A STUDY ON API SECURITY PENTESTING

A Thesis
presented to
the Faculty of California Polytechnic State University,
San Luis Obispo

In Partial Fulfillment
of the Requirements for the Degree
Master of Science in Computer Science

by
Hadi Asemi
June 2023

PREVIEW

PREVIEW

© 2023
Hadi Asemi
ALL RIGHTS RESERVED

COMMITTEE MEMBERSHIP

TITLE: A Study on API Security Pentesting

AUTHOR: Hadi Asemi

DATE SUBMITTED: June 2023

COMMITTEE CHAIR: Dongfeng Fang, Ph.D.
Professor of Computer Science

COMMITTEE MEMBER: Bruce Edward DeBruhl, Ph.D.
Professor of Computer Science

COMMITTEE MEMBER: Devkishen Sisodia Ph.D.
Professor of Computer Science

PREVIEW

ABSTRACT

A Study on API Security Pentesting

Hadi Asemi

Application Programming Interfaces (APIs) are essential in the digital realm as the bridge enabling seamless communication and collaboration between diverse software applications. Their significance lies in simplifying the integration of different systems, allowing them to work together effortlessly and share data. APIs are used in various applications, for example, healthcare, banks, authentication, etc. Ensuring the security of APIs is critical to ensure data security, privacy, and more. Therefore, the security of APIs is not only urgent but mandatory for pentesting APIs at every stage of development and to catch vulnerabilities early. The primary purpose of this research is to provide guidelines to help apply existing tools for reconnaissance and authentication pentesting. To achieve this goal, we first introduce the basics of API and OWASP's Top 10 API security vulnerabilities. Secondly, we propose deployable scripts developed for Ubuntu Debian Systems to install pentesting tools automatically. These scripts allow future students to participate in API security courses and conduct API security pentesting. API security pentesting, regarding reconnaissance and authentication, is discussed based on the configured system. For reconnaissance, passive and active approaches are introduced with different tools for authentication, including password-based authentication brute-forcing, one-time password (OTP) brute-forcing, and JSON web token brute force.

PREVIEW

ACKNOWLEDGMENTS

I extend my deepest gratitude to my parents, whose unwavering support and love have been the bedrock of my journey. Their encouragement and sacrifices have fueled my aspirations, and I am profoundly thankful for the sacrifices they've made to ensure my education. Their constant belief in my potential has been a guiding light, and I am forever grateful for their immeasurable contributions to my academic and personal growth.

I would also like to express my sincere appreciation to my esteemed professor Dr. Dongfeng Fang, whose guidance and mentorship have been invaluable throughout this academic endeavor. Her expertise, dedication, and encouragement have shaped my intellectual development and enriched my learning experience. I am grateful for the time and effort she invested in imparting knowledge, challenging my perspectives, and fostering an environment of intellectual curiosity. Her influence has left an indelible mark on my academic journey, and I am indebted to her for her unwavering support.

In addition, I want to thank my committee Dr. Brue DeBruhl, and Dr. Devkishen Sisodia for their support, expertise, and constructive feedback. Their dedication has played a pivotal role in shaping the outcome of my research.

Finally, I would like to thank friends and classmates whose steadfast support, encouragement, and motivation have served as the cornerstones of my academic journey. Your kindness and faith in my abilities have been a driving force, propelling me through both challenges and triumphs.

PREVIEW

TABLE OF CONTENTS

Page

LIST OF TABLES x
LIST OF FIGURES xi

CHAPTER

1 Introduction 1
  1.1 Basics of API 1
    1.1.1 Different Types of API Protocols 2
      1.1.1.1 REST 2
      1.1.1.2 SOAP 3
      1.1.1.3 GraphQL 4
    1.1.2 Popular Applications in API 4
      1.1.2.1 E-commerce 4
      1.1.2.2 Banking System 5
      1.1.2.3 Epic Electronic Health Record (EHR) 5
      1.1.2.4 Stripe 6
  1.2 Security and Privacy of API 6
    1.2.1 Confidentiality 6
    1.2.2 Integrity 8
    1.2.3 Availability 9
  1.3 Breaches 10
    1.3.1 Coinbase 11
    1.3.2 US Postal Service (USPS) 11
    1.3.3 Peloton's Leaky API 12

PREVIEW

1.3.4 Venmo 12

1.3.5 Instagram 13

1.3.6 Bumble 13

1.3.7 T-Mobile 14

1.3.8 OPTUS 14

1.4 Motivation 14

1.5 Contribution 15

2 OWASP Top 10 API Security Vulnerabilities 17

2.1 API I: Broken Object Level Authorization (BOLA) 18

2.2 API II: Broken User Authentication 19

2.3 API III: Excessive Data Exposure 20

2.4 API IV: Lack of Resources & Rate Limiting 21

2.5 API V: Broken Function Level Authorization (BFLA) 23

2.6 API VI: Mass Assignment 24

2.7 API VII: Security Misconfiguration 26

2.8 API VIII: Injection 27

2.9 API IX: Imporper Assets Management 29

2.10 API X: Insufficient Logging & Monitoring 29

2.11 Comparision of OWASP API Top 10 2019-2023 30

2.12 Ethical Statement 31

2.12.1 Practical Hacking Practices: 32

3 API System and Vulnerability Environment Setup 34

3.1 Tools 34

3.1.1 Burpsuite 34

3.1.2 Ffuf 35

PREVIEW

3.1.3 Wfuzz 35

3.1.4 FireFox Plugins 36

3.1.5 Zaproxy 37

3.1.6 Jwt_tool 37

3.1.7 Postman 38

3.1.8 Amass 39

3.1.9 Burp Suite vs Zaproxy 40

3.1.10 Ffuf vs Wfuzz 41

3.1.11 Rustscan vs Nmap 43

3.2 Automatic Install Script 45

3.2.1 Ansible 45

3.2.2 Jinja 48

3.2.3 Configure Foxyproxy 50

3.2.4 Burp Suite Extentions 51

3.2.5 Limitation of Script 52

4 API Security Tools Testing: Reconnaisance and Authentication 54

4.1 API Reconnaissance 54

4.1.1 Passive 55

4.1.1.1 Google Dorking 56

4.1.1.2 Git Dorking 57

4.1.1.3 Wayback Machine: 61

4.1.1.4 Shodan: 64

4.1.1.5 Amass: 65

4.1.2 Active 69

4.1.2.1 Port Scanning 70

PREVIEW

4.1.2.2 Burp Suite 72

4.1.3 Gobuster 73

4.2 Authentication 74

4.2.1 Password-based Authentication Brute-Force Attack 75

4.2.1.1 Wfuzz 76

4.2.1.2 Ffuf 77

4.2.2 One Time Password (OTP) Brute-Force Attack 78

4.2.3 JSON Web Tokens (JWT) 81

4.2.3.1 Attacking JWT 84

5 Conclusion & Future Work 88

5.1 Conclusion 88

5.2 Future Work 89

BIBLIOGRAPHY 91

PREVIEW

LIST OF TABLES

Table Page

1.1.1.1 Comparison table SOAP vs REST API 3

3.1.10.1ComparisonFfuf and Wfuzz 41

3.1.11.1Rustscan vs Nmap 44

4.1.0.1 Reconnaissance Approaches 55

4.1.1.1 Google Dorking Queries [1] 57

4.1.1.2 Git Dork keys 59

4.1.1.3 Oam-tools default flags [2] 67

4.1.1.4 Oam_subs flags [2] 68

4.1.1.5 The oam_viz flag for showing network graph 69

4.1.2.1 Nmap flags and use cases 70

PREVIEW

LIST OF FIGURES

Figure Page

1.1.0.1 Communication between devices and backend with the help of API 2

2.11.0.1OWASP Top 10 API 2019 vs. 2023 [1] 31

3.1.8.1 Under the hood of Amass database [3] 40
3.1.10.1Wfuzz time performance 42
3.1.10.2FFuf time performance 43
3.2.1.1 Ansible folder tree structure 46
3.2.2.1 Choose the right terminal font 49
3.2.2.2 Ubuntu desktop after configuration 50
3.2.3.1 Foxyproxy Configuration 50
3.2.4.1 Burp Suite Authorize Plugin 51
3.2.4.2 Setup Jython 52
3.2.5.1 Golang installation 53

4.1.1.1 NASA GIT-OAUTH pushed to GitHub 58
4.1.1.2 Github-dorks usage 60
4.1.1.3 Wayback Machine for owasp.org 63
4.1.1.4 Search Shodan for ”content-type: application/json” 64
4.1.1.5 Amass intel subcommand 65
4.1.1.6 OWASP root domain with using Amasss 65
4.1.2.1 Nmap for crapi.apisec.ai 71
4.1.2.2 Rustscan help page 72

PREVIEW

4.1.2.3 Burp Suite intercept creating account 73
4.2.1.1 Crack the user password and get 200 response 76
4.2.1.2 Post request intercepted by Bursp Suite 77
4.2.1.3 Crack the user password and get 200 responses 79
4.2.2.1 CrAPI OTP password reset 79
4.2.2.2 OTP Burp Suite intercept 80
4.2.2.3 Brute-force OTP 81
4.2.3.1 JWT authentication between server and client 83
4.2.3.2 Authentication with website and get token with Burp Suite 84
4.2.3.3 jwt_tool JWT token analysis 85
4.2.3.4 Crack Attack 87
4.2.3.5 Using jwt.io to generate token 87

PREVIEW

Chapter 1

INTRODUCTION

1.1 Basics of API

API stands for Application Programming Interface. It is a set of rules, protocols, and tools for building software applications that allow different software systems to communicate with each other [5].

It is a way that software applications talk to each other and exchange data. These are three different types of API communication: a web API (which uses the HTTP/HTTPS protocol to send and receive data), a database API (which allows the software to access and manipulate data in a database), or a hardware API (which enables software to communicate with hardware devices). APIs are widely used for many applications, such as the weather app or your stock app on your phone, which communicates with the server to give you the most updated information.

When working with APIs, it's essential to consider both the client and server aspects [6]. The client is the application that runs on each device and sends the request, while the backend responds to that request. As you can see in the diagram in 1.1.0.1. The device sends a request, and the backend server gets the information from the databases and sends back the data.

PREVIEW

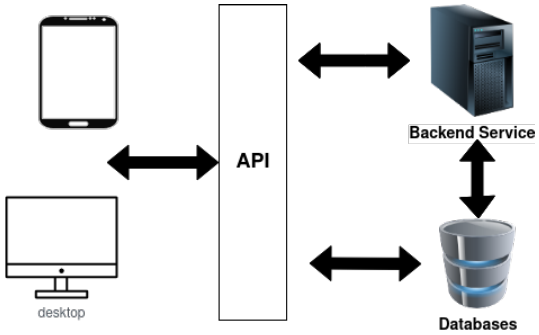

Figure 1.1.0.1: Communication between devices and backend with the help of API

1.1.1 Different Types of API Protocols

To utilize APIs proficiently, developers need to follow a shared set of guidelines or protocols while initiating API requests [5]. In this section, I will be discussing the three types of API protocols available.

1.1.1.1 REST

Representational State Transfer, commonly known as REST, is a design approach that facilitates communication between a client and server. This design can be implemented in multiple ways, allowing users to access databases. By requesting the appropriate header, users can obtain the information they require [14]. The information obtained through HTTP can be in different formats, such as JSON, HTML, XML, or plain text. The REST API adheres to the CRUD functions: create, read, update, and delete. By utilizing a stateless REST API, you can significantly enhance the performance and scalability of your system. The beauty of this approach is that it eliminates the need for the server to keep track of the client's state or previous messages. This results in unparalleled reliability, lightning-fast performance, and seamless scalability that can be effortlessly managed, updated, and reused without

PREVIEW

adversely impacting the system's overall functionality. With a stateless architecture, you can build a rock-solid and highly adaptable system that can easily accommodate changing needs and requirements over time [13].

1.1.1.2 SOAP

Simple Object Access Protocol (SOAP) is a technology widely used for communication between operating systems using HTTP and XML [15]. It stands for Simple Object Access Protocol and is known for its extensibility, neutrality, and independence. With SOAP, developers can easily maintain accounts and searches using any programming language they prefer. In table 1.1.1.1 you can see difference between SOAP and REST API.

PREVIEW

Table 1.1.1.1: Comparison table SOAP vs REST API

Aspect	**SOAP API**	**REST API**
Standards	Has specific standards and protocols.	Does not have official standards due to its architectural style.
Functionality	Limited to HTTP and XML.	Utilizes uniform resource locator (URL) and HTTP.
Business Logic	Relies on service interfaces like @WebService.	Utilizes URL interfaces (/WeatherService).
Bandwidth Usage	Requires more bandwidth due to XML payload.	Uses less bandwidth.
Description Language	Uses Web Services Description Language.	Uses Web Application Description Language.
Error Handling	Uses specific standards, which can lead to miscommunication and errors.	More flexible but even a tiny error could corrupt the API.
Implementation Ease	Requires adherence to specific standards.	Easier to implement using specific programming languages.
Rules and Standardization	Must follow particular rules for standardization.	Doesn't require as many

1.1.1.3 GraphQL

Facebook created GraphQL, a query language that enables the creation of client applications using an intuitive and flexible syntax [16]. GraphQL is a data query language that allows users to request specific data sets tailored to their needs. This contrasts REST APIs, which typically return a fixed data set. GraphQL's flexibility makes it a good choice for mobile apps, as it can reduce the amount of data that needs to be transferred over the network. Another advantage of GraphQL is that it can scale to handle more requests and data. GraphQL uses a single URL for fetching and mutating data, making it easier to maintain than REST APIs, which require multiple endpoints. GraphQL is a powerful and flexible data query language that can improve APIs' performance, scalability, and simplicity.

1.1.2 Popular Applications in API

Many applications rely on APIs to facilitate front and back-end communication. In this section, I will discuss a few examples.

1.1.2.1 E-commerce

E-commerce refers to buying and selling goods, products, or services over the Internet. Excellent examples are Amazon, Best Buy, and Shopify. APIs play a vital role in e-commerce by enabling integration with payment gateways for secure transactions, shipping, and logistics services for accurate tracking and inventory management, order management systems for streamlined processing, CRM systems for personalized customer experiences, social media platforms for marketing and engagement, review and rating systems for transparency, analytics platforms for data-driven decision-making,

PREVIEW

and more. These APIs allow e-commerce platforms to seamlessly connect with various services, enhance functionality, and provide a seamless shopping experience for customers while optimizing backend operations.

1.1.2.2 Banking System

APIs have revolutionized the banking system in how users communicate with the server to perform transactions, deposit checks, check balances, and authenticate their identity. This integration has made banking more accessible, convenient, and faster. In addition, with the power of APIs, users can now enjoy the luxury of banking from their mobile applications, eliminating the need to visit the bank physically. As a result, using the API is both time-saving and a more cost-effective solution for banks in the long run.

PREVIEW

1.1.2.3 Epic Electronic Health Record (EHR)

The Epic electronic health record (EHR) software from Epic Systems Corporation uses APIs to provide seamless integration and interoperability across the medical and healthcare system. To facilitate the real-time sharing of patient information and improve care coordination, APIs are used to share data with external systems, including laboratory information systems, radiology systems, and pharmacy systems [20]. Patients may access their health information and communicate with healthcare professionals. APIs assist clinical decision support tools, integrate telemedicine and remote monitoring platforms, enable data analytics and reporting, and make it easier for third parties to interface with internal systems. They also allow interoperability with external systems like health information exchanges. APIs are used by Epic and other medical and health systems to facilitate data sharing, teamwork, and better patient care. As a result, the security of this application is in high priority because if doctors can not access patients' records, it can be caused put the life of the patients in danger.

1.1.2.4 Stripe

Stripe, a famous payment processing company, uses APIs to enable businesses to take online payments securely. APIs handle the basic payment processing operations, allowing businesses to incorporate Stripe's payment gateway into their websites or applications. This integration will enable clients to pay via various methods, while APIs assure the secure transmission and authorization of transactions.

PREVIEW

1.2 Security and Privacy of API

To ensure that our API is secure, we must consider three essential pillars of security: confidentiality, integrity, and availability, commonly called CIA. In this section, I will go over more, expanding on how these affect API systems.

1.2.1 Confidentiality

Protecting sensitive data from unauthorized access, modification, or disclosure is known as confidentiality. It is essential to consider factors such as insufficient authentication and authorization, lack of encryption, insecure storage of sensitive data, inadequate access controls, inadequate logging monitoring, API misconfiguration, and insecure third-party integration to ensure confidentiality.

1. **Insufficient authentication and authorization:** To maintain security, we must implement rigorous authentication and authorization protocols that only restrict access to authorized users. As an added measure, we must incorporate a second-factor authentication (2FA) system to enhance security and make it more difficult for unauthorized individuals to access the system.

2. **Lack of encryption:** It is essential to use encryption such as SSL/TLS. Encryptions prevent attackers from intercepting the network and accessing the data being transmitted. An attacker could easily sniff the packet without encryption and access a user's credentials in plain text.

3. **Insecure storage of sensitive data:** APIs frequently handle private user data, such as login credentials, date of birth, and even social security numbers. To avoid data breaches and ensure confidentiality, this data must be protected. One efficient strategy is to ensure secure user data storage by putting strong encryption measures in place for servers and databases that handle such data. By adding a layer of security through encryption, these systems increase security and make it far more difficult for unauthorized individuals to access and utilize sensitive user data. Optimizing encryption may improve overall security posture and lower data breach danger.

4. **Inadequate access controls:** Establishing appropriate access restrictions for different users or applications is essential. Without adequately configuring these controls, unauthorized individuals may gain visibility into sensitive information, leading to potential data breaches or privacy violations. By implementing robust authentication and authorization mechanisms, API providers can ensure that only authorized entities can access sensitive data. Additionally, employing role-based access control and implementing fine-grained permission settings

PREVIEW

can further enhance the security posture of the API, limiting access to specific resources based on the user's role and privileges.

5. **Inadequate logging and monitoring:** Strong logging and monitoring systems should be in place for APIs to identify and react to unauthorized access or suspicious activity attempts. Finding possible vulnerabilities or detecting breaches of confidentiality might be difficult without effective recording and monitoring.

1.2.2 Integrity

Ensuring data accuracy, completeness, and consistency throughout an application is known as integrity. Breaches in API integrity can result in unauthorized modifications, tampering, or data corruption, which can have severe consequences for systems, applications, and users relying on the API. A study by Mendoza et al. (2018) found that many mobile apps tidy up user input within the app, but if this differs from the server, it can pose significant security risks [17]. In addition, hackers can manipulate the data by identifying the API endpoint, enabling them to purchase items without payment. This underscores the importance of maintaining data integrity. In this section, I will be discussing various vulnerabilities that have the potential to compromise the integrity of APIs.

PREVIEW

1. **Lack of input validation and sanitization:** To guard against numerous security concerns, it is essential to incorporate strong input validation and sanitization procedures while designing an API. Input validation's main objective is to make sure that user-provided data adheres to the desired format and fulfills the relevant requirements. On the other hand, sanitization entails removing any potentially hazardous or undesired material by filtering and cleaning user in-

put. By validating and sanitizating the input we can metigate SQL injectection, prevent cross-site scritpting (XSS), etc.

2. **Broken session management:** Handling sessions securely when using APIs that require session management mechanisms is vital. For example, to prevent session hijacking or fixation attacks, session tokens should be adequately generated, stored securely, and invalidated after logout or a certain period of inactivity.

3. **Lack of versioning and backward compatibility:** Changes to API structures or behaviors without proper versioning and backward compatibility can break client integrations and cause integrity issues. For example, we patch the older version's vulnerabilities in the newer one, but the other version is still available. In that case, the attacker can still use the older version to attack our API. Therefore, implement versioning strategies and communicate changes effectively to ensure smooth transitions and avoid unintended data modifications.

1.2.3 Availability

PREVIEW

Authorized users must have uninterrupted access to the application, known as availability. This is because any disruptions caused by attackers can lead to service outages, performance issues, or denial of service. This section covers some of the vulnerabilities that can happen regarding availability.

1. **DDoS attacks:** Distributed Denial of Service (DDoS) attacks try to overload the API infrastructure by sending many fake requests, which can disrupt the service. To reduce the impact of these attacks, it's vital to use DDoS protection methods like rate limiting, traffic filtering, and load balancing. In addition,

using services like Content Delivery Networks (CDNs) can also help by distributing traffic and absorbing volumetric attacks.

2. **Resource exhaustion:** The overuse of system resources like CPU, memory, or storage can result from improper management of APIs, making them subject to attacks. Adopting measures like input validation, rate limitation, and resource monitoring is critical to prevent this. Placing reasonable restrictions on request numbers, response times, and other resource utilization criteria is also advised to guarantee equitable distribution and stop misuse.

By focusing on confidentiality, integrity, and availability (CIA), we can create a secure API that can withstand any attempts of hacking or unauthorized access. It is crucial to prioritize security and testing, even if it means taking a little extra time to deliver the product. In the long run, it will save a lot of time and money that could be lost due to a security breach. So, let's prioritize security while developing applications to protect ourselves and our users from any potential harm.

PREVIEW

1.3 Breaches

Data breaches have become an all-too-common occurrence in today's digital age, with businesses big and small falling prey to cybercriminals. This especially worries smaller companies, as they often need more resources to combat such attacks. The root cause of these security breaches is often the pressure to meet deadlines and launch products quickly, leading to insufficient testing of applications. In many cases, API development is divided into two groups: the engineering team, which may lack security expertise, and the security team, which may lack API knowledge [19]. To prevent such incidents, developers must prioritize secure coding and ensure that APIs are configured securely. By implementing these measures, businesses can protect themselves from potential security threats and safeguard their customers' sensitive information. It's imperative to prioritize security in today's world, where technology plays a vital role. This section will discuss the recent security breaches in the past few years.

1.3.1 Coinbase

On February 11th, 2022, the Twitter account reported that they had discovered a potential vulnerability that could have significant market implications for Coinbase. Specifically, it was found that the Retail Brokerage API endpoint was missing a logic validation check, which allowed users to submit trades to a specific order book using a mismatched source account. In this vulnerability, users can scrape the API calls, manipulate four critical parameters in the API call, and sell the crypto they did not own to the user [7]. This vulnerability is Broken Object Level Authorization (BOLA), mentioned in the OWASP top 10 API vulnerability. This issue highlights the importance of implementing thorough security measures to protect against threats and vulnerabilities.

PREVIEW

1.3.2 US Postal Service (USPS)

The US Postal Service (USPS) had a flaw in its Informed Visibility program that could have exposed the personal data of around 60 million users. Security expert Brian Krebs discovered that the USPS API allowed users to request changes to other accounts without checks to prevent unauthorized access. While passwords were not exposed, spammers and phishing attackers could have exploited the vulnerability by building up mass-targeted spam [8]. In addition, the API allowed users to convert accounts into Informed Visibility business accounts and vice versa, which could have created issues for the USPS's largest customers. This is another Broken Object Level Authorization.

1.3.3 Peloton's Leaky API

A security flaw was discovered, which allowed unauthorized access to the private account information of ride users. The open-source API permitted requests for user data without any authentication. This exposed four million user accounts, including those marked as private, such as Joe Biden's. Despite the severity of the situation, the company failed to respond for 90 days [9]. However, the vulnerability was eventually resolved by incorporating authentication measures. These security flaws include two OWASP top 10 such as Broken Object Level Authorization and Broken Authentication.

1.3.4 Venmo

Venmo is a popular app for sending and receiving money, with a live feed of transactions being made by strangers displayed on the home page. A hacker discovered that the data for this feed was accessible through a public API endpoint, meaning that anyone could make a GET request to see the latest 20 transactions made on the app by anyone around the world, even outside the app, with no authorization required. The hacker scraped this data and found that they could download 115,000 transactions daily, leading them to question the ease with which they could amass an extensive collection of people's financial activity. The hacker discovered that the public data is not as innocuous as it might seem. It can reveal information such as which app is used for business on Venmo, which can be useful for nefarious purposes such as phishing for Apple ID credentials [10]. These vulnerabilities broked the three

PREVIEW

Reproduced with permission of copyright owner. Further reproduction prohibited without permission.