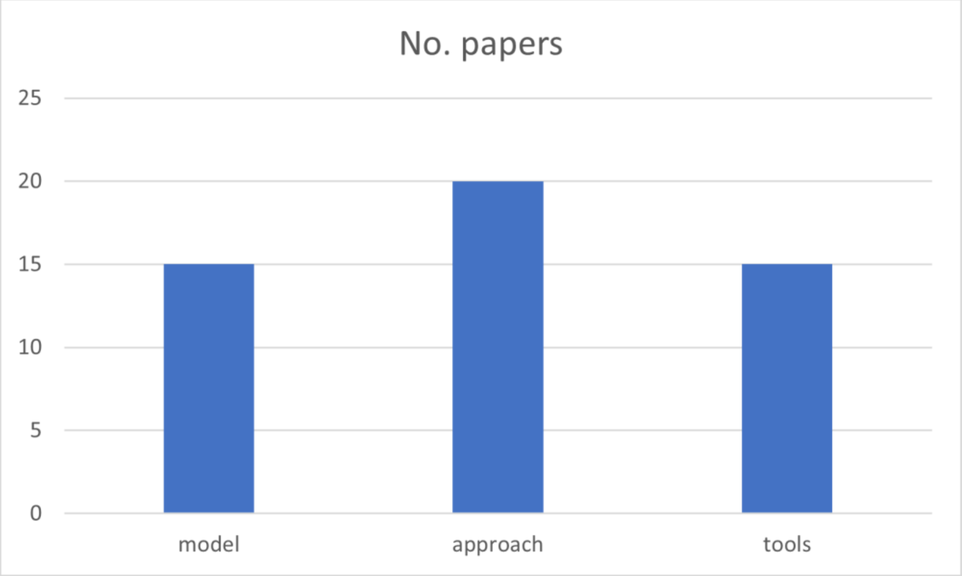

Fig. 1: Distribution of papers by model, approaches, and tools used for RESTful API security testing

Here are some key aspects to consider when testing the security of RESTful APIs:

A. *Authentication*

The authentication method used by the API, such as username/password login, token-based authentication, or OAuth 2.0 authentication, must be tested(Setiadi et al., 2019). Testers should attempt to get around the authentication process by accessing restricted sites without authorisation. Additionally, testers must look for account lockout mechanisms, brute-force assaults, and weak passwords.

B. *Authorisation*

This includes putting the API's authorisation system, which bases a user's access to resources on their role or privilege level, to the test. Access control list (ACL) vulnerabilities, privilege escalation vulnerabilities, and authorisation bypass vulnerabilities should all be looked for by testers(Modi et al., 2022).

C. *Input Validation*

This entails putting the input validation system of the API to the test, which verifies the accuracy of data submitted to the API. Common injection attacks that need to be tested for include SQL injection and cross-site scripting (XSS) (Hamza Ed-douibi et al., 2016). Additionally, testers should check for input-related security flaws and file upload vulnerabilities.

D. *Output Validation*

This involves putting the API's output validation system under test, which verifies the accuracy of the data the API returns. Cross-site request forgery (CSRF), Cross-site scripting (XSS), and other output-related vulnerabilities should all be tested for by testers (Compagna et al., 2018).

E. *Secure Communication*

This entails testing the API's communication channel to ensure it is safe and cannot be eavesdropped on (Garg & Dave, 2019). Testers should look for communication-related security flaws, SSL/TLS certificate validation problems, and other concerns.

F. *Error Handling*

This entails putting the API's error handling system under test, which establishes how the API handles exceptions and errors. Testers should check for sensitive information-revealing error messages and other error-related security problems (Garg & Dave, 2019).

G. *Session Management*

This involves testing the API's session management system, which controls how long user sessions last and handles them. (Ehsan et al., 2022). Testers should look for vulnerabilities that could lead to session hijacking, session fixation, and other security problems.

H. *Third-Party Integrations*

This entails putting to the test the API's integration with outside services, which poses security issues. Testers should check for security risks in third-party APIs and services, such as data leakage, access control flaws, and other security hazards(Modi et al., 2022).

I. *Rate Limiting*

It entails testing the API's rate-limiting mechanism, which establishes how many requests can be sent to the API in a specific amount of time, following (Malki et al., 2022). Rate-limiting bypass vulnerabilities and other rate limitation-related security problems should be tested for.

J. *Logging and Monitoring*

Testing the API's logging and monitoring systems, which keep track of all API activity and notify administrators of security events and abnormalities, is required. Testers should check for security concerns connected to logging and monitoring, such as log manipulation, log injection, and other hazards (Lee et al., 2014).

Table 1: RESTful API Security Testing

Aspect of Security	Testing Approach
Authentication	Test for weak authentication mechanisms, such as weak passwords or lack of multi-factor authentication.
Authorisation	Test for improper access controls, such as privilege escalation attacks or inadequate role-based access controls.
Input Validation	Test for proper validation of user inputs to prevent injection attacks, such as SQL injection or cross-site scripting (XSS).
Error Handling	Test for proper error handling, such as ensuring error messages do not reveal sensitive information or cause application crashes.
Session Management	Tests for proper session management, such as preventing session fixation attacks or session hijacking attacks.
API Rate Limiting	Test for proper API rate limiting to prevent denial of service (DoS) attacks or brute force attacks.
Integration Testing	Test for security vulnerabilities in third-party APIs or services that the API interacts with.

According to a study, data from online apps can leak even when encryption is used (Chen et al., 2010). This is done through routes known as "side channels." It was found by Serme et al. (2012) that the security of RESTful services is based either on transit layer security or ad hoc security techniques, both of which have security weaknesses. REST APIs can be examined for security issues using a collection of automatic security evaluations; it has been found (Ovidiu Baniaș et al., 2021). The risk that an attacker could take advantage of a RESTful application programming interface weakness is alarmingly raised by these publications when taken as a whole. Although APIs can be exploited (Macy, 2018), the effects of a hacking attempt depend on the situation and the type of data being transferred.

III. RESTFUL API AND ITS ROLE IN MODERN SOFTWARE DEVELOPMENT

The use of HTTP requests to access and modify data in web-based applications is known as a RESTful API. RESTful APIs have become integral to modern software development due to their flexibility, scalability, and ability to facilitate communication between different software applications (Lablans et al., 2015). RESTful APIs give programmers the ability to create web-based apps that are simple to link with other software programs. RESTful APIs use standard HTTP methods such as GET, POST, PUT, and DELETE to access and manipulate data (Christensen, 2009). RESTful APIs have become popular due to their ease of use, low overhead, and ability to support different data formats. They have become an essential part of modern software development and are used in various domains, such as e-commerce, finance, social media, and healthcare (Carneiro et al., 2021).

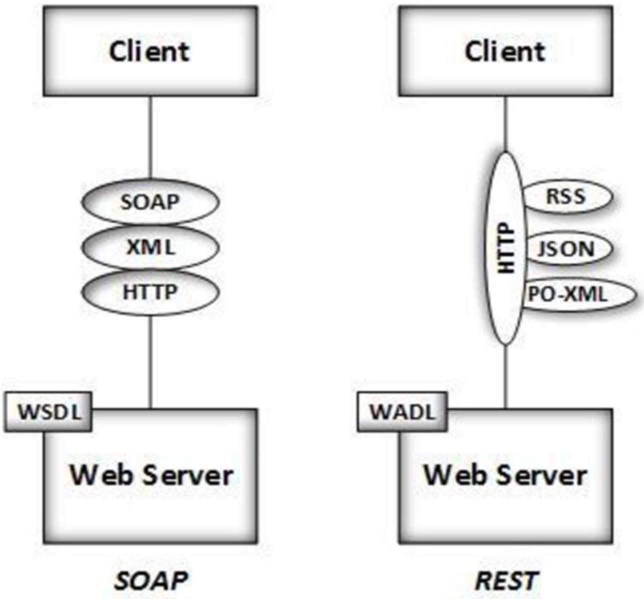

Fig. 2: SOAP vs REST API, Source (Malik & Kim, 2017)

It is concerned with the following:

A. *Separation of Concerns*

The separation of client-side and server-side concerns in RESTful APIs makes it simpler for developers to create, implement, and manage complex applications (Padmanaban et al., 2022). Because of this division, developers can modify one application component without affecting the others.

B. *Scalability*

Scalability refers to an API's ability to effectively manage a high volume of requests and responses(Le-Dang & Le-Ngoc, 2019). They are the best choice for use in large-scale enterprise applications because of their scalability.

C. *Flexibility*

Since RESTful APIs are adaptable, they can be used to send various data kinds, including text, photos, audio, and video. Thanks to this flexibility, developers can create various apps and services (Hästbacka et al., 2019).

D. *Statelessness*

RESTful APIs are stateless, which implies that every request includes all the data required to fulfil it(Guha, 2020). Performance is enhanced, and this statelessness facilitates the scalability of applications.

The fact that RESTful APIs work with any programming language that supports HTTP is one of their main advantages (Belkhir et al., 2019). The development of applications that can seamlessly connect is made more straightforward. RESTful APIs adhere to a standardised set of guidelines and restrictions, which helps to guarantee the API's effectiveness and scalability. Utilising RESTful APIs also allows developers to create simple applications for other developers to consume. As other programmers can build on top of the API to produce new applications and services, this can promote collaboration and creativity(Marilenaa et al., 2022).

According to the study's authors (Schreibmann & Braun, 2015), the development process would be enhanced by a model-driven approach in which an API is modelled using a new formal language created expressly for this application area at a higher level of abstraction. The source code for the business logic and database layers, as well as the API, can all be easily created from this model. The cost of documenting this procedure is nonexistent, and productivity increases along with a reduction in maintenance expenses and an increase in quality.

IV. IMPORTANCE OF SECURITY TESTING FOR RESTFUL APIS

A security breach in a RESTful API can result in unauthorised access to sensitive data, loss of user trust, financial loss, and legal consequences (Akhtar et al., 2021). Security testing can help detect and fix vulnerabilities before attackers exploit them. Security testing ensures that RESTful APIs are secure, reliable, and can be trusted by their users (Pourvahab & Ekbatanifard, 2019). Here are some reasons why security testing is essential for RESTful APIs:

A. *Protects Sensitive Sata*

RESTful APIs can handle sensitive data, including user passwords, financial data, and personal information. Security audits can find any API flaws that might expose this data to unauthorised users (Rivera et al., 2019).

B. *Mitigates the Risk of Attacks*

RESTful APIs are often used to communicate between different systems, making them vulnerable to attacks(Rafique et al., 2019). Security testing can identify any weaknesses in the API that malicious actors could exploit.

C. *Ensures Compliance*

Many industries like finance and healthcare have strict data privacy and security regulations. Security testing can ensure that RESTful APIs comply with these regulations(Tek Raj Chhetri et al., 2022).

D. *Maintains Brand Reputation*

If an API is compromised, it can damage the brand reputation of the company that owns it(Buitelaar et al., 2018). Security testing can identify and mitigate any vulnerabilities before they can be exploited by attackers.

RESTful APIs are exposed to various security threats such as injection attacks, authentication and authorisation issues, cross-site scripting (XSS), cross-site request forgery (CSRF), and sensitive data exposure (MacDonald, 2013). These vulnerabilities can lead to data breaches, loss of confidential information, and damage to the organisation's reputation. Therefore, performing security testing on RESTful APIs is crucial to identify and mitigate these vulnerabilities before attackers exploit them.

One of the primary reasons for the security testing of RESTful APIs is to protect sensitive data. RESTful APIs may handle sensitive data, such as personal, financial, or business-critical information. Without proper security measures in place, this data could be compromised, resulting in severe consequences for the organisation(Karlsson et al., 2020). Security testing helps identify vulnerabilities in the API that could be exploited to gain access to this data. Another important reason for the security testing of RESTful APIs is to prevent unauthorised access. Unauthorised users can access unsecured APIs, potentially leading to data theft or manipulation. Security testing helps identify and remediate such vulnerabilities by checking access controls, authentication mechanisms, and authorisation policies (Kornienko et al., 2021).

V. DIFFERENT TYPES OF SECURITY TESTING FOR RESTFUL APIS

Several types of security testing can be used to test the security of RESTful APIs.

Table 2: Types of security testing

Type of testing	Title	Reference and year
Authentication and authorisation testing	Security evaluation of the OAuth 2.0 framework	2015
Input validation testing	Deep Learning-Based Prediction of Test Input Validity for RESTful APIs	2021
Parameter tempering testing	Classification of Web-Service-Based Attacks and Mitigation Techniques	2018
Session management testing	Static analysis for web service security - Tools & techniques for a secure development life cycle	2015
Penetration testing	Checking Security Properties of Cloud Service REST APIs	2020
Vulnerability scanning	Automation of active reconnaissance phase: an automated API-based port and vulnerability scanner	2021
Fuzz testing	REST API Fuzzing by Coverage Level Guided Blackbox Testing	2021

A. *Black-box testing*

RESTful APIs frequently undergo black-box testing, a sort of security testing (Martin-Lopez et al., 2020). In a black-box test, the tester is unaware of how the system being evaluated operates from the inside. According to Alberto Martin-Lopez, black-box testing's objective is to find security flaws and vulnerabilities that a potential attacker may exploit. A variety of techniques can be used during black-box testing. These consist of the following:

➢ *Authentication and Authorisation Testing:*

Authentication and authorisation are essential security features that prevent unauthorised access to RESTful APIs (Sánchez et al., 2017). The authors of the study, (Bhat &Kansal, proposed that the open authorisation (OAuth) 2.0 industry-standard protocol for authorisation enables users to grant a third-party website or application access to the user's protected resources without the user having to reveal their long-term credentials or even their identity.

As opposed to this, the researchers (Paoli&Zavattaro, 2012) showed how a single, centralised security service with a lightweight application programming interface might manage authentication and authorisation for dependable RESTful services. A person must trade their information for a token to access limited resources. The services may check with the security provider to confirm the validity of a user's code and any rights that have been granted to them. The system enables fine-grained control over which resources a specific user has access to using the role-based access control (RBAC) paradigm.

➢ *Input Validation Testing:*

According to Rodriguez et al. (2020), input validation testing ensures that data submitted to RESTful APIs is validated to prevent malicious input, such as SQL injection or cross-site scripting attacks. Input Validation is a semi-automated device created to improve upon the current state of insufficient and inappropriate input validation claims study (Miller et al., 2008). Although many of the difficulties on the web are still relatively simple, developers do not seem to be aware of or able to address those successfully(Danezis, 2012).

➢ *Parameter Tampering Testing:*

Testing for parameter tampering involves changing input parameters to see if getting unauthorised access or tampering with data is possible (Musa et al., n.d.). Parameter tampering testing, in the authors' opinion (Atashzar et al., 2011), can aid in locating weaknesses such as insufficient parameter encryption or weak parameter validation.

➢ *Session Management Testing:*

The RESTful API's secure administration of user sessions is ensured by session management testing (Chaleshtari et al., 2023). Session fixation, session hijacking, and short session timeout are examples of vulnerabilities that can be found via session management testing, as shown by the studies (DEWI, 2022).

➢ *Boundary Testing:*

This entails evaluating how the API responds to inputs that are outside the acceptable range—for instance, testing the API's ability to handle extremely big or minimal inputs (Zhiwei & Zhongliang, 2020). Integer overflow vulnerabilities or other forms of input mistakes can be found using this technique.

➢ *Penetration Testing:*

Penetration testing is a technique for evaluating a system's security by simulating an adversarial assault (Sandhya et al., 2017). By spotting flaws and vulnerabilities that an attacker could take advantage of, penetration testing can be performed to assess the security of RESTful APIs. Penetration testing is possible using either human or automated techniques (Patel, 2019).

➢ *Vulnerability Scanning:*

The authors claim that it entails employing automated tools to scan the program for known vulnerabilities (Shah & Mehtre, 2015). The tools generate a report for the tester after locating vulnerabilities in the application.

- *Protocol Testing:*

Since HTTP is the foundation of RESTful APIs, it is critical to test how the API responds to various HTTP methods (GET, POST, PUT, DELETE, etc.) and HTTP status codes. This can assist in locating vulnerabilities brought on by incorrect HTTP requests and response handling (Xiong et al., 2021).

- *Fuzz Testing:*

Fuzz testing, also known as "fuzzing," is a technique for testing software by providing unexpected or invalid input to the system to see how it responds (IEEE Conference Publication, 2023). A novel utility called SAGE (Scalable, Automated, Guided Execution) uses x86 instruction-level tracing and emulation to perform whitebox fuzzing of random file-reading Windows apps, as found by (Atlidakis et al., 2019). This indicates that RESTful APIs can be tested with fuzz to discover bugs. It has been discovered by (Fertig & Braun, 2015) that test cases for RESTful APIs can be generated automatically by a software creator. This indicates that taint testing can be utilised when evaluating RESTful APIs. However, as discovered by (Klees et al., 2018), experimental reviews of fuzz testing methods can be flawed, resulting in inaccurate or misleading verdicts.

There are several methods for security testing RESTful APIs, including dynamic testing, static testing, and manual testing. To find any security flaws, dynamic testing entails executing the APIs and examining the results (Atlidakis et al., 2019). This method entails making different kinds of queries to the APIs and checking the replies to make that they adhere to the necessary security criteria. For instance, a dynamic security testing framework for RESTful APIs was presented by Corradini et al. in 2022. The framework comprises several processes, such as creating a testing environment, creating test cases, running tests, and producing reports. The authors tested their framework on several RESTful APIs and saw encouraging results.

On the other hand, static testing entails studying the API's source code without actually running it. This method is frequently used to find vulnerabilities that dynamic testing could miss. Code review is a typical static testing technique in which a group of developers or security specialists examine the code to find any security flaws (Khayer et al., 2020). The study's authors Talukder et al. (2019) developed a static analysis tool that examines the source code of RESTful APIs and finds security vulnerabilities using a combination of machine learning and natural language processing approaches. The authors had success using a real-world API to test their solution.

In manual testing, human testers carefully examine the APIs to find any security flaws. This method is frequently used in conjunction with dynamic testing to find vulnerabilities that might not be found otherwise. As an illustration, (Martin-Lopez et al., 2020) suggested a manual testing strategy for RESTful APIs that entails developing test cases based on security requirements and manually executing them. The authors successfully tested their strategy on a real-world API and got positive results.

Despite these security testing methods' success, testing RESTful APIs still presents several difficulties. The complexity of RESTful APIs, which can involve several levels and dependencies, is one of the significant difficulties. It is challenging to guarantee that all API components are appropriately tested due to their complexity (Laranjeiro et al., 2021). Additionally, RESTful APIs frequently interact with other APIs and services, increasing the complexity of testing, according to the research (Ehsan et al., 2022). The dynamic nature of APIs, which can lead to endpoints and behaviours that are continually changing, presents another difficulty. Therefore, it is crucial to maintain the testing procedure to guarantee that all potential vulnerabilities are found and fixed.

VI. THE CHALLENGES OF SECURITY TESTING FOR RESTFUL APIS

RESTful APIs have become a popular means of communication between applications and systems. They provide internet-based exposure to web services, allowing for system interoperability. However, the problem of protecting the security of the APIs comes with this ease of communication. RESTful APIs must be subject to security testing to ensure the confidentiality, integrity, and availability of data as well as the overall security of the system. Vulnerabilities must be found and mitigated.

Security testing for RESTful APIs is not without its challenges. Some of the challenges include the following:

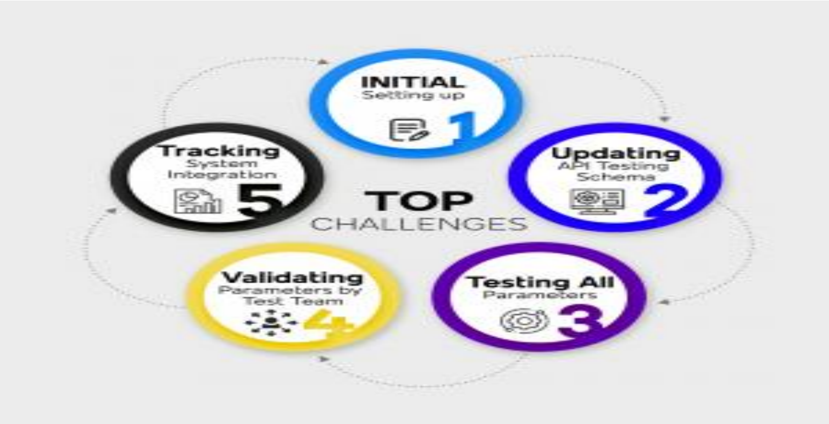

Fig. 3: The Challenges of API Testing

A. *API Complexity:*

According to the author D V Kornienko (2021), RESTful APIs can be complex, making it difficult to identify potential vulnerabilities. APIs can constantly be evolving, making it challenging to keep up with changes.

B. *Specialised Knowledge:*

As observed from the study Peng et al. (2022), Security testing for RESTful APIs requires specialised knowledge and skills. Developers and security testers must be familiar with the REST architectural style, HTTP protocols, and API security best practices.

C. *Secure Transmission:*

RESTful APIs transmit data over the internet, which means that data can be intercepted and viewed by unauthorised parties. Testing for secure transmission involves ensuring that data is encrypted in transit using HTTPS and that the encryption is implemented correctly (A framework for measuring organisational information security vulnerability, 2023).

D. *Rate Limiting:*

RESTful APIs can be vulnerable to denial-of-service attacks where an attacker overwhelms the system by sending many requests. Testing for rate limiting involves verifying that the API can handle high volumes of requests and that rate limits are appropriately enforced (Barabanov et al., 2022).

E. *API Abuse:*

RESTful APIs can be abused by attackers who use the API to scrape data or perform actions that are not intended. Testing for API abuse involves identifying and mitigating such attacks (Christensen, 2009).

F. *Tool Limitations:*

As observed from the studies (Nuno Realista et al ., 2022), Automated tools such as vulnerability scanners may not be able to identify all vulnerabilities in RESTful APIs (Lamothe et al., 2021) argues that automated tools may also generate false positives or false negatives, making it challenging to determine the actual state of the APIs.

G. *Lack of Standardisation:*

There is a lack of standardisation in RESTful API development, making it challenging to create a standardised testing methodology (Gill et al., 2022).

H. *Lack of Expertise:*

The study's authors revealed that (Aljedaani & Babar, 2021)there is a shortage of experts with the required knowledge and skills to perform security testing on RESTful APIs.

One of the main challenges in security testing for RESTful APIs is the complexity of the interactions between different system components, as observed from the studies of (Karlsson et al., 2020). Since RESTful APIs rely on HTTP and are stateless, they require complex interactions between different components of the system to function correctly. This complexity can make it challenging to identify vulnerabilities and test the security of the system (Ozdemir, 2020).

Another challenge observed from the studies of (Keping Yu et al., 2021) is the use of third-party libraries and components. RESTful APIs often rely on third-party libraries and components to perform various tasks, such as authentication, encryption, and validation. However, these components may have their vulnerabilities or be misconfigured, leading to vulnerabilities in the overall system. Additionally, these components may be updated or changed without notice, leading to unexpected vulnerabilities (Qingyang Zeng et al., 2023).

Furthermore, as observed from the studies of (Mai et al., 2020), RESTful APIs are often used in distributed systems, which can make it challenging to test the security of the entire system. Since RESTful APIs are stateless, they do not maintain information about previous requests or responses, making it challenging to test the system's overall security. Additionally, distributed systems often have multiple points of entry, making it challenging to identify all potential vulnerabilities (Setiadi et al., 2019).

Finally, as highlighted from the studies of (Krishnan et al., 2023), the increasing use of cloud computing and virtualisation technologies can introduce additional security challenges for RESTful APIs. Cloud providers may have their security policies and procedures that must be followed, and virtualisation technologies may introduce additional abstraction layers that can make identifying vulnerabilities challenging (Almutairy & Al-Shqeerat, 2019).

VII. COMMON VULNERABILITIES IN RESTFUL APIS

RESTful APIs have become a popular choice for developers due to their simplicity, flexibility, and ability to integrate with other systems. However, the authors of the study (A framework for measuring organisational information security vulnerability, 2023) show that this ease of use also creates various security challenges. RESTful APIs are vulnerable to various attacks, which can have severe consequences, such as data breaches, financial losses, and reputational damage. This chapter focuses on the most common vulnerabilities found in RESTful APIs and their impact on the security of the system.

Table 3: Research on vulnerabilities

SR. NO.	Vulnerabilities	Reference	Approach	Model	Tools
1	Broken Authentication	2021	No	Yes	Yes
2	Broken authentication and session management	2018	Yes	No	No
3	Broken Authentication	2021	Yes	No	No
4	Excessive Data Exposure	2023	Yes	No	Yes
5		2023	Yes	Yes	No
6	Lack of Resources & Rate Limiting	2019	Yes	No	No
7	Broken Function Level Authorisation	2021; 2022	No	Yes	Yes
8	Mass assignment	2023; 2020	Yes	No	Yes
9	Security misconfiguration	2015	Yes	No	yes
10	Improper asset management	2023	Yes	Yes	No
11	Insufficient Logging & Monitoring	2019	Yes	No	No
12	Injection		Yes	No	No

A. Broken Object Level Authorisation

The broken object-level authorisation is a vulnerability that occurs when an API does not restrict access to objects based on the user's privileges, and this means that a user can access and modify any object within the API, even if they do not have the required permissions (Haddad & Malki, 2022). Attackers can exploit this vulnerability to gain access to sensitive data and perform unauthorised actions, as observed in the study of (Taya et al., 2022).

The causes of this vulnerability include the lack of proper access control mechanisms and insufficient testing of access controls. Attackers can exploit this vulnerability by modifying requests to access unauthorised objects (Votipka et al., 2020). An attacker could manipulate a request to access another user's data or escalate their privileges to perform actions beyond their permissions.

A real-world example of this vulnerability is the Facebook Cambridge Analytica scandal, where a third-party app exploited the vulnerability in Facebook's API to access and harvest user data without consent. This resulted in a massive data breach and significantly damaged Facebook's reputation (Jeune, 2021).

B. Broken Authentication

As observed from the study Bach-Nutman (2020), Broken authentication is a vulnerability that occurs when an API does not properly authenticate users, allowing attackers to access the system without proper credentials. This vulnerability can be exploited through various techniques, such as brute force attacks, session hijacking, and credential stuffing.

The causes of this vulnerability include the use of weak or easily guessable passwords, the lack of multi-factor authentication, and the failure to implement secure session management; the study (Kabir & Elmedany, 2022) shows that attackers can exploit this vulnerability by stealing user credentials and using them to access the system.

One real-world example of this vulnerability is the Equifax data breach, where attackers exploited a vulnerability in Equifax's API to gain access to sensitive customer data. This breach compromised the personal information of over 143 million individuals and resulted in a significant loss of trust and financial damage for Equifax (Dennis et al., 2020).

C. Excessive Data Exposure

The authors of the study Pan et al. (2023) showed that Excessive data exposure is a vulnerability that occurs when an API exposes more data than necessary, such as sensitive data or user credentials; attackers can exploit this vulnerability to gain access to sensitive data or perform unauthorised actions.

The causes of this vulnerability include the lack of proper data sanitisation and validation, the failure to implement proper access controls, and the use of insecure data storage; attackers can exploit this vulnerability by sending specially crafted requests to access sensitive data (Khan et al., 2021).

D. Lack of Resources & Rate Limiting

Lack of resources and rate limiting is a vulnerability that occurs when an API does not appropriately limit the number of requests that can be made, allowing attackers to overwhelm the system with requests and cause denial-of-service attacks (Sharieh & Ferworn, Securing APIs and Chaos Engineering, 2021).

The causes of this vulnerability include the failure to implement rate limiting, the use of weak or easily guessable API keys, and the lack of monitoring for unusual traffic patterns; attackers can exploit this vulnerability by sending a large number of requests to the API, causing the system to become overloaded and unresponsive (Azad et al., 2020).

One real-world example of this vulnerability is the Twitter API outage, where a group of attackers overloaded the API with requests, causing it to become unavailable for several hours (A, 2023).

E. Broken Function Level Authorisation:

As observed from the studies (Haddad & Malki, 2022), Broken function level authorisation is a vulnerability that occurs when an API does not restrict access to specific functions or operations based on user roles or permissions, the authors (Fredj et al., 2021) showed that this vulnerability could allow attackers to perform unauthorised actions on the system, such as deleting or modifying sensitive data, the vulnerability is typically caused by poor implementation of access control mechanisms, such as failing to check user permissions before allowing them to act.

Several studies have proposed diverse techniques to detect and mitigate broken function-level authorisation vulnerabilities in RESTful APIs. For example, a study by (Barabanov et al., 2022) proposed an access control testing approach that uses a combination of static and dynamic analysis techniques to identify vulnerabilities in APIs. The approach involves analysing the source code of the API to identify potential vulnerabilities and then using dynamic analysis techniques to test the API's behaviour under different scenarios.

F. Mass Assignment:

The author of the study D V Kornienko (2021)discussed that mass assignment is a vulnerability that occurs when an API allows users to modify multiple attributes of an object in a single request. Attackers can exploit this vulnerability to modify sensitive data or gain unauthorised access to the system. The vulnerability is typically caused by poor validation of user input or a lack of proper access control mechanisms (Sidra & Michael, 2023).

To mitigate mass assignment vulnerabilities, several researchers have proposed different techniques. For example, a study by Gantikow et al. (2020) proposed a rule-based approach to detect and prevent mass assignment vulnerabilities in RESTful APIs. The approach involves defining rules that specify which attributes of an object can be modified by different user roles or permissions. When a request is received, the system checks the user's permissions and applies the relevant rules to determine which attributes can be modified.

An attacker can exploit Mass Assignment vulnerability by sending specially crafted requests that include additional parameters or by modifying the values of existing parameters. As shown from the studies (Al-Jody, 2021), an attacker could modify a user's account information by sending a request that includes the "isAdmin" field set to "true". The attacker could gain administrative privileges if the API does not correctly validate this parameter.

One real-world example of a mass Assignment vulnerability was discovered in 2011 in the Ruby on Rails framework. This vulnerability allowed attackers to modify any database record by sending specially crafted requests. The vulnerability affected thousands of websites and applications and was considered one of the most severe vulnerabilities ever discovered in the framework (Park et al., 2021).

G. Security Misconfiguration:

Security misconfiguration is a vulnerability that occurs when an API is configured with insecure settings, such as default passwords or unnecessary features enabled. This vulnerability can allow attackers to gain unauthorised access to the system or perform other malicious actions (Aljabri, Aldossary, Al-Homeed, Alhetelah, & Althubian, 2022). The vulnerability is typically caused by poor configuration management practices, such as failing to disable unnecessary features or using default passwords (Loureiro, 2021).

As observed from the studies of Rahman et al. (2023), Security Misconfiguration occurs when the API allows unrestricted access to specific resources or functionality. This can happen when developers do not properly configure access controls or when they do not properly configure the API's authentication mechanisms. An attacker can exploit this vulnerability by accessing sensitive data or by performing actions on behalf of another user.

One real-world example of Security Misconfiguration occurred in 2017 when an unprotected Amazon Web Services (AWS) S3 bucket was discovered. The bucket contained sensitive data belonging to the US Army and was accessible to anyone who had the URL. This vulnerability was caused by the misconfiguration of the S3 bucket and highlighted the importance of proper configuration of cloud-based services (Jäger, 2021).

H. Injection:

As observed from the studies of Hasan & Rahman (2023), Injection vulnerabilities occur when an attacker can inject malicious code into an API, such as SQL or code injection. This vulnerability can allow attackers to execute arbitrary code on the system or access sensitive data. The vulnerability is typically caused by poor input validation or a lack of proper access control mechanisms.

To mitigate injection vulnerabilities, several researchers have proposed different techniques. For example, a study by Erik Trickel et al. (2022) proposed a technique that uses a combination of static and dynamic analysis to detect injection vulnerabilities in RESTful APIs. The approach involves analysing the source code of the API to identify potential injection points and then using dynamic analysis techniques to test the API's behaviour under different scenarios.

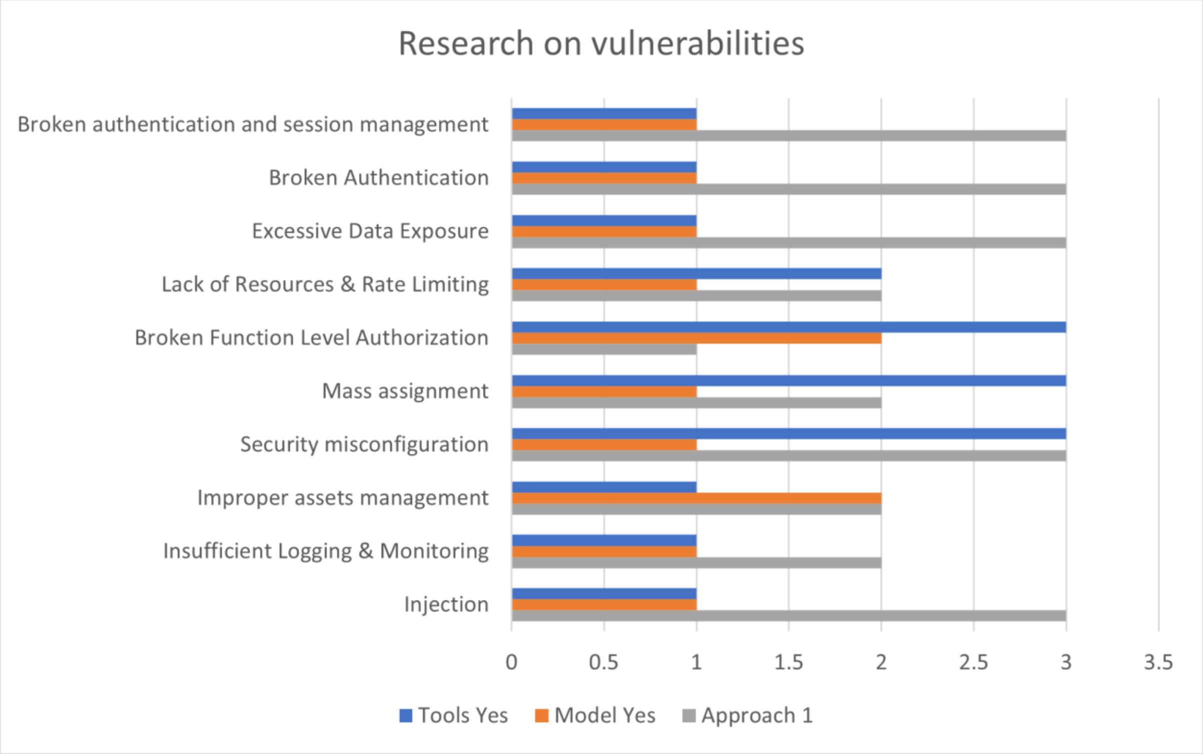

Fig. 4: Usage of tools, frameworks, and approaches for vulnerabilities testing

I. *Improper Assets Management:*

Improper assets management is a vulnerability that occurs when an API does not properly manage its assets, such as files or resources. This vulnerability can allow attackers to access or modify sensitive data or resources (Idris, Syarif, & Winarno, Web Application Security Education Platform Based on OWASP API Security Project, 2022). Poor access control mechanisms or improper asset management practices typically cause vulnerability.

The Capital One breach in 2019, where a hacker gained unauthorised access to the personal data of over 100 million customers. The vulnerability was caused by a misconfigured firewall, which allowed the hacker to exploit a broken authentication and session management vulnerability (Khan et al., 2022). Attackers can exploit insufficient authorisation to access sensitive data, perform unauthorised actions, or manipulate the behaviour of the system. This can lead to severe consequences, such as data breaches, financial losses, or reputational damage. The consequences of not detecting and mitigating vulnerabilities in RESTful APIs can be severe. They can result in the loss of sensitive data, financial losses, and damage to the reputation of the organisation. For example, the Equifax breach resulted in a settlement of $700 million, and the Capital One breach resulted in a settlement of $80 million. In addition to financial losses, organisations may also face legal penalties and damage to their reputation (Okafor, 2021).

Several researchers have proposed different techniques to detect and mitigate insufficient authorisation vulnerabilities in RESTful APIs. (Padma & Srinivasan, 2023) proposed a novel method that analyses access control policies specified in OAuth 2.0 and OpenID Connect to detect potential authorisation issues. Their approach involves analysing the relationships between different entities in the authorisation process, such as resource servers, clients, and authorisation servers, to identify potential authorisation conflicts or inconsistencies. The authors also propose an automated tool that implements their approach and can be integrated into the API testing workflow.

Using machine learning techniques to automatically find and fix authorisation flaws in RESTful APIs is another strategy suggested by Sharieh and Ferworn (Securing APIs and Chaos Engineering, 2021). Their strategy is looking through API request logs to find patterns of unusual behaviour that might point to authorisation problems. To find these patterns and send out notifications when possible vulnerabilities are found, the authors combine supervised and unsupervised learning approaches. Additionally, they suggest a mitigation technique that can be applied to deny requests coming from malicious users or IP addresses automatically.

In addition to insufficient authorisation vulnerabilities, other common vulnerabilities in RESTful APIs include injection attacks, broken authentication and session management, and insecure data storage. Injection attacks, such as SQL injection and cross-site scripting (XSS), can be particularly damaging and are often used by attackers to gain access to sensitive data or take control of the system (Idris, Syarif, & Winarno, Development of Vulnerable Web Application Based on OWASP API Security Risks, 2021), whereas observed from the studies of (Gill et al., 2022) Broken authentication and session management vulnerabilities, on the other hand, can allow unauthorised users to access protected resources or perform actions on behalf of legitimate users. Insecure data storage vulnerabilities can result in sensitive data being exposed or stolen, which can have severe consequences for both users and the organisation.

Researchers have suggested several strategies to identify and address these vulnerabilities, including static and dynamic analysis techniques, vulnerability scanning tools, and secure coding practices. For instance, Cao et al.(2020) proposed a dynamic analysis method that takes advantage of symbolic execution to create test cases for RESTful APIs and find injection vulnerabilities. Their strategy entails modelling the API as a finite state machine and producing constraints that accurately represent the API's behaviour. The authors also suggest a mitigating method that makes use of runtime monitors to find and deny requests that go against the restrictions.

The use of vulnerability scanning tools to automatically identify and prioritise vulnerabilities in RESTful APIs is another strategy suggested by (Jorge Reyes et al., 2022). To find potential vulnerabilities, they use analysis of the API documentation and source code, grading them according to impact and severity. To increase the precision of the detection process, the authors also suggest a feedback mechanism that enables developers to offer more information or context about particular vulnerabilities.

VIII. CONCLUSION AND FUTURE DIRECTIONS

In conclusion, common RESTful API vulnerabilities and their potential effects on system security have been found through security testing and mitigation of RESTful APIs. These flaws include faulty authentication, faulty object-level authorisation, excessive data exposure, insufficient resources and rate limiting, faulty function-level authorisation, mass assignment, faulty security configuration, injection, faulty asset management, and insufficient logging and monitoring. Along with instances of actual attacks that make use of these vulnerabilities, each vulnerability's causes and techniques of exploitation have also been covered.

This article's discussion on RESTful API security testing and mitigation also highlights the need for appropriate testing and mitigation procedures to prevent security breaches. Finally, the article has outlined future directions for further study in this area, including applying machine learning algorithms for vulnerability detection and creating automated security testing tools. There have also been discussions of open research issues like the lack of standardisation in RESTful API security testing and the complexity of finding complicated vulnerabilities.

This article's conclusion emphasises the significance of thorough testing and mitigation strategies for securing RESTful APIs. It highlights prospective topics for more research and offers insightful information on the state of this field's research at the moment.

REFERENCES

[1.] (2023, March 17). Retrieved from A framework for measuring organisational information security vulnerability: http://dspace.library.uvic.ca/handle/1828/11300

[2.] A, H. (2023, March 17). *Twitter suffers large outages on the web and mobile*. Retrieved from https://www.theguardian.com/technology/2016/jan/19/twitter-down-over-web-and-mobile

[3.] Akhtar, D. N., Kerim, B., Perwej, D. Y., Tiwari, A., & Praveen, D. S. (2021). A Comprehensive Overview of Privacy and Data Security for Cloud Storage. *International Journal of Scientific Research in Science Engineering and Technology*.

[4.] Aljabri, M., Aldossary, M., Al-Homeed, N., Alhetelah, B., & Althubian, M. (2022). Testing and Exploiting Tools to Improve OWASP Top Ten Security Vulnerabilities Detection. *14th International Conference on Computational Intelligence and Communication Networks (CICN)*, 797-803.

[5.] Aljedaani, B., & Babar, M. A. (2021). Challenges With Developing Secure Mobile Health Applications: Systematic Review. *JMIR mHealth and uHealth*, 15654.

[6.] Al-Jody, T. (2021). Barricade: A Novel High-Performance Computing User and Security Management System Augmented with Machine Learning Technology.

[7.] Almutairy, N. M., & Al-Shqeerat, K. H. (2019). A Survey on Security Challenges of Virtualization Technology in Cloud Computing. *International Journal of Computer Science & Information Technology (IJCSIT).*

[8.] Atashzar, H., Torkaman, A., Bahrololum, M., & Tadayon, M. H. (2011). A survey on web application vulnerabilities and countermeasures. *6th International Conference on Computer Sciences and Convergence Information Technology (ICCIT)*, 647-652.

[9.] Atlidakis, V., Godefroid, P., & Polishchuk, M. (2019). RESTler: Stateful REST API Fuzzing. *IEEE/ACM 41st International Conference on Software Engineering (ICSE)*, pp. 748–758.

[10.] Azad, B. A., Starov, O., Laperdrix, P., & Nikiforakis, N. (2020). Web Runner 2049: Evaluating Third-Party Anti-bot Services. *Detection of Intrusions and Malware, and Vulnerability Assessment: 17th International Conference, DIMVA 2020, Lisbon, Portugal, June 24–26, 2020, Proceedings 17*, 135-159.

[11.] Bach-Nutman, M. (2020). Understanding The Top 10 OWASP Vulnerabilities. *arXiv preprint arXiv*, 2012.09960.

[12.] Barabanov, A., Dergunov, D., Makrushin, D., & Teplov, A. (2022). Automatic detection of access control vulnerabilities via API specification processing. *arXiv preprint arXiv*, 2201.10833.

[13.] Belkhir, A., Abdellatif, M., Tighilt, R., Moha, N., & Guéhén, Y.-G. (2019). An Observational Study on the State of REST API Uses in Android Mobile Applications. *IEEE/ACM 6th International*

Conference on Mobile Software Engineering and Systems (MOBILESoft), 66-75.

[14.] Bhat, P. K., & Kansal, R. (n.d.). Development of RESTful Web API using Token-based OAuth 2.0 Authorisation. *International Journal of Engineering Research*.

[15.] Buitelaar, P., Wood, I. D., Negi, S., Arcan, M., & McCrae, J. P. (2018). MixedEmotions: An Open-Source Toolbox for Multimodal Emotion Analysis. *IEEE Transactions on Multimedia*, 2454-2465.

[16.] Cao, C., Guan, L., Ming, J., & Liu, P. (2020). Device-agnostic Firmware Execution is Possible: A Concolic Execution Approach for Peripheral Emulation. *Annual Computer Security Applications Conference*, 746-759.

[17.] Carlos Rodríguez et al. (2016). REST APIs: A Large-Scale Analysis of Compliance with Principles and Best Practices. *Web Engineering: 16th International Conference, ICWE 2016, Lugano, Switzerland, June 6-9, 2016. Proceedings 16*, 21-39.

[18.] Carneiro, G., Toniolo, A., Ncenta, M. A., & Quigley, A. J. (2021). Text vs Graphs in Argument Analysis. *IEEE Symposium on visual languages and human-centric computing (VL/HCC)*, 1-9.

[19.] Chaleshtari, N. B., Pastore, F., Goknil, A., & Briand, L. C. (2023). Metamorphic Testing for Web System Security. *IEEE Transactions on Software Engineering.*

[20.] Chen, S., Wang, R., Wang, X., & Zhang, K. (2010). Side-Channel Leaks in Web Applications: A Reality Today, a Challenge Tomorrow. *IEEE Symposium on Security and Privacy*, 191-206.

[21.] Christensen, J. H. (2009). Using RESTful web services and cloud computing to create next-generation mobile applications. *Proceedings of the 24th ACM SIGPLAN conference companion on Object-oriented programming systems languages and applications*, 627-634.

[22.] Compagna, L., Guilleminot, P., & Brucker, A. D. (2018). Business Process Compliance via Security Validation as a Service. *IEEE sixth international conference on software testing, Verification, and Validation*, 455-462.

[23.] Corradini, D., Zampieri, A., Pasqua, M., Viglianisi, E., Dallago, M., & Ceccato, M. (2022). Automated black-box testing of nominal and error scenarios in RESTful APIs. *Software Testing, Verification and Reliability*, 1808.

[24.] Costa, B., Pires, P. F., Delicato, F. C., & Merson, P. (2014). Evaluating a Representational State Transfer (REST) Architecture: What is the Impact of REST in My Architecture? *IEEE/IFIP Conference on Software Architecture*, 105-114.

[25.] D V Kornienko, S. V. (2021). The Single Page Application architecture when developing secure Web services. *Journal of Physics: Conference Series*, 012065.

[26.] Danezis, G. (2012). Financial Cryptography and Data Security. *Springer Berlin Heidelberg.*

[27.] Dennis, K., Alibayev, M., & Ligatti, J. (2020). Cybersecurity Vulnerabilities in Mobile Fare Payment Applications: A Case Study. *Transportation Research Record*, pp. 616–624.

[28.] DEWI, B. T. (2022). Web Security Compliance To Owasp And Sans Standard.

[29.] Ehsan, A., Abuhaliqa, M. A., Catal, C., & Mishra, D. (2022). RESTful API Testing Methodologies: Rationale, Challenges, and Solution Directions. *Applied Sciences*, 12(9), 4369.

[30.] Fertig, T., & Braun, P. (2015). Model-driven Testing of RESTful APIs. *Proceedings of the 24th International Conference on World Wide Web*, 1497-1502.

[31.] Fredj, O. B., Cheikhrouhou, O., Krichen, M., Hamam, H., & Derhab, A. (2021). An OWASP Top Ten Driven Survey on Web Application Protection Methods. *Risks and Security of Internet and Systems: 15th International Conference, CRiSIS 2020, Paris, France, November 4–6, 2020, Revised Selected Papers 15*, 235-252.

[32.] Gantikow, H., Reich, C., Knahl, M., & Clarke, N. (2020). Rule-Based Security Monitoring of Containerized Environments. *Cloud Computing and Services Science: 9th International Conference, CLOSER 2019, Heraklion, Crete, Greece*, 66-86.

[33.] Garg, H., & Dave, M. (2019). Securing IoT Devices and SecurelyConnecting the Dots Using REST API and Middleware. *4th International Conference on Internet of Things: Smart Innovation and Usages*, pp. 1–6.

[34.] Gill, S. S., Sharma, B., Bansal, V., Sharma, K., & Goyal, A. (2022). Vulnerability Exploiter for Web Applications. *2nd International Conference on Innovative Practices in Technology and Management (ICIPTM)*, pp. 292–299.

[35.] Guha, S. (2020). A Comparative Study Between Graph-QL & Restful Services in API Management of Stateless Architectures. *International Journal on Web Service Computing (IJWSC)*, 11(2).

[36.] Haddad, R., & Malki, R. E. (2022). OpenAPI Specification Extended Security Scheme: A method to reduce the prevalence of Broken Object Level Authorization. *arXiv preprint arXiv*, 2212.06606.

[37.] Hamza Ed-douibi et al. (2016). EMF-REST: generation of RESTful APIs from models. *Proceedings of the 31st Annual ACM Symposium on Applied Computing*, 1446-1453.

[38.] Hasan, M. A., & Rahman, M. M. (2023). Minimise Web Applications vulnerabilities through the early Detection of CRLF Injection. *arXiv preprint arXiv*, 2303.02567.

[39.] Hästbacka, D., Halme, J., Larrañaga, M., More, R., & Mesiä, H. (2019). Dynamic and Flexible Data Acquisition and Data Analytics System Software Architecture. *IEEE SENSORS*, 1-4.

[40.] Idris, M., Syarif, I., & Winarno, I. (2021). Development of Vulnerable Web Application Based on OWASP API Security Risks. *International Electronics Symposium (IES)*, 190-194.

[41.] Idris, M., Syarif, I., & Winarno, I. (2022). Web Application, Security Education Platform, Based on OWASP API Security Project. *EMITTER*

International Journal of Engineering Technology, 246-261.

[42.] *IEEE Conference Publication*. (2023, March 17). Retrieved from Web application fuzz testing: https://ieeexplore.ieee.org/abstract/document/8285893?casa_token=x2Oe_U-Y0DkAAAAA:Pd1zA1IRCO5LKa1iugF_V4iCwPrTbnoCnoDhkD0WoFm5TMKHtLjKsiHD9SSxHsFkCzsKeKcjYhIbvBI

[43.] Jäger, A. (2021). Finding and evaluating the effects of improper access control in the Cloud.

[44.] Jeune, M. L. (2021). Facebook and the Cambridge Analytica Scandal: Privacy and Personal Data Protection in Canada. *Curve. Carleton*.

[45.] Jorge Reyes et al. (2022). An Environment-Specific Prioritisation Model for Information-Security Vulnerabilities Based on Risk Factor Analysis. *Electronics*, 1334.

[46.] Kabir, M. A., & Elmedany, W. (2022). An Overview of the Present and Future of User Authentication. *4th IEEE Middle East and North Africa COMMunications Conference (MENACOMM)*, 10-17.

[47.] Karlsson, S., Čaušević, A., & Sundmark, D. (2020). QuickREST: Property-based Test Generation of OpenAPI-Described RESTful APIs. *IEEE 13th International Conference on Software Testing, Validation and Verification (ICST)*, 131-141.

[48.] Keping Yu et al. (2021). Blockchain-Enhanced Data Sharing With Traceable and Direct Revocation in IIoT. *IEEE Transactions on industrial informatics*, 7669-7678.

[49.] Khan, F., Kim, J. H., Mathiassen, L., & Moore, R. (2021). DATA BREACH MANAGEMENT: AN INTEGRATED RISK MODEL. *Information & Management*, 103392.

[50.] Khan, S., Kabanov, I., Hua, Y., & Madnick, S. (2022). A Systematic Analysis of the Capital One Data Breach: Critical Lessons Learned. *ACM Transactions on Privacy and Security*, 1-29.

[51.] Khayer, A. A., Almomani, I., & Elkawlak, K. (2020). ASAF: Android Static Analysis Framework. *First International Conference of Smart Systems and Emerging Technologies*, 197-202.

[52.] Klees, G., Ruef, A., Cooper, B., Wei, S., & Hicks, M. (2018). Evaluating Fuzz Testing. *Proceedings of the 2018 ACM SIGSAC conference on computer and communications security*, 2123-2138.

[53.] Kornienko, D. V., Mishina, S. V., Shcherbatykh, S. V., & Melnikov, M. O. (2021). Principles of securing RESTful API web services developed with Python frameworks. *Journal of Physics: Conference Series*, 032016.

[54.] Krishnan, P., Jain, K., Aldweesh, A., Prabu, P., & Buyya, R. (2023). OpenStackDP: a scalable network security framework for SDN-based OpenStack cloud infrastructure. *Journal of Cloud Computing*, p. 26.

[55.] Lablans, M., Borg, A., & Ückert, F. (2015). A RESTful interface to pseudonymisation services in modern web applications. *BMC medical informatics and decision making*, 1-10.

[56.] Lamothe, M., Li, H., & Shang, W. (2021). Assisting Example-Based API Misuse Detection via Complementary Artificial Examples. *IEEE Transactions on Software Engineering*, pp. 3410–3422.

[57.] Laranjeiro, N., Agnelo, J., & Bernardino, J. (2021). A Black Box Tool for Robustness Testing of REST Services. *IEEE Access*, 24738-24754.

[58.] Le-Dang, Q., & Le-Ngoc, T. (2019). Scalable Blockchain-based Architecture for Massive IoT Reconfiguration. *IEEE Canadian Conference of Electrical and computer engineering (CCECE)*, 1-4.

[59.] Lee, S., Jo, J.-Y., & Kim, Y. (2014). ENVIRONMENTAL SENSOR MONITORING WITH SECURE RESTFUL WEB SERVICE. *International Journal of Services Computing*, 30-42.

[60.] Loureiro, S. (2021). Security misconfigurations and how to prevent them. *Network Security*, pp. 13–16.

[61.] MacDonald, N. (2013). Time lags in biological models. *Springer Science & Business Media*.

[62.] Macy, J. (2018). API security: Whose job is it anyway? *Network Security*, pp. 6–9.

[63.] Mai, P. X., Pastore, F., Goknil, A., & Briand, L. (2020). Metamorphic Security Testing for Web Systems. *IEEE 13th International Conference on Software Testing, Validation, and Verification (ICST)*, pp. 186–197.

[64.] Malik, S., & Kim, D.-H. (2017). A comparison of RESTful vs SOAP web services in actuator networks. *2017 ninth international conference on Ubiquitous and future networks (ICUFN)*, 753-755.

[65.] Malki, A. E., Zdun, U., & Pautasso, C. (2022). Impact of API Rate Limit on Reliability of Microservices-Based Architectures. *IEEE International Conference on Service-Oriented System Engineering (SOSE)*, 19-28.

[66.] Marilenaa, D., Ivana, H., Silvioa, P., & Davida, S. (2022). Creating RESTful APIs over SPARQL endpoints using RAMOSE. *Semantic Web*, 195-213.

[67.] Martin-Lopez, A., Segura, S., & Ruiz-Cortés, A. (2020). RESTest: Black-Box Constraint-Based Testing of RESTful Web APIs. *Service-Oriented Computing: 18th International Conference, ICSOC 2020, Dubai, United Arab Emirates, December 14–17, 2020, Proceedings 18*, 459-475.

[68.] Miller, J., Zhang, L., Ofuonye, E., & Smith, M. (2008). The Theory and Implementation of InputValidator: A Semi-Automated Value-Level Bypass Testing Tool. *International Journal of Information Technology and Web Engineering (IJITWE)*, pp. 28–45.

[69.] Modi, B., Chourasia, U., & Pandey, R. (2022). Design and implementation of RESTFUL API-based model for vulnerability detection and mitigation. *IOP Conference Series: Materials Science and Engineering*, 012010.

[70.] Musa, A., Empakeris, M., Chan, V., & Chan, &. Y. (n.d.). Security Assessment of istline Market Web Application.

[71.] Nuno Realista et al . (2022). Improving Android Application Quality Through Extendable, Automated

Security Testing. *Emerging Trends in Cybersecurity Applications*, 251-274.

[72.] Okafor, R. (2021). Cybersecurity Due Diligence in Mergers & Acquisitions Transactions. *Available at SSRN*, 3915861.

[73.] Ovidiu Baniaș et al. (2021). Automated Specification-Based Testing of REST APIs. *Sensors*, 5375.

[74.] Ozdemir, E. (2020). A General Overview of RESTful Web Services. *Applications and approaches to object-oriented software design: emerging research and opportunities*, pp. 133–165.

[75.] Padma, P., & Srinivasan, S. (2023). DAuth—Delegated Authorization Framework for Secured Serverless Cloud Computing. *Wireless Personal Communications*, pp. 1–21.

[76.] Padmanaban, R., Thirumaran, M., Anitha, P., & Moshika, A. (2022). Computability evaluation of RESTful API using Primitive Recursive Function. *Journal of King Saud University-Computer and Information Sciences*, pp. 457–467.

[77.] Pan, L., Cohney, S., Murray, T., & Pham, V.-T. (2023). Detecting Excessive Data Exposures in Web Server Responses with Metamorphic Fuzzing. *arXiv preprint arXiv*, 2301.09258.

[78.] Paoli, D., F., P. E., & Zavattaro, G. (2012). Service-oriented and Cloud Computing: First European Conference, ESOCC 2012, Bertinoro, Italy, September 19-21, 2012, Proceedings (Vol. 7592). *Springer*.

[79.] Park, D. B., Li, X., Shahhosseini, A. M., & Tsay, L.-S. (2021). A static code analysis-based mathematical model-driven vulnerability risk assessment framework for health information applications in the Cloud. *International Journal of Forensic Engineering and Management*, 179-208.

[80.] Patel, K. (2019). A Survey on Vulnerability Assessment & Penetration Testing for Secure Communication. *3rd International Conference on Trends in Electronics and Informatics (ICOEI)*, pp. 320–325.

[81.] Peng, C., Gao, Y., & Yang, P. (2022). Automated Server Testing: An Industrial Experience Report. *IEEE International Conference on Software Maintenance and Evolution (ICSME)*, 519-522.

[82.] Pourvahab, M., & Ekbatanifard, G. (2019). Digital Forensics Architecture for Evidence Collection and Provenance Preservation in IaaS Cloud Environment Using SDN and Blockchain Technology. *IEEE Access*, 153349-153364.

[83.] Qingyang Zeng et al. (2023). Full-stack vulnerability analysis of the cloud-native platform. *Computers & Security*, 103173.

[84.] Rafique, W., He, X., Liu, Z., Sun, Y., & Dou, W. (2019). CFADefense: A Security Solution to Detect and Mitigate Crossfire Attacks in Software-Defined IoT-Edge Infrastructure. *IEEE 21st International Conference on High-Performance Computing and Communications; IEEE 17th International Conference on Smart City; IEEE 5th International Conference on Data Science and Systems (HPCC/SmartCity/DSS)*, 500-509.

[85.] Rahman, A., Shamim, S. I., & Bose, D. B. (2023). Security Misconfigurations in Open Source Kubernetes Manifests: An Empirical Study. *ACM Transactions on Software Engineering and Methodology*.

[86.] Rivera, D., García, A., Martín-Ruiz, M. L., & Alarcos, B. (2019). Secure Communications and Protected Data for an Internet of Things Smart Toy Platform. *IEEE Internet of Things Journal*, 3785-3795.

[87.] Rodríguez, G. E., Torres, J. G., Flores, P., & Benavides, D. E. (2020). Cross-site scripting (XSS) attacks and mitigation: A survey. *Computer Networks*, 106960.

[88.] Sánchez, Y. K., Demurjian, S. A., & Baihan, M. S. (2017). Achieving RBAC on RESTful APIs for Mobile Apps Using FHIR. *2017 5th IEEE International Conference on Mobile Cloud Computing, Services, and Engineering (MobileCloud)*, 139-144.

[89.] Sandhya, S., Purkayastha, S., Joshua, E., & Deep, A. (2017). Assessment of website security by penetration testing using Wireshark. *4th International Conference on Advanced Computing and Communication Systems (ICACCS)*, 1-4.

[90.] Schreibmann, V., & Braun, P. (2015). Model-driven development of RESTful APIs. *International Conference on Web Information Systems and Technologies*, 5-14.

[91.] Sean B. Cleveland et al. (2020). Tapis API Development with Python: Best Practices In Scientific REST API Implementation: Experience implementing a distributed Stream API. *Practice and Experience in Advanced Research Computing*, pp. 181–187.

[92.] Serme, G., Oliveira, A. S., Massiera, J., & Roudier, Y. (2012). Enabling Message Security for RESTful Services. *IEEE 19th International Conference on Web Services*, 114-121.

[93.] Setiadi, D. R., Najib, A. F., Rachmawanto, E. H., & Sari, C. A. (2019). A Comparative Study MD5 and SHA1 Algorithms to Encrypt REST API Authentication on Mobile-based Application. *International Conference on Information and Communications Technology (ICOIACT)*, 206-211.

[94.] Shah, S., & Mehtre, B. M. (2015). An overview of vulnerability assessment and penetration testing techniques. *Journal of Computer Virology and Hacking Techniques*, 27-49.

[95.] Sharieh, S., & Ferworn, A. (2021). Securing APIs and Chaos Engineering. *IEEE Conference on Communications and Network Security (CNS)*, 290-294.

[96.] Sharieh, S., & Ferworn, A. (2021). Securing APIs and Chaos Engineering. *IEEE Conference on Communications and Network Security (CNS)*, 290-294.

[97.] Sidra, A., & Michael, M. (2023). A framework for privacy-aware and secure decentralised data storage. *Computer Science and Information Systems*.

[98.] Talukder, M. A., Shahriar, H., Qian, K., Rahman, M., & Ahamed, S. I. (2019). DroidPatrol: A Static

Analysis Plugin For Secure Mobile Software Development. *IEEE 43rd annual computer software and applications conference (COMPSAC)*, pp. 565–569.

[99.] Taya, T., Hanada, M., Murakami, Y., Waseda, A., Ishida, Y., & M, T. (2022). An Automated Vulnerability Assessment Approach for WebAPI that Considers Requests and Responses. *24th International Conference on Advanced Communication Technology (ICACT)*, 423-430.

[100.] Tek Raj Chhetri et al. (2022). Data Protection by Design Tool for Automated GDPR Compliance Verification Based on Semantically Modeled Informed Consent. *Sensors*, 2763.

[101.] Votipka, D., Fulton, K. R., Parker, J., Hou, M., Mazurek, M. L., & Hicks, M. (2020). Understanding security mistakes, developers make Qualitative analysis from Build It, Break It, Fix It. *USENIX Security Symposium (USENIX Security 20)*, pp. 109–126.

[102.] Xiong, H., Jin, C., Alazab, M., Yeh, K.-H., & Wang, H. (2021). On the Design of Blockchain-Based ECDSA With Fault-Tolerant Batch Verification Protocol for Blockchain-Enabled IoMT. *IEEE Journal of Biomedical and health informatics*, 1977-1986.

[103.] Yahya, F., Chang, V., Walters, R., & Wills, G. (2014). Security Challenges in Cloud Storages. *IEEE 6th International Conference on Cloud Computing Technology and Science*, pp. 1051–1056.

[104.] Zhiwei, L., & Zhongliang, P. (2020). The Realisation of Integrity Test of Boundary-Scan Structure. *IEEE International Conference on Artificial Intelligence and Computer Applications (ICAICA)*, pp. 722–724.