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

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

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.

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.

TABLE OF CONTENTS

LIST OF TABLES

Table Page

LIST OF FIGURES

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.

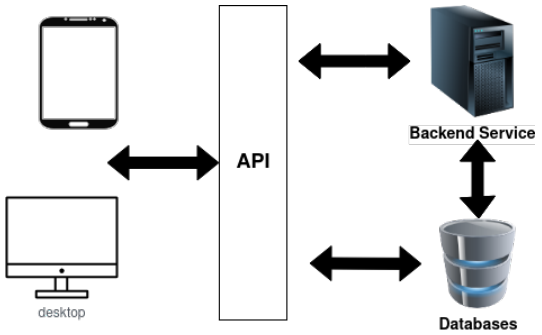


Figure 1.1.0.1: Communication between devices and backend with the help of API

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

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.

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,

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.

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.

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

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.

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

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.

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.

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

OWASP top 10 API security, such as Broken Authentication, Broken Object Level Authorization, and Unrestricted Resource Consumption.

1.3.5 Instagram

It's been reported that Instagram recently experienced a security breach that allowed hackers to access the personal information of high-profile users. The hackers exploited the Application Programming Interface (API) bug to steal email addresses and phone numbers. A 6-digit code is required to reset accounts, which the hackers could guess using the API. While the guesses were limited to 200 per IP, researchers discovered that by rotating through 5,000 IPs, the hackers could take over any account they wanted [11]. The security breach was caused by two vulnerabilities in the API - Broken Object Level Authorization and Broken Authentication. This is a concerning issue that Instagram needs to address as soon as possible.

1.3.6 Bumble

It has come to light that there was a significant breach in the security of a specific API, which allowed unauthorized access to the details of a staggering 95 million user accounts. Even more concerning is that incrementing the IDs led to scrapping the entire database. This breach also enabled the exact location of users to be calculated through triangulation. Another issue that arose was that paid features could be enabled without the proper privileges. As a result of these vulnerabilities, there were multiple instances of Broken Object Level Authorization, Broken Authentication, and Broken Function Level Authorization. This is a severe problem that needs to be addressed immediately.

1.3.7 T-Mobile

Recently, a security breach occurred on January 5th, where an attacker gained access to 37 million customer accounts on postpaid and prepaid plans due to a weak API. Without any authentication, the attacker could get users' data. T-Mobile has stated that only certain personal information such as names, billing addresses, emails, phone numbers, and dates of birth was exposed [18].

1.3.8 OPTUS

An Australian telecommunications company, Optus, was targeted by a cyber attacker who successfully obtained and collected 9.8 million user details [12]. Alongside this breach, the attacker also issued a ransom demand of $1 million. The compromised data consisted of driver's licenses, Medicare IDs, names, phone numbers, and email addresses. This security breach occurred due to the absence of authentication protocols on API endpoints and the company's failure to enforce restrictions on data resource usage.

1.4 Motivation

Application Programming Interfaces (API) are growing in various areas like IoT, websites, medical data, and banking. As a result, it is essential to ensure that the resources remain confidential, integral, and available. It is also crucial to understand that security by design for API implementation. Unfortunately, many API develops lack knowledge of security and privacy. Moreover, pentesting is not well applied before the APIs are used. One common API vulnerability is broken object-level authorization (BOLA), which controls users' access to resources. Exposures can occur

when users can access other users due to authorization flaws. For instance, the UPS hacks were due to BOLA, where users could authenticate with the server and pivot to other users' accounts.

1.5 Contribution

The significance of my thesis research is multifaceted, encompassing several key contributions:

1. **Comparative Analysis:** I conduct a comprehensive comparative analysis between OWASP's top 10 API security concerns in 2019 and the latest trends 2023. This examination sheds light on the evolving landscape of API security vulnerabilities, providing valuable insights for the cybersecurity community.

2. **Tool Assessment and Demonstration:** I identify and evaluate a range of open-source tools designed for the penetration testing of APIs. Notably, I demonstrate the practical usage of these tools, offering a hands-on approach to understanding their capabilities and utility in securing APIs.

3. **Automated Configuration with Ansible:** I developed a sophisticated Ansible script as part of my research. This script empowers users to effortlessly configure Ubuntu operating systems, equipping them with the full suite of penetration testing tools showcased in my study. This automation streamlines the setup process, saving time and effort for security practitioners.

4. **Scalable Deployment:** I design a deployment strategy that enables the seamless rollout of the configured environment to multiple servers. This infrastructure flexibility paves the way for future scalability and allows professors to facilitate API pentesting classes for students easily.

5. **Addressing Critical API Security Challenges:** My comprehensive research rigorously addresses two paramount facets of API security: reconnaissance and authentication. In a meticulous exploration of passive reconnaissance, I adeptly deploy a diverse array of powerful tools, including Gooogle Dorking, Git Dorking, Wayback Machine, Shodan, Amass, and TruffleHog. For active reconnaissance, I delve into the intricacies of Nmap, Rustscan, Burp Suite, Zaproxy, and Gobuster. Authentically illuminating the realm of authentication, my study encompasses password-brute forcing with Wfuzz Ffuf, OTP brute force methods, and a meticulous examination of JWT attacks utilizing Jwt_tool for penetrating tests. By focusing on reconnaissance and authentication, I provide actionable insights and solutions to bolster the protection of APIs, addressing vulnerabilities at their root.

In the following chapters, my thesis embarks on an extensive exploration of API security, aiming to provide a comprehensive and impactful contribution to the field. In Chapter 2, I meticulously examine the evolving landscape of API security, focusing on OWASP's Top 10 API security vulnerabilities, elucidating the differences between the 2019 and 2023 editions. Chapter 3 introduces and explains a suite of penetration testing tools designed explicitly for APIs while culminating in unveiling an Ansible script simplifying Ubuntu 22.04 configuration for penetration testing. Chapter 4 delves even more profound, offering an in-depth analysis of each tool's flags and parameters, empowering practitioners with practical insights. Chapter 5 charts a visionary course for future research in API security, and finally, Chapter 6 synthesizes the thesis's findings, insights, and contributions, providing a profound conclusion that underscores the research's significance and impact in the realm of API security.

Chapter 2

OWASP TOP 10 API SECURITY VULNERABILITIES

This chapter examines the top ten API security issues identified by the OWASP (Open Web Application Security Project). The Open Online Application Security Project (OWASP) is a well-known organization that gives significant insights and assistance on online application security. The OWASP top ten API security vulnerabilities emphasize the most severe dangers connected with APIs and provide mitigation advice [24].

Our comprehensive examination will delve into the intricacies of each of the OWASP top ten API security issues, unraveling subjects such as faltering authentication and session management, inappropriate authorization, sensitive data exposure, and the absence of robust resource and rate limits, among others. Emphasizing the critical role of authentication, we highlight any misconfiguration in this aspect could provide attackers unauthorized access to restricted resources. Equipping developers and security experts with knowledge about these issues empowers them to proactively identify and address potential security vulnerabilities in their API implementations.

In addition, we will compare the 2019 and 2023 versions of the OWASP top 10 API security issues. This comparison will illuminate any new patterns or threats in the API security landscape. It will provide vital insights into how the security landscape has developed, allowing firms to keep current on best practices and security procedures.

2.1 API I: Broken Object Level Authorization (BOLA)

Object-level authorization is a vital access control strategy frequently employed at the code level to guarantee users only access authorized objects [24]. This security measure is crucial in protecting API endpoints from unauthorized access and is widely recognized as an industry best practice.

The object-level authorization ensures users have the appropriate privileges to access specific resources or objects within an application or system. By enforcing strict controls at the code level, organizations can prevent unauthorized users from manipulating identifiers or parameters to gain access to sensitive data belonging to other users.

One of the most prevalent security vulnerabilities in the API landscape is the exploitation of object-level authorization flaws. This occurs when an attacker successfully manipulates the identification mechanisms or parameters used in API calls to retrieve or modify data that should be inaccessible.

To illustrate this vulnerability, let's consider a scenario where two users, A and B, are part of an online platform. User A, through a flaw in the object-level authorization implementation, manages to manipulate an ID parameter in an API request. Consequently, the API mistakenly grants access to user B's private data, which should have been restricted to user B exclusively. This breach of object-level authorization can have severe consequences, ranging from unauthorized disclosure of sensitive information to unauthorized modifications or data deletion.

Developers and security practitioners must implement robust access control mechanisms to mitigate the risks associated with object-level authorization vulnerabilities. This includes carefully validating and sanitizing input parameters, utilizing proper

authentication and authorization frameworks, and regularly auditing and reviewing the security of APIs to identify and address any potential weaknesses.

2.2 API II: Broken User Authentication

Ensuring robust security involves safeguarding authentication endpoints. Equally vital is the treatment of "Forgot password / reset password" processes, which should be handled with the same diligence as primary authentication mechanisms. The vulnerability arises if malicious actors exploit credential stuffing, leveraging stolen credentials from external databases to gain unauthorized access.

Furthermore, fortifying the API against brute-force attacks is imperative by implementing countermeasures such as captchas or account lockout mechanisms. Strengthening the initial line of defense, stringent password policies should be enforced by default during account creation.

Continuing the defense, a meticulous approach entails thorough token validation, encompassing validation of token integrity and expiration dates. To uphold confidentiality, passwords must never be stored in plain text, devoid of encryption, weakly hashed, or reliant on feeble encryption keys. The API can effectively thwart potential threats by adhering to these comprehensive security measures.

A great guideline provided by OWASP Cheat sheet would help with authentication setup and the correct way of authentication.

2.3 API III: Excessive Data Exposure

Exposing sensitive data by mistake is a big concern in software design. Despite our efforts to hide this data on the user's device, clever attackers can still find ways to get it. They might spy on or capture the data sent between the user and the server. This can lead to them accessing critical information they shouldn't have access to.

Imagine an online store where someone who wants to cause trouble could add a harmless comment to an online chat. When the server replies, it might accidentally give away essential details meant to stay private. This situation exposes valuable information to the wrong person, which is a big problem.

To solve this, we need to be diligent in our efforts. Relying only on hiding data on the user's device isn't enough. The real solution is to make sure the server doesn't share any secret information in the first place. We do this by creating strict rules for the server to follow so it only shares pertinent user data with users.

Additionally, we need to be mindful about how we handle data. Using simple methods that change data into a particular format, like to_json() or to_string(), can accidentally provide more information than we want. Instead, we create specific methods that only give the exact information we intend to share.

Another good idea is to have a special security check. This check looks at the data before it's given to the user. It makes sure everything is okay and follows the rules we set. This way, even if something sneaky gets past the first defenses, this check can catch it and keep the data safe.

Ultimately, keeping sensitive data safe is a mix of different techniques. It's not only about hiding it on the user's side, it's about ensuring the server shares only what

needs to be shared, being careful with how we change data, and having a final check to ensure everything is secure.

2.4 API IV: Lack of Resources & Rate Limiting

In API functionality, a crucial factor often disregarded pertains to the reasonable allocation of essential resources. These resources include network bandwidth, CPU processing capability, memory allotment, and storage capacity. The user input nature and the intricate business logic inherent in various API endpoints directly influence how these resources are parceled out.

However, a notable aspect to consider in this resource allocation is the competition among multiple API clients competing for the same pool of resources. This competitive scenario introduces complexity, necessitating prudent resource distribution among many incoming requests.

APIs are vulnerable when the necessary protective measures are not enforced. Lack of rate limiting during authentication can make an API susceptible to brute force attacks, allowing unauthorized access to resources. Proper boundaries play a crucial role in API security. Excess, rigid, or lenient boundaries can make an API more vulnerable. It is essential to configure boundaries optimally to avoid security breaches. Certain critical thresholds also come into play in this context.

1. Execution Timeouts: The designated timeframe for the execution of a specific API request. The absence of this constraint or setting it at an overly liberal level can result in resource bottlenecks and potential misuse.

2. Max Allocable Memory: The upper ceiling on the memory amount that can be assigned to a solitary API request. Inadequate allocation may lead to memory overflows or inefficient resource utilization.

3. Number of File Descriptors: A cap on the count of concurrently open files by the API. Inordinate file openings can strain system resources.

4. Number of Processes: An API can generate the utmost quantity of processes. Unchecked process generation might trigger resource depletion.

5. Request Payload Size: The scale of data permissible in a single API request. Outsize payloads can congest the network and hamper efficiency.

6. Number of Requests per Client/Resource: A restriction on the volume of requests a particular client or resource can initiate within a specified span. Insufficient rate limiting can lead to overwhelming the API.

7. Number of Records per Page: The quantum of records the API can provide in response to a sole request. This could lead to ineffectual data retrieval and heightened resource consumption without suitable control.

Mitigating these vulnerabilities mandates a comprehensive strategy involving resource management and prudent rate limiting. Using a Docker streamlines the control of memory, CPU, restart frequency, file descriptors, and processes while enhancing API management by enforcing limitations on client calls within set timeframes and notifying clients when exceeded, sharing limit counts and reset times. Additionally, it bolsters the API's security with thorough server-side validation, especially for parameters influencing response record quantity. Strengthen data integrity by imposing strict limits on incoming elements and payloads, encompassing maximum data size, string length, and array size.

2.5 API V: Broken Function Level Authorization (BFLA)

Broken Function Level Authorization (BFLA) is a critical security vulnerability outlined in the OWASP (Open Web Application Security Project) Top Ten list. It refers to an application allowing users to perform functions or actions they are not authorized to access based on their privilege level. In the context of APIs (Application Programming Interfaces), BFLA vulnerabilities can lead to severe security breaches and unauthorized access to sensitive resources or operations.

In the case of API IV, the primary concern revolves around improper enforcement of authorization rules for various CRUD (Create, Read, Update, Delete) operations. Let's delve into the specific vulnerabilities that can happen based on CRUD:

1. Create Operation Vulnerability: Imagine a scenario where a regular user can create new users or tasks, even though these actions should only be allowed for administrators. This vulnerability could allow an attacker to flood the system with unauthorized accounts or tasks, potentially leading to resource exhaustion or manipulation of user data.

2. Update Operation Vulnerability: Similarly, if the application allows a user to update any user's information or tasks, regardless of their role, this can lead to unauthorized modifications. An attacker could alter crucial account details or task attributes, disrupting the system's integrity and leading to user confusion.

3. Delete Operation Vulnerability: Allowing non-admin users to delete other users or tasks can have serious consequences. An attacker could perform malicious deletions, causing data loss and service disruption and potentially rendering the application unusable for legitimate users.

Measures to mitigate broken function level authorization include:

1. The default approach should prohibit all access, necessitating explicit permissions for each role to access individual functions.
2. Evaluate your API endpoints for vulnerabilities related to function-level access control while considering the application's operational rules and hierarchical groups.
3. Ensure that all administrative controllers are derived from an abstract administrative controller. This parent controller should include checks for authorization based on the user's group or role.
4. Confirm that administrative tasks within standard controllers incorporate authorization validations that rely on the user's group and role.

2.6 API VI: Mass Assignment

Mass Assignment vulnerabilities arise when an unauthorized actor gains the ability to overwrite properties of objects they are not supposed to modify. A specific set of conditions must align for this to occur: an API should possess endpoints that accept user input, these requests must hold the capability to modify concealed values, and the API ought to lack security measures that would typically thwart user input from tampering with data structures.

A classic illustration of a mass assignment scenario is demonstrated when an intruder successfully includes parameters during the user registration to elevate their account privileges from a primary user to an administrator. The user registration solicitation typically encompasses attributes like username, email address, and password.

However, a malicious entity could intercede with this request and introduce extra parameters such as "isadmin": "true" as you can see in 7. Should the underlying data structure contain a corresponding attribute and the API provider neglects to sanitize the infiltrator's input, an avenue potentially opens for the intruder to register their administrative account.

The other example of this would be "admin":"1", "admin":"true".

```
1        {
2            "name": "MassAssingment",
3            "email": "jack@email.com",
4            "isadmin": "true",
5            "password": "pasword123"
6        }
7
8
```

Listing 2.6.0.1: json Post request

The application is also used to PUT requests to change the user_name and age. The GET response includes the credit_balance property, which the attacker can replay to change the credit_balance [24].

```
1
2        Legitimate PUT /api/v1/users/me
3        {"user_name":"inons","age":24}
4        Response GET /api/v1/users/me
5        {"user_name":"inons","age":24,"credit_balance":10}
6        Attacker replay that with PUT
7        {"user_name":"attacker","age":60,"credit_balance":99999}
8
```

To prevent this, we can minimize reliance on functions that automatically connect client input to internal code variables or objects. Selectively permit updates to specific properties by whitelisting them and employ system tools to block client access to unauthorized properties. When relevant, precisely outline and reinforce frameworks for input data structures.

2.7 API VII: Security Misconfiguration

Security misconfiguration arises when adequate security reinforcement is absent across any segment of the app's structure or if cloud service permissions are wrongly configured.

The most recent security fixes are not in place, or the systems remains outdated. Unneeded functionalities are activated (such as specific HTTP methods). With the absence of Transport Layer Security (TLS). Clients aren't given security directives (like Security Headers). Lack of proper setup or presence of incorrect Cross-Origin Resource Sharing (CORS) policy. Error messages reveal stack traces, or sensitive details are disclosed.

An example would be when the attacker could gain access to the server's root and check the bash_history of a server. As a result, the attacker could see the commands run by the DevOps team. In addition, the attacker could find the new endpoint for the API server [24].

```
1
2       $ curl -X GET 'https://api.server/endpoint/' -H 'authorization: Basic
    ↪ Zm9vOmJhcg=='
3
```

To avoid these vulnerabilities, we need to establish a repeatable reinforcement procedure facilitating the swift and straightforward deployment of a securely fortified context. In addition, conduct regular evaluations and updates of settings across the entire API structure, encompassing orchestration files, API constituents, and cloud service permissions (e.g., S3 bucket). Introduce a protected communication conduit for all API interactions, enabling access to static resources like images. Employ an automated mechanism to consistently gauge the efficacy of configurations and parameters in all scenarios. Furthermore, if applicable, formulate and enforce comprehensive API response schemas, incorporating error responses, to prevent transmitting sensitive data to potential attackers. Ensure API access is confined solely to designated HTTP verbs, while other HTTP verbs should remain inactive (e.g., HEAD). APIs anticipating interaction from browser-based clients (e.g., WebApp frontend) should institute a sound Cross-Origin Resource Sharing (CORS) policy [24].

2.8 API VIII: Injection

The surreptitious insertion of unauthorized data into an API, commonly called an API injection attack, encompasses a range of techniques malicious actors use to manipulate the input fields of an API, aiming to compromise the system's integrity, gain unauthorized access, or extract sensitive data. This attack capitalizes on vulnerabilities within the API's design or implementation, allowing attackers to introduce malicious commands, queries, or code fragments. The result can lead to harmful consequences, such as unintended system behaviors, data leaks, or even complete system compromise.

API injection attacks come in various forms, each exploiting specific weaknesses in the targeted API:

1. SQL Injection: Attackers inject malicious SQL queries into API input fields, exploiting inadequately sanitized inputs to manipulate databases, retrieve sensitive data, or even modify data within the database.

2. Command Injection: Malicious commands are inserted into API inputs, often targeting operating system commands. This can lead to unauthorized execution of commands on the server, enabling attackers to gain control over the system.

3. XML Injection: Attackers insert malicious XML content into API input fields, potentially causing XML parsers to behave unintendedly, leading to information disclosure or system disruption.

4. XPath Injection: This type of attack targets APIs that use XPath expressions for querying XML data. Attackers manipulate input to gain unauthorized access to data or modify queries.

5. LDAP Injection: Exploiting APIs that interact with LDAP (Lightweight Directory Access Protocol) systems, attackers insert malicious input to manipulate LDAP queries and potentially access sensitive directory information.

6. NoSQL Injection: Similar to SQL injection, this attack targets NoSQL databases by injecting malicious queries that exploit weak input validation, potentially leading to unauthorized data access or manipulation.

Robust input validation, output encoding, and thorough input parameter sanitization are essential to mitigate the risk of API injection attacks. Employing parameterized queries, utilizing security libraries, and adhering to secure coding practices are crucial steps to fortify APIs against these insidious threats. Regular security audits, continuous monitoring, and staying informed about emerging attack vectors are fundamental to maintaining a robust defense against evolving API injection techniques.

2.9 API IX: Imporper Assets Management

This issue happens if the previous version of the API is still running and unpatched and does not have any plan on retiring the older API. The attacker will go to the older version to access the information which is supposed to not be able to be accessed. This issue can be seen when the developer fixes the rate limit for authentication, but attackers were able to downgrade and brute force the authentication. Also, if they integrated the services' inventory, their first or third company is outdated or missing, making the API vulnerable.

We must ensure that documentation is only available to authorized users to prevent improper asset management. In addition, steer clear of employing operational data in non-operational API deployments. If circumstances require it, ensure that these endpoints receive identical security measures as those designated for production. Also, use external safeguarding techniques like API security firewalls for all publicly accessible iterations of your APIs, not exclusively for the existing live edition.

2.10 API X: Insufficient Logging & Monitoring

Insufficient logging & monitoring happen when the server does not produce any logs, logging is not set correctly, or logging messages do not include enough details. In addition, the logs are not continuously monitored. Another important thing that needs to be considered is if the server does not check for integrity which could cause API VIII vulnerabilities. An example of this would be administrative APIs that were leaked on a public repository, and the repository owner was notified, but it took 48 hrs to respond. As a result, data was leaked, and because of sufficient logging, the company cannot assess what data was leaked to the attackers.

To overcome this issue, we must log all failed attempts, denied access, and input validation errors. Log management needs to be formatted correctly so that log management can consume the logs and should include enough details to identify malicious actors. In addition, implement a Security Information and Event Management (SIEM) framework to centralize and oversee logs from every facet of the API stack and hosting environments.

2.11 Comparision of OWASP API Top 10 2019-2023

At the highest level, as illustrated in Figure 2.11.0.1, two risks that were previously present have been removed, three have remained the same, four have been updated, and five new ones have emerged in 2023. The two formerly present risks, which were injection attacks and insufficient logging and monitoring, persist but are now categorized outside of the API top 10, as the growing use of APIs has pushed them beyond it. These two risks have been colored in red in Figure 2.11.0.1. Injection attacks are still a possibility, but the implementation of better firewalls and other mitigation techniques has reduced their risk. Three risks, namely Broken Object Level Authorization (BOLA), Broken Function Level Authorization (BAFLA), and Security Misconfiguration, remain unchanged. In Figure 2.11.0.1, the yellow color indicates that four risks were simply renamed. The five new risks, colored in blue, are Server Side Request Forgery (SSRF), Unsafe Consumption of APIs, Broken Object Property Level Authorization (BOPLA), Unrestricted Resource Consumption, and Unrestricted Access to Sensitive Business Flows.

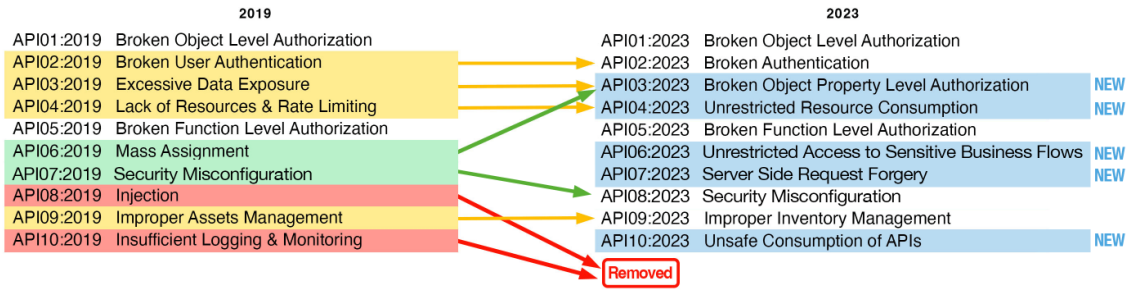


Figure 2.11.0.1: OWASP Top 10 API 2019 vs. 2023 [1]

2.12 Ethical Statement

Engaging in ethical hacking, alternatively termed penetration testing or adopting a white-hat hacking approach, entails examining computer systems, networks, or web applications to uncover security vulnerabilities susceptible to exploitation by nefarious hackers [41]. If you're interested in the field of ethical hacking, here are some guidelines to consider:

1. **Legal and Authorized Access:** Before any engagement with any system or network, we need explicit from the system's owner. In addition, it needs to be considered to work within the boundaries specified by the organizations.

2. **Safeguarding Sensitivity and Compliance:** Before and during ethical hacking endeavors, assess the sensitivity or confidentiality of the information at hand [41]. This is crucial to avoid any infringement of legal statutes, guidelines, or regulations when dealing with sensitive personal, financial, or proprietary data.

3. **Communication:** Maintain communication with the organization's stakeholders. Be clear about progress and findings.

4. **Informed Consent:** It is necessary to make sure the stakeholders have an understanding of the risk regarding penetration testing.

5. **Disclosure of Information:** Don't disclose the client information with any other parties. Make sure of the security of your client.

6. **Documentation:** It is important to document all the findings in detail and report them to the companies so they can follow your guidelines to secure their system.

2.12.1 Practical Hacking Practices:

Different resources can be used for ethical hacking practices, such as [42]:

- PortSwigger Web Security Academy
- Hack The Box
- PentesterLab
- VulnHub
- TryHackMe
- Google Gruyere
- Juice Shop
- WebGoat

The information provided here, including discussions on penetration testing and security tools, is strictly for educational purposes. Any actions taken based on this information are the sole responsibility of the individual. It is crucial to adhere to legal and ethical standards when engaging in activities related to penetration testing or cybersecurity exploration. Unauthorized access to computer systems is illegal, and individuals are strongly advised to obtain proper authorization before conducting any security testing. This serves as a reminder to act responsibly, ethically, and within

the bounds of the law, promoting the use of knowledge for positive contributions to the field of cybersecurity.

Chapter 3

API SYSTEM AND VULNERABILITY ENVIRONMENT SETUP

In pentesting, the most crucial step is setting up the required tools. This chapter will cover the essential tools for learning API pentesting for reconnaissance and authentication. I will conclude with the script I created, which can make life much easier and sets up your system with only one command. The operating system I built and tested this script with is Ubuntu 22.04.

3.1 Tools

In this section, I will provide a brief description of each pentesting tool for reconnaissance and authentication.

3.1.1 Burpsuite

Burp Suite is a highly regarded and widely embraced tool for testing web application security. It has gained significant popularity among ethical hackers due to its robust capabilities. By utilizing Burp Suite, ethical hackers can effectively detect vulnerabilities and possible security concerns in web applications. The suite encompasses various integrated tools, such as a proxy server, web spider, scanner, intruder, repeater, and sequencer, which work harmoniously to conduct a comprehensive security assessment. This powerful tool enables ethical hackers to simulate attacks, intercept and modify web traffic, analyze application behavior, and uncover potential vulnerabilities. Its adaptability and customizable features make it an essential asset

for evaluating the security strength of web applications. As a result, ethical hackers can identify weaknesses and provide valuable recommendations to enhance security without revealing the origin of their written content. In addition, this application has two versions: the community edition and the pro version. All my explanations about Burp Suite in the next chapter are done in the community edition.

3.1.2 Ffuf

Fuzz Faster you Fool (FFuf) is a web fuzzing tool widely used to identify hidden paths, directories, and files in web applications through brute-forcing and fuzzing techniques. It is built in the Go language, and its primary advantages include speed and efficiency compared to other tools. FFuf is capable of high-speed scanning and can process a large number of requests in a short time, making it ideal for ethical hackers to explore target applications for potential vulnerabilities that malicious actors could exploit. The tool also offers flexibility and customization, enabling ethical hackers to define their wordlists or use predefined ones to fuzz various aspects of a web application, such as URLs, parameters, or headers. This allows them to tailor their fuzzing efforts to the specific target, increasing the chances of identifying vulnerabilities. Moreover, FFuf supports various output formats, making it easier for ethical hackers to analyze and interpret the results.

3.1.3 Wfuzz

Wfuzz is a powerful tool that enhances web application assessments by seamlessly replacing the "FUZZ" keyword with customizable payloads [22]. These payloads consist of valuable data, enabling the injection of inputs into various aspects of an HTTP request, including parameters, authentication, forms, directories/files, and headers.

Wfuzz surpasses being a web content scanner as it excels in identifying and exploiting web application vulnerabilities. Its modular design fosters plugin development, while its language interface empowers users to conduct comprehensive manual and semi-automatic tests. This approach ensures a deep understanding of actions and context without relying solely on a web application scanner like Burp Suite.

3.1.4 FireFox Plugins

For our pentesting, there are a couple of plugins required for Firefox.

1. **Foxyproxy**

   To manage traffic routing to different ports, we use a Foxyproxy plugin. This is particularly useful when intercepting traffic using Burp Suite or conducting passive API reconnaissance through Postman.

2. **uBlock-origin**

   uBlock Origin is a highly efficient content blocker that provides a comprehensive solution to undesired ad content. By default, it effectively blocks ads, trackers, coin miners, popups, and more through a range of pre-enabled filter lists, including EasyList, EasyPrivacy, Peter Lowe's Ad server list, and the Online Malicious URL Blocklist. Furthermore, uBlock-origin offers a wide selection of additional filter lists like EasyList Cookie, Fanboy Annoyances, and AdGuard Annoyances, among others, for users to customize their browsing experience. Notably, it empowers users to block JavaScript selectively or globally, create personalized rules to override filter list entries, and access numerous advanced features. The best part is that uBlock Origin is free, open source, and operates under the public GPLv3 license, embodying a collaborative effort by users, for users [23].

3. **Wappalyzer**

   The Wappalyzer extension is a helpful web browser extension that identifies the technologies used by a website. It can detect the content management system (CMS), web frameworks, programming languages, analytic tools, and other components that comprise a website's stack. This tool is useful for reconnaissance as it allows us to analyze the technologies used by the target website. Ethical hackers can leverage this information to conduct exploit research and identify vulnerabilities or exploits associated with those technologies. It can also be used to craft targeted attacks on the website. Knowing about the website's technologies can help us choose the right tool for pen-testing.

3.1.5 Zaproxy

ZAP (Zed Attack Proxy) is an open-source web application security testing tool widely used to identify web application vulnerabilities. Ethical hackers and security professionals use ZAP for penetration testing, automated scanning, and vulnerability management. It helps simulate attacks, discover security weaknesses, and automate scans to detect common vulnerabilities. ZAP also assists in managing and tracking discovered vulnerabilities, raising security awareness among developers, and integrating security testing into the software development lifecycle. With its API security testing capabilities and extensibility, ZAP provides a versatile toolset for enhancing web application security.

3.1.6 Jwt_tool

Jwt_tool is a comprehensive solution, providing various features to enhance security and mitigate potential vulnerabilities. Among its capabilities, it can check the valid-

ity of a token, ensuring the authenticity and integrity of the information it carries. It also identifies known exploits, including the alg=none signature-bypass vulnerability, RS/HS256 public critical mismatch vulnerability, key injection vulnerability, blank password vulnerability, and null signature vulnerability. Its ability to scan for misconfigurations and weaknesses acts as a proactive safeguard, preventing potential security breaches. Additionally, the tool employs fuzzing techniques to provoke unexpected behaviors by manipulating claim values. It also enables testing secret/essential files, public keys, and JWKS keys for validity. Furthermore, it can identify weak keys through a high-speed dictionary attack, reinforcing the overall security posture.
Moreover, it offers the option to forge a new token header and payload content, creating a unique signature using the key or employing alternative attack methods. Timestamp tampering is another feature available, allowing for the examination and manipulation of timestamp data. The tool also facilitates RSA and ECDSA key generation and reconstruction, leveraging JWKS files. These are just some of the functionalities the logo embodies, and many more features remain to be discovered [25].

3.1.7 Postman

Postman is a widely utilized collaboration platform developers use to design, test, and document APIs (Application Programming Interfaces). Its intuitive interface enables seamless transmission of HTTP requests, API management, and response analysis. Ethical hackers leverage the capabilities of Postman to conduct comprehensive security assessments on APIs. By engaging in API penetration testing, they employ diverse request types (e.g., GET, POST, PUT, DELETE) to target API endpoints. Through meticulous scrutiny of API responses and their intricate composition, these professionals adeptly uncover potential vulnerabilities. They methodically scrutinize common security pitfalls, such as cross-site scripting (XSS), cross-site request forgery

(CSRF), and deficiencies in communication encryption. Moreover, Postman empowers ethical hackers with automated test script functionality, request collections, and the ability to manage environment variables. These features greatly enhance efficiency, enabling streamlined testing processes, time-saving endeavors, and effortless sharing of test cases among team members. Importantly, Postman boasts comprehensive support for various protocols and authentication mechanisms, rendering it an ideal tool for scrutinizing diverse API landscapes encompassing RESTful APIs, GraphQL, SOAP, and more.

3.1.8 Amass

Amass is an open-source tool that excels in network mapping and uncovering potential vulnerabilities. It uses various techniques like active reconnaissance and external asset discovery to collect extensive data. Amass is equipped with internal machinery and seamlessly integrates with external services, enhancing its efficiency and effectiveness.

This tool finds DNS, HTTP, and SSL/TLS data. It also works with APIs like the SecurityTrails API to expand its capabilities. Additionally, Amass taps into web archiving engines to uncover hidden online data. In addition, the framework saves the findings and metadata in a database (SQLite and PostgreSQL), which is updated and queried across the sessions. This database is in a graph-like structure. Figure 3.1.8.1 the assets show details about the findings, and the rows are connected to the assets table. Furthermore, all the data is time-stamped when created and last seen.

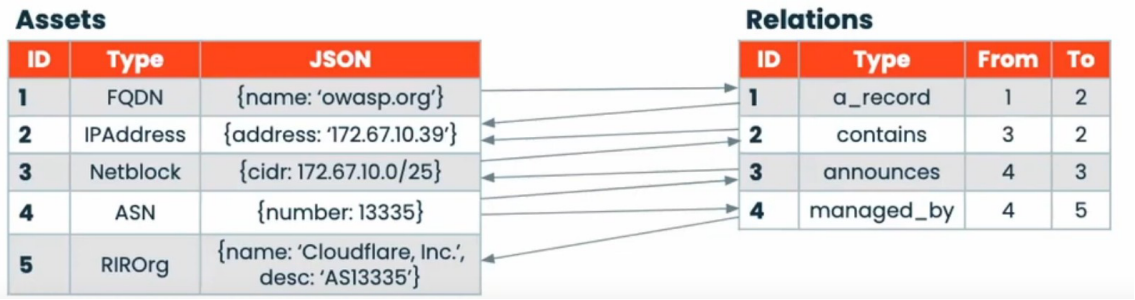


Figure 3.1.8.1: Under the hood of Amass database [3]

In network discovery, Amass stands out as a powerful and discreet tool, evading AI detection while delivering exceptional results.

3.1.9 Burp Suite vs Zaproxy

The Burp Suit provides both automated and semi-automated processes; however, the new update of Zaproxy gives us the capability to use the YAML file. As a result, Zaproxy will provide us with the capability of operating in container technologies such as Kubernetes and implementing CI/CD pipelines [44] [43]. In addition, when looking at the efficiency, product documentation, and community support of Burp Suite vs Zaproxy, it was discovered that both Burp Suite and Zaproxy have active support from online forums and other support functions. Likewise, both Burp Suite and Zaproxy come with well-documented manuals and product documentation. The one area where we see a difference is while both use a GUI interface, Zaproxy also includes a Command Line Interface (CLI) [47].

Furthermore, Burp Suite has more tools for vulnerability detection than ZAP. Burp Suite also comes with the essential tools needed for initiation, with the capability for add-ons if needed. Zaproxy, however, requires additional add-ons that must be installed first. These mandatory add-ons are necessary for proper functionality [48].

Alternatively, The Burp Suite has higher accuracy and fewer false positives than Zaproxy [43]. Zaproxy is excellent for finding vulnerabilities, but it has limitations in the scope of its scans. Lastly, Burp Suite is costly if you use the professional version; it costs $449 a year [49], but Zaproxy is free.

3.1.10 Ffuf vs Wfuzz

Table 3.1.10.1 compares three tools used in our pentesting. There are several tools available for content discovery, but the most commonly used ones for detecting vulnerabilities are Gobuster, Ffuf, and Wfuzz. Although these tools share the same purpose of finding directories, subdomains, and pentesting authentication, each of them has unique capabilities. Here is a comparison of the features of these three tools, which are widely used in the domain of vulnerability detection.

Table 3.1.10.1: ComparisonFfuf and Wfuzz

Feature	Ffuf	Wfuzz
Language	Golang	Python
Protocols	Primarily designed for HTTP	Supports multiple protocols (HTTP, FTP, etc.)
Purpose	Web content and directory fuzzing	Web application fuzzing
Performance	Optimized for speed and efficiency	Flexible and slower
Customization	Offers customization but streamlined	Highly customizable with various options
Wordlist Support	Efficient handling of large wordlists	Extensive payload generation capabilities
Ease of Use	Designed for simplicity	Powerful but may have a steeper learning curve

The first difference is the language used; while Ffuf is written in Golang, Wfuzz is based on Python [46]. The benefit of using Python over Golang is the ease of

installation for most users who may not be familiar with Golang. When it comes to fuzzing, the fuzzing speed is faster in the tools that use the Goland language, as the threading in Golang is much faster than the threading speed used in Python. The reason for this is due to Python only allows one thread to execute at a time and is helpful for I/O-bound tasks. However, Golang has an advantage because it supports concurrent programming using goroutines and channels. These goroutines allow both I/O-bound and CPU-bound tasks. In one article reviewed (cite) that focuses on the two fuzzing tools, Ffuz and Wfuzz, the author compared the two based on execution time, memory requirements, CPU utilization, and vulnerabilities identified. In this study, it was found that Wfuzz was the leading in every metric except for the memory footprint. However, in the conduction of this experiment, threading was not used. Based on our experiment in figure 3.1.10.1,3.1.10.2, we identified that this experiment can not be correct because Ffuf is faster because of the power of Golang. In Figure 3.1.10.2, the sys time plus user time indicates that the CPU time is more efficient than Wfuzz as seen in figure 3.1.10.1.

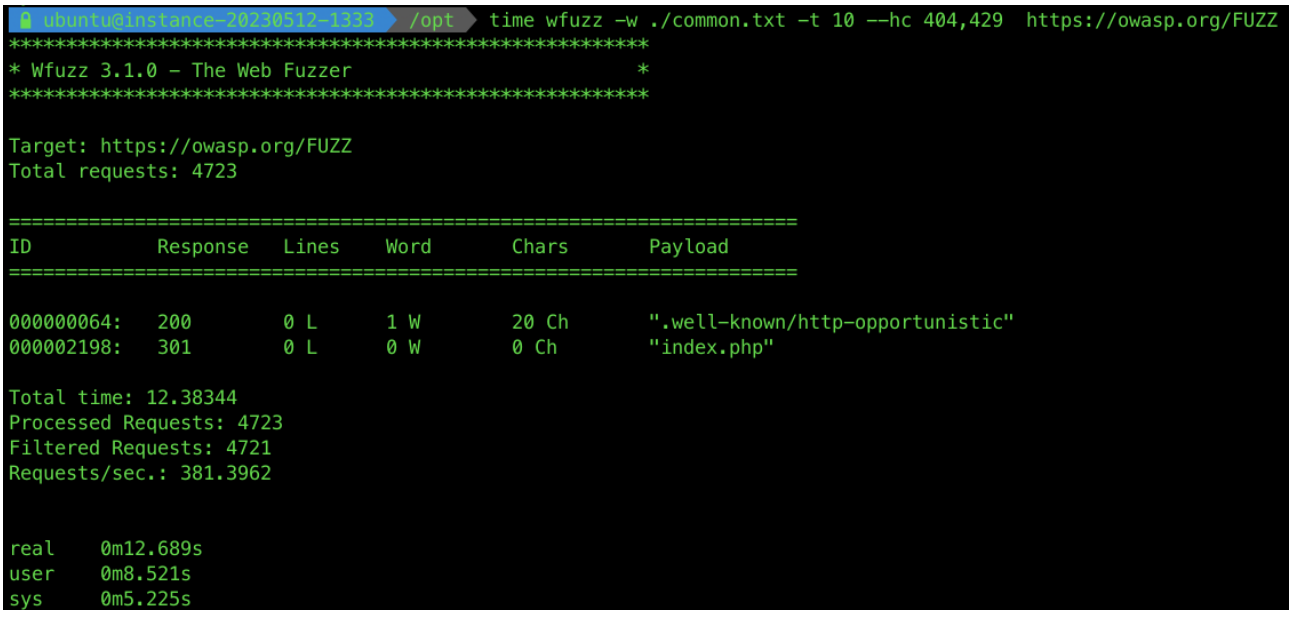


Figure 3.1.10.1: Wfuzz time performance

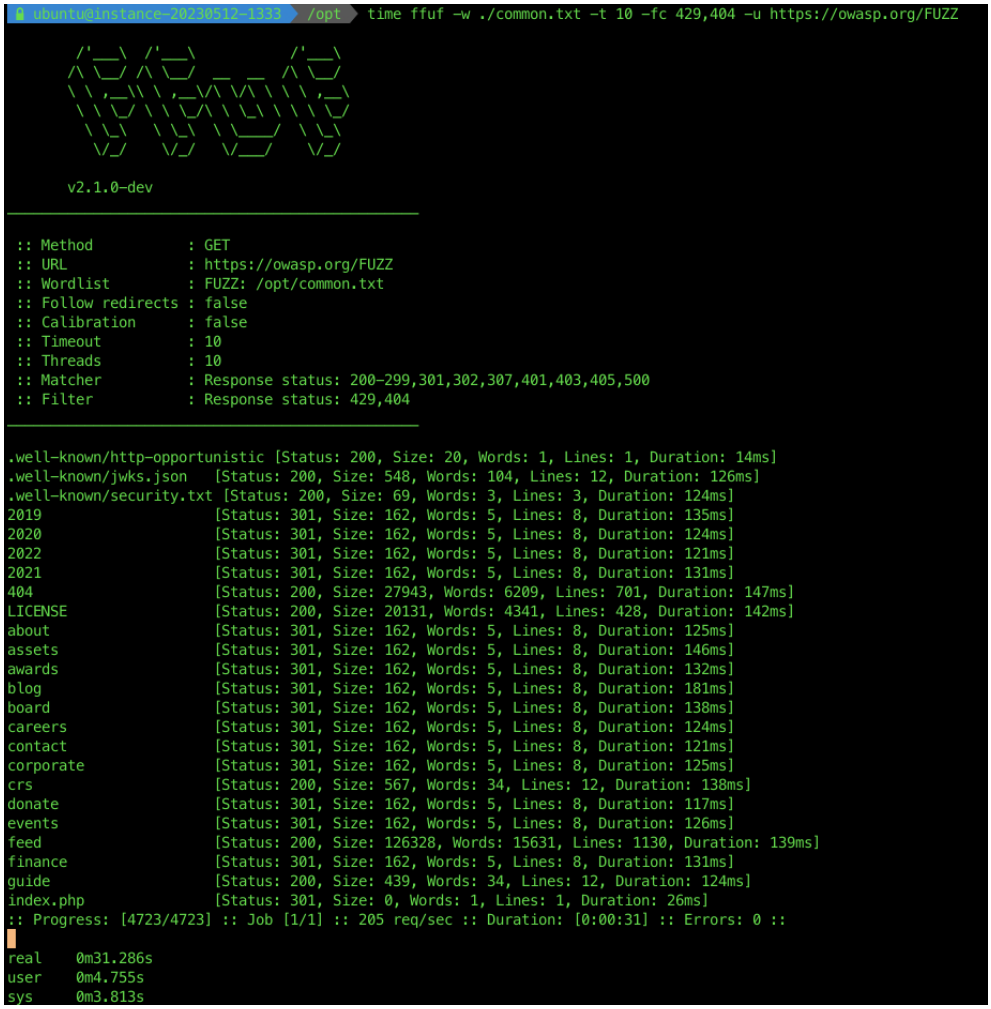


Figure 3.1.10.2: FFuf time performance

3.1.11 Rustscan vs Nmap

When Rustscan is used in conjunction with Nmap, it enhances the performance by asynchronously conducting port scans [45]. The Rustscan programming language, Rust, facilitates the execution of numerous tasks concurrently using a minimal number of operating system threads. While Rustscan uses Rust, Nmap executes in C language as seen in table 3.1.11.1. Rustscan provides caching, which can drastically reduce scanning time on the extensive network [45]. On its own, Nmap speed is powerful

but often perceived as being slower 3.1.11.1. The reason is Nmap uses detection evasion of firewalls and intrusion detection systems (IDS). However, Nmap uses flags to speed up scans with the -sS, and -T flags. The -sS can be performed quickly with the capability to scan thousands of ports per second. This is known as the stealthy scan method. The -T provides a range of speed capabilities from very slow using the flag -T0, to highly aggressive using the flag -T5.

Being faster, as in the case of Rustscan, doesn't necessarily mean better. Newer firewalls can easily detect fast scans and block IPs scanning the network. Therefore, we can limit detection by firewalls and IDS devices using Nmap and slower scanning methods.

Table 3.1.11.1: Rustscan vs Nmap

Feature	RustScan	Nmap
Language	Rust	C
Speed	Fast and efficient but noisy	Powerful but may be perceived as slower
Scanning Techniques	Simple and fast, focused on speed	Versatile with a wide range of techniques
Ease of Use	User-friendly command-line interface	Extensive options, can be more complex
Community Support	Growing community	Large and active community, extensive resources
Flexibility	Basic scans, quick reconnaissance	Comprehensive feature set, suitable for various scenarios

In addition, Rustscan doesn't use more than one thread and doesn't call more than one Nmap instance at a time [50]. Rustscan provides scans for both host and IP asynchronously. The benefit of this is a significant boost to the performance compared to scanners that only scan hosts synchronously. The purpose of Rustscan is for port scanning, and it does not support other features like Nmap. Nmap supports various tasks, such as OS detection, service version detection, and vulnerability scanning. Rustscan gives an option by using "–" to run flags provided for Nmap. Rustscan can

have 65000 ports in 3 seconds; however, it's essential to remember that this will be noisy for all intrusion detection and prevention [50].

3.2 Automatic Install Script

To streamline the installation process, the initial step is to download the Ubuntu ISO image from https://ubuntu.com/download/desktop. Once the download is complete, there are two available installation options. We can utilize virtual machines like VMware or Virtualbox or install it as a dual-boot system. After installation of the Ubuntu operating system, open the terminal and run the command below:

```
1        sudo apt-get install git ansible
2
```

We need git to clone the API-lab GitHub repository, and Ansible to configure our system automatically. The script provided requires Ansible. The next step is to run the following command to clone the repository.

```
1        git clone git@github.com:Hadiasemi/API-lab-Setup.git
2
```

3.2.1 Ansible

Ansible is a free and open-source tool written in Python. Ansible can configure systems, deploy software, and orchestrate advanced workflows to support deployment, system updates, and more [26].

The primary goal of Ansible is simplicity and ease of use. In addition, it has essential consideration of security and reliability. For deployment, it uses OpenSSH for transport and human-readable language to configure [26].

In this project, the purpose of using Ansible is users do not need to configure everything manually. Another critical thing to remember is we can use this script to deploy to multiple servers so professors can implement this in their classrooms. This will be an excellent opportunity for future developers to have the skills to develop API applications securely.

The configuration of the ansible-playbook (set of instructions we need to configure for deployment) is in YAML format, and files end with .yml. The directory structure looks like the figure 3.2.1.1. The **local.yml** is the main YAMAL file, which includes all the playbooks for our setup we want to run as tasks.

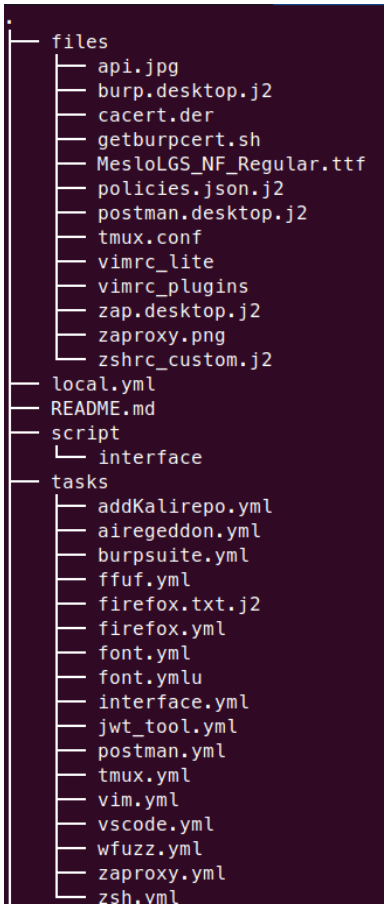


Figure 3.2.1.1: Ansible folder tree structure

In local.yml, we set up all the hosts we want to run this script on. Here, we used a local host. However, we can have a separate file in Ansible to define all host IPs. In this way, we can deploy our configuration to multiple servers. As we can see in 3.2.1.1, these are all the hosts we define in our hosts' file and give root permission to run our config as a sudo user. Vars is the variable that we predefined that we use later in our code. Tasks are the steps in which we want our project to configure our system. In my local.yml, instead of all configurations in the same place, I used include, which refers to the task folder, as you can see in 3.2.1.1. The file consists of the configuration, images, .tmux.conf, .zshrc, and other configurations.

```
1
2           ---
3           - hosts: all
4             become: true
5             vars:
6               created_username: api
7
8             tasks:
9               - name: Install curl
10                apt:
11                  name: curl
12                  state: latest
13                  update_cache: true
14
```

Listing 3.2.1.1: Sample example of Ansible playbook

As you can see in the files folder, some files end in j2, which stands for Jinja. Ansible uses Jinja to set up its configuration.

3.2.2 Jinja

Jinja is an Ansible and Python template engine for generating dynamic content like HTML, JSON, and text-based formats. It separates templates containing static text and template expressions enclosed in , allowing data to be inserted or logic to be processed when rendered. With variables, filters, and control structures, Jinja efficiently combines data and templates, promoting a maintainable code. Commonly employed in web frameworks like Flask and Django, Jinja facilitates tasks such as generating web pages with personalized content or crafting configuration files and emails. Its security features include automatic content escaping and guarding against potential vulnerabilities.

After going to API-lab-Setup, we must run the following command in the terminal to set up our system.

```
1         ansible-playbook local.yml -K
2
```

The **-K** prompts us to provide a sudo password because some configurations require sudo access to be installed on our system. Another configuration that can be set up in our ansible-playbook commands is providing the tags we want to run without running the whole configuration. For example, we can run

```
1         ansible-playbook local.yml -K --tags "setup"
2
```

By running the setup, we only install the based configuration, which includes installation-based setup, without installing any cyber security tools. In addition, we can provide multiple tags like ”setup, burp.” This installs setup configuration and installs Burp Suite. After installation, we must go to the terminal, right click, and choose the

preferences options. In the opened window from the menu tab, select the text and hit the checkmark on Custom font. Next, choose the MesloLGS NF from the font options so that our prompt resembles the figure 3.2.2.1.

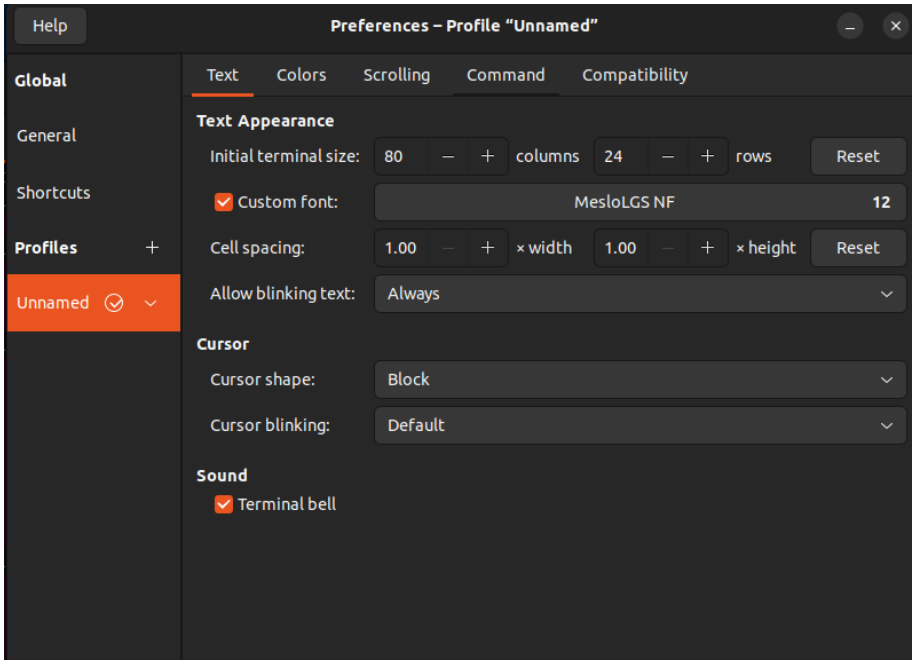


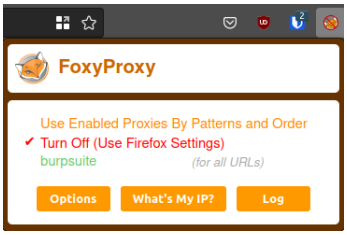
Figure 3.2.2.1: Choose the right terminal font

After completing the setup of your system, your desktop should look like figure 3.2.2.2.

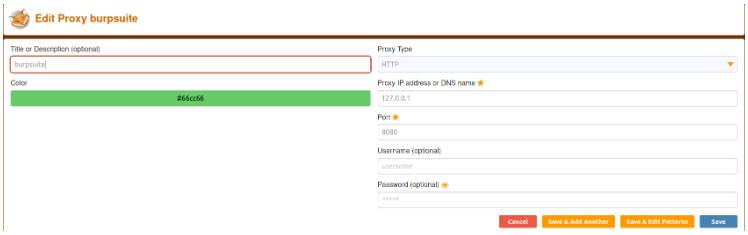
Figure 3.2.2.2: Ubuntu desktop after configuration

3.2.3 Configure Foxyproxy

To set up Foxyproxy, open Firefox and click on the plugin as per 3.2.3.1a, select "options" to open the plugin, click on the "Add" button to configure your proxy settings. Set up your proxy as shown in figure 3.2.3.1b and save the changes.



(a) Foxyproxy plugin



(b) Foxyproxy configuration set up

Figure 3.2.3.1: Foxyproxy Configuration

For both Postman and Burp Suite, the proxy IP is **127.0.0.1**. In addition, the port for Postman is **5555**, and the Burp Suite is **8080**. After configuration, it should have two proxies in the menu.

3.2.4 Burp Suite Extentions

After opening the Burp Suite and accepting the configuration, go to the extensions tab and click on BApp Store. Click on the search bar and look for **Authorize**. Figure 3.2.4.1 indicates that we need Jyton to use this library.

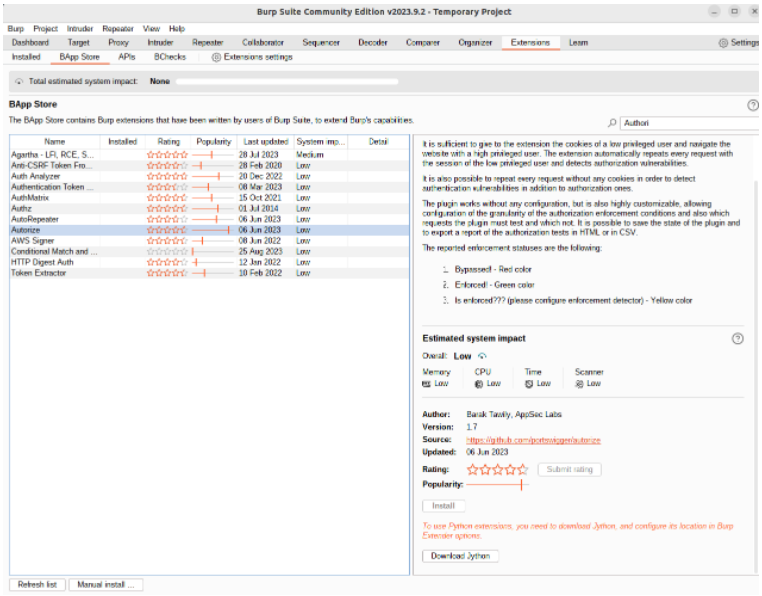


Figure 3.2.4.1: Burp Suite Authorize Plugin

Click on the "Download Jython" to open the Jython website. Please download the standalone version of this library. Then, from the menu, choose "Extension settings". In the Python environment, click on select file and choose the directory downloaded file. An example of this configuration is in the figure 3.2.4.2.

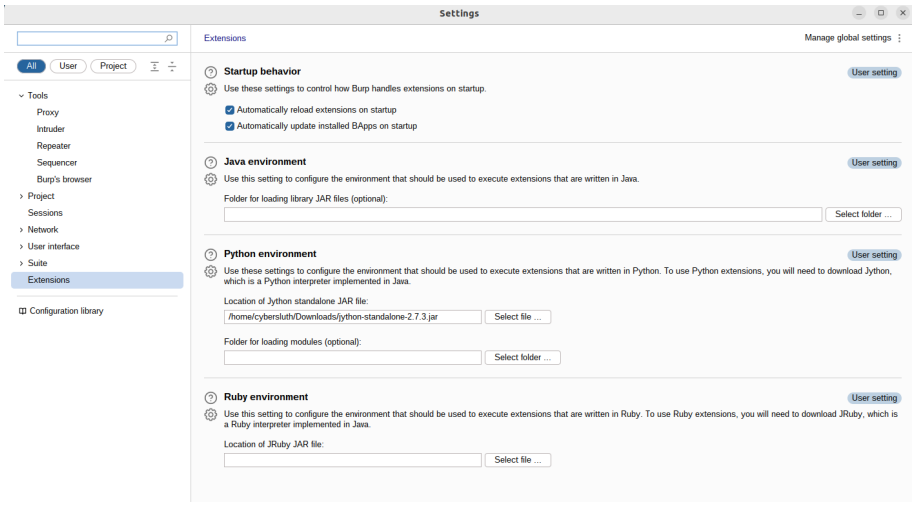


Figure 3.2.4.2: Setup Jython

Close the menu, return to the App, and search the Authorize app again. The download option should be available. Click download; it is crucial to install the **Param Miner** extension. This extension is designed to identify hidden and unlinked parameters and is particularly useful in detecting vulnerabilities related to web cache poisoning. The extension is equipped with sophisticated differential logic inherited from the Backslash Powered Scanner, along with a binary search method, which enables it to speculate on up to 65,000 parameter names in a single request. The extension draws these parameter names from a carefully curated internal wordlist, which is further supplemented with words harvested from all relevant incoming traffic.

3.2.5 Limitation of Script

This script was built to run on Ubuntu 22.04. The script is mainly constructed to get the application from a repository or website. One of the caveats that cause issues is the version of Golang installation in golang.yml in the tasks folder is hard coded, which in figure 3.2.5.1 indicates that we use version 1.21. If any dependencies that

later require the higher version, it must manually be changed to the version required for the update.

```
- name: Add Golang repository
  become: true
  apt_repository:
    repo: ppa:longsleep/golang-backports
    state: present
    update_cache: yes

- name: Install specific version of Golang
  become: true
  apt:
    name: golang-1.21
    state: present
```

Figure 3.2.5.1: Golang installation

Another limitation after running the script, the user will need to go to Firefox to configure the FoxyProxy plugin manually.

Additionally, you have to go to Burp Suite to download the plugins.

Chapter 4

API SECURITY TOOLS TESTING: RECONNAISANCE AND AUTHENTICATION

In this chapter, we will delve into an exploration of open-source tools designed for API security, specifically focusing on reconnaissance and authentication aspects. Within the realm of reconnaissance, we introduce both passive and active methodologies. Authentication is scrutinized, encompassing discussions on password-based authentication attacks, OTP authentication attacks, and JWT token attacks.

To facilitate practical testing in a controlled environment, the crapi.apisec.ai [51] serves as the designated platform for evaluating the tools under consideration. It is important to note that CrAPI is a deliberately vulnerable system expressly authorized for penetration testing purposes.

4.1 API Reconnaissance

Reconnaissance constitutes the initial phase of ethical hacking, encompassing the systematic acquisition of information about the target system [37]. This data compilation extends from details concerning network infrastructure to particulars about employee contacts. The overarching objective of the reconnaissance phase is to discern and catalog a comprehensive array of potential attack vectors.

This section, as delineated in table 4.1.0.1, comprehensively addresses both passive and active reconnaissance methodologies. Passive reconnaissance involves a meticulous examination of websites that serve as instrumental tools for information gath-

ering. If tools are identified on GitHub that automate the process of extracting information from this website, such tools will be presented in section. In active reconnisance cover tools for actively gathering information. Table 4.1.0.1 indicated two types of reconnisance and the material will be covered in this section.

Table 4.1.0.1: Reconnaissance Approaches

	Passive Approach	**Active Approach**
Reconnaissance	Google Dorking	Port Scanning (Nmap, Rustscan)
	Git Dorking (Github-Dorks, TruffleHog, Gitrob)	Burp Suite
	Wayback Machine (waybackurls)	Zaproxy
	Shodan	Gobuster
	Amass	

4.1.1 Passive

Passive reconnaissance is obtaining information about targeted computer systems and networks without actively engaging with them without interruption [21]. It relies on data that can be publicly accessed through open-source intelligence (OSINT) research on the target infrastructure to identify vulnerabilities or potential attack vectors [38]. The primary goal of this approach is to collect as much information on competitors, industry trends, and consumer behavior as possible about the target without direct interaction [38]. In addition, credentials such as usernames and credentials can help to be used as authenticated users, which helps to do better in reconnaissance. Versioning and API documentation can also help understand how to interact with API. If we can use older APIs, maybe access some information that cannot be accessed through new patches of vulnerabilities. The following approaches are introduced: Google Dorking, Git Dorking, Wayback Machine, Shodan, and Amass. In addition, it provides different tools with respect to each approach if it exists.

4.1.1.1 Google Dorking

Google Dorking, alternatively known as Google hacking, denotes the utilization of specialized Google search techniques to exploit vulnerabilities in websites or to retrieve information not readily accessible through public search results [39].

Google Dorking is a technique that involves using specific search queries on the Google website. These queries are listed in Table 4.1.1.1, which displays the most common Google Dorking queries to use. The essential keywords used in the table are **inurl, intitle, intext,** and **ext**.

When you use the **inurl** keyword, Google will display URLs that include the keywords you were searching for. Similarly, **intitle** and **intext** will look for keywords in the title or text of the page, respectively. The **ext** keyword searches for files with the specific extensions you provide.

Table 4.1.1.1: Google Dorking Queries [1]

Query	Expected Results
inurl:"/wp-json/wp/v2/users"	Find publicly available WordPress API user directories
intitle:"index.of" intext:"api.txt"	Finds publicly available API key files.
inurl:"/api/v1" intext:"index of /"	Finds potentially interesting API directories.
ext:php inurl:"api.php?action="	Finds all sites with a XenAPI SQL injection vulnerability. (This query was posted in 2016; four years later, there are currently 141,000 results.)
intitle:"index of" api_key OR "api key" OR apiKey -pool	It lists potentially exposed API keys.
inurl:"/swagger-ui/"	Discover websites that have exposed their API documentation through the Swagger UI interface.
inurl:"/api-docs"	This can help to find API documentation

Overall, Google Dorking is a powerful tool that can be used for both good and bad purposes. Therefore, it is essential to use it responsibly and ethically.

4.1.1.2 Git Dorking

The search functionality within Github represents a robust and valuable tool for the exploration of repositories in pursuit of sensitive information. The assembly of Github dorks, or specialized search queries, can potentially expose confidential details,

encompassing personal and organizational data, including but not limited to private keys, credentials, and authentication tokens [27].

Numerous companies have relied on the GitHub repository for their development needs for several years. Despite having several competitors, GitHub continues to be the most popular platform for software development. While many developers upload modifications to GitHub, they may unintentionally push critical information such as environment variables, token keys, usernames, and passwords during stressful moments. Figure 4.1.1.1 indicated that if you search for "nasa.gov credentials" in GitHub, you will see that one of the software developers, by mistake, pushed the code to GitHub with the credentials. As a result, malicious actors exploit the possibility of such occurrences by using Git Dorking to search for sensitive data that may have been inadvertently included in the source code uploaded to GitHub.

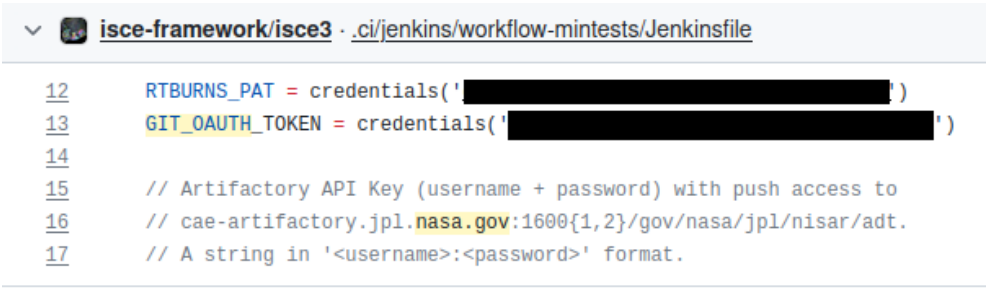


Figure 4.1.1.1: NASA GIT-OAUTH pushed to GitHub

Table 4.1.1.2 demonstrated the common Git Dorking command, which is helpful such as finding files, finding extensions, finding paths, languages, credentials, etc.

In the following, I will discuss tools that automate the process instead of doing it manually.

Table 4.1.1.2: Git Dork keys

GitHub Dork	Info
filename: travis.yml	finding files
extension: json	finding the extension like json
path: sites databases password	find the path
language: python	finding languages
api_key "api key" authorization_bearer oauth auth authentication client_id password user_password user_pass passcode client_secret secret password hash OTP user auth	Finding API keys, Tokens, and passwords
user:name (user:admin) org:name (org:google type:users) in:login (<username>in:login) in:name (<username>in:name) fullname:firstname lastname (fullname:<name><surname>) in:email (data in:email)	Finding usernames
created:<2012–04–05 created:>=2011–06–12 created:2016–02–07 location:iceland created:2011–04–06..2013–01–14 <user>in:username	Finding information-specific dates

1. **Github-Dorks: [27]**

   Github-Dorks is a powerful and useful tool that enables us to search for sensitive data in repositories like private keys, credentials, authentication tokens, etc. This repository is written in Python [27]. The good part is that users can give customized GitHub dork files or use the default GitHub-dorks.txt.

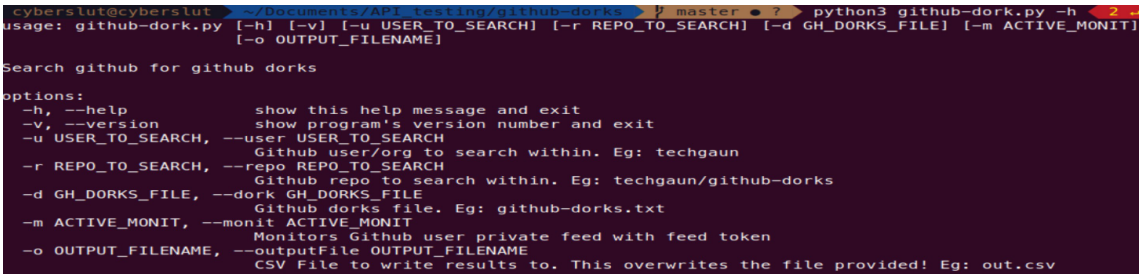


Figure 4.1.1.2: Github-dorks usage

Figure 4.1.1.2 indicate many functionality in which we can search for the user, Github repo, provide a Github Dork file, or monitor the user's private feed and output the result in the CSV file.

2. **TruffleHog: [28]**

   This tool is excellent for finding leaked credentials in a git repository. This program was written in Golang [28] and gave multiple ways to install this application [28].

   With TruffleHog, you could scan a repo for only verified secrets, scan a GitHub organization for only verified secrets, scan a repo for verified keys, and get JSON output. Below is examples which we can use for pentesting. The –**org** specified the organization we were looking for leakage of credentials.

```
1                       # check for specified repository
2                       trufflehog git https://github.com/trufflesecurity/
    ↪ test_keys --only-verified
3                       # Scan a GitHub Organization
4                       trufflehog github --org=trufflesecurity --only-
    ↪ verified
5
```

3. **Gitrob: [29]**

   Gitrob is designed to assist in locating possibly confidential files uploaded to

public repositories on GitHub. It works by cloning a user or organization's repositories to a depth that can be adjusted and then reviewing their commit history to identify and highlight files that exhibit characteristics associated with potentially sensitive data. These results are conveniently displayed via a web interface, making navigating and analyzing them effortless. This code was developed with Golang. This library has not been updated in the last five years and has been archived by the owner on Jan 13, 2023.

4.1.1.3 Wayback Machine:

The Internet Archive Wayback Machine is an Internet-based digital repository with extensive web pages and multimedia files collected from the web over the last two decades. It was created by the Internet Archive, a non-profit entity committed to safeguarding digital content and ensuring its availability to the general public [31]. The Wayback Machine serves as a virtual time-travel tool for the internet. It permits users to retrieve past iterations of websites and web pages precisely as they were at distinct moments in history. The Wayback Machine curls the internet and takes periodic snapshots of web pages.

The benefits of using Wayback Machine are for five reasons [31]:

1. **Historical data:** The Wayback Machine offers access to past website data, enabling penetration testers to observe its evolutionary journey. This aids in detecting alterations that might have introduced fresh security flaws or vulnerabilities.

2. **Comparative Examination:** When scrutinizing intricate websites or web applications, pentesters can utilize the Wayback Machine to compare various

website versions across timeframes. This process helps pinpoint modifications that might have introduced novel security weaknesses.

3. **Unearthing Confidential Information:** The Wayback Machine can unveil sensitive data that may have existed on a website previously, such as login credentials, credit card details, or other private information.

4. **Spotting Outdated Software:** By delving into the Wayback Machine, outdated software or plugins that were once present on a website can be identified. These antiquated technologies may harbor known vulnerabilities that malicious actors can exploit.

5. **Confirmation of Remediation Efforts:** Pentesters can employ the Wayback Machine to validate whether website owners have effectively addressed previously identified vulnerabilities. This ensures that security concerns have been adequately resolved.

The setps required for using Wayback Machine:

1. Go to archive.org/web.

2. Once there, navigate to the homepage, which should have a search bar.

3. In the search bar, enter the website for which you want to gather information.

Figure 4.1.1.3 shows that for owasp.org. The calendar allows us to find the information between the different periods and the changes that happened to the website.

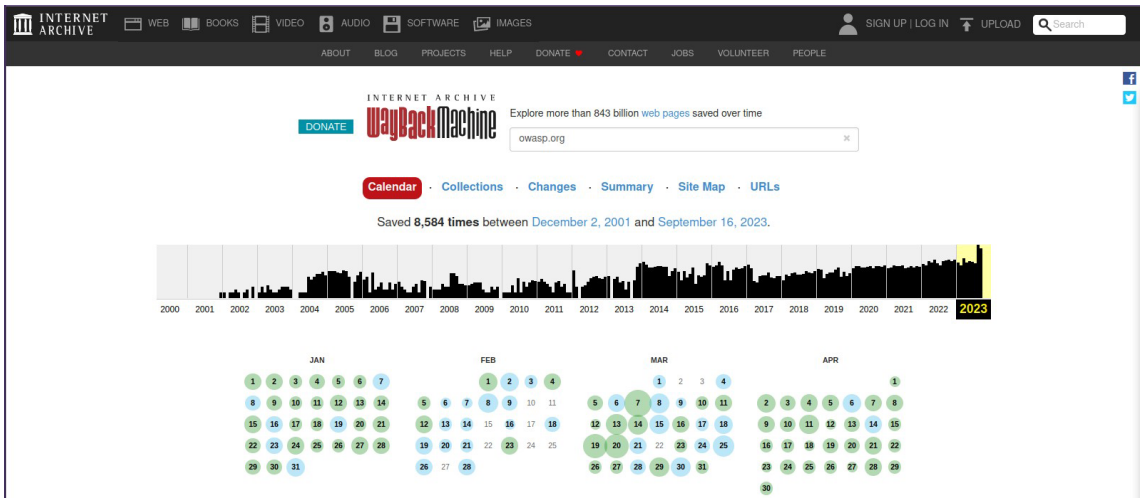


Figure 4.1.1.3: Wayback Machine for owasp.org

The changes section button shows calender for each year and by clicking on each squres shows the snapshot of the website content. By clicking on the URLs, we were find all links for specified website.

Wayback Machine offers exclude your website by emailing info@archive.orge and indicate the URLs, the time period, the period you have control on the website to remove your information [40].

Waybakurls:

The waybackurls tool can grab all URLs found by Wayback Machine. This was fully developed by Golang languages. Below is an example of this tool [32].

```
$ cat domains.txt | waybackurls > urls.txt
```

Once the command displayed above is entered into the terminal, a urls.txt file will be returned. The file will include all the domains from which we gather information and the waybackurls read from the output of the domain.txt

4.1.1.4 Shodan:

Shodan is also an excellent tool typically used in reconnaissance. This tool allows the user to search for key words to find information about different IPs and open ports on those IPs. Additionally, Shodan can be used to gather information about a particular API. Shodan has regularly monitored devices accessible to the internet and checked all the IPv4 address spaces with open ports. For API pentesting, we can search in the search bar **"content-type: application/json"** to only display the APIs that return JSON or change JSON to XML format to return XML formats. Figure 4.1.1.4, shows 5,365,719 results based on our search. Also, Shodan enables us to filter from the left side menu based on countries, organizations, and operating systems to get more related results to our needs. In addition, to find WordPress API, we can search for **"wp-json"**.

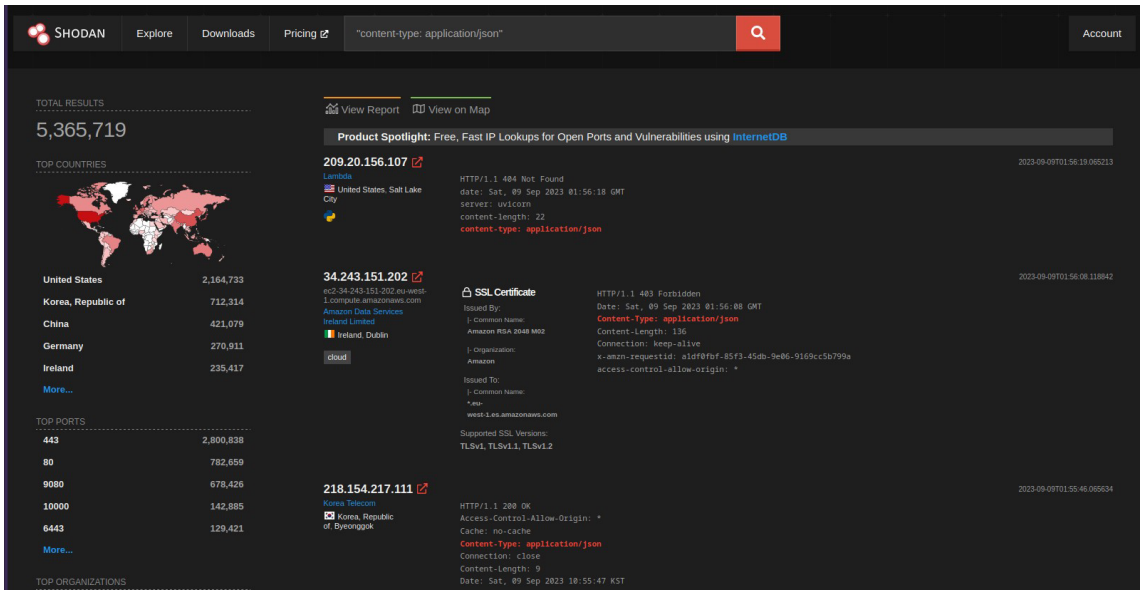


Figure 4.1.1.4: Search Shodan for "content-type: application/json"

4.1.1.5 Amass:

Amass version 4.2 has two subcommands, intel, and enum. The following goes into detail on each of them.

1. **Intel:** Intel command is used for collecting open source intelligence on the organization and allows finding the root domain names associated with the organization. When you run an Amass intel command, the command provides a help menu on how to use the intel sub-command (see figure 4.1.1.5). After the **intel** command, we could choose the optional flags and specify the domain we will be pentesting. For example, figure 4.1.1.6 shows that OWASP owned the blockster.com and modeltime.

```
Usage: amass intel [options] [-whois -d DOMAIN] [-addr ADDR -asn ASN -cidr CIDR]

  -active
        Attempt certificate name grabs
  -addr value
        IPs and ranges (192.168.1.1-254) separated by commas
  -asn value
        ASNs separated by commas (can be used multiple times)
  -cidr value
        CIDRs separated by commas (can be used multiple times)
  -config string
        Path to the YAML configuration file. Additional details below
  -d value
        Domain names separated by commas (can be used multiple times)
  -demo
        Censor output to make it suitable for demonstrations
  -df value
        Path to a file providing root domain names
  -dir string
        Path to the directory containing the output files
```

Figure 4.1.1.5: Amass intel subcommand

```
cybersluth@cybersluth ~ amass intel -d owasp.org -whois
blockster.com
modeltime.com
```

Figure 4.1.1.6: OWASP root domain with using Amasss

When using Amass intel for search operations, you have the flexibility to enhance its functionality by incorporating additional configuration choices, such as the "-active" parameter, which initiates zone transfers and actively scans to retrieve SSL/TLS certificates for data extraction. It's crucial to emphasize the importance of obtaining proper authorization before conducting active searches on the designated target [2].

In addition, with "-org" we can look for organizational names that return Autonomous System Network (ASN) IDs assigned to the target.

2. Amass **enum** enables you to engage in DNS enumeration and mapping of the target, enabling the identification of an organization's vulnerable attack surface. The results of this enumeration are stored in a database displayed in a graph either in Amass' default output location (∼/.config/amass) or the designated output directory using the "-dir" flag. The same arrangement applies to other Amass subcommands. Two attacks can be achieved with an enum subcommand: -passive and -active. The passive method is faster; however, Amass will not check for DNS information, such as resolving the subdomains. The command below displays the output of the Amass passive attack on the owasp.org.

```
1
2                       $ amass enum -passive -d owasp.org
3                       [...]
4                       update-wiki.owasp.org
5                       [...]
6                       my.owasp.org
7                       www.lists.owasp.org
8                       www.ocms.owasp.org
9                       [...]
10
```

Another option is the active configuration mode. This method will give us more precise results, and more assets could be discovered because all DNS enumeration techniques will be enabled.

The command below is active and performs subdomain brute-forcing with the displayed word list. In addition, this will be validated by Amass default or the specified resolver. The below command -d specifies the domain, and -w specifies the wordlist we want to brute-force and saves the output results at amass_results_owasp.txt.

```
1
2                    $ amass enum -active -d owasp.org -brute -w ./
   ↪ deepmagic.com-top50kprefixes.txt -dir amass4owasp -config /root/amass/
   ↪ config.yaml -o amass_results_owasp.txt
3
```

For the -config, you can specify special configs which are located in our Ansible script at ".config/amass/config.yml".

In addition, we have other tools for Amass, which are installed on our system from https://github.com/owasp-amass/oam-tools.git. The oam-tool provides four different tools. Each of these tools includes different flags to run with default and non-default commands that are common in all of them. Table 4.1.1.3 include the default flags and the four tools included as oam_i2y, oam_subs, oam_track, and oam_viz.

Table 4.1.1.3: Oam-tools default flags [2]

Flag	Description	Example
-h/-help	Show the program usage message	oam_command -h
-config	Path to the YAML configuration file	oam_command -config config.yaml
-dir	Path to the directory containing the graph database	oam_command -dir PATH -d example.com
-nocolor	Disable colorized output	oam_command -nocolor -d example.com
-silent	Disable all output during execution	oam_command -silent -d example.com

1. The **oam_i2y** is for converting legacy ini configuration formats to a new YAML format. The flag provided for this tool is -ini, which is indicative of the ini file we want to convert to. The -cf and -df are for config.yml and dastasrc.yml, which we used for our configuration. Below is an example of this command.

```
1
2                   oam_i2y -ini config.ini -cf ../../config.yaml -df
    ↪ datasrc.yaml
3
```

2. The **oam_subs** is for analyzing collected Amass assets. Oam_subs conducts actions related to the graph database, allowing users to view and control it. This operation can utilize the SQLite file generated through enumerations or the remote graph database settings specified in the configuration file [2].

Table 4.1.1.4: Oam_subs flags [2]

Flag	Description	Example
-d	Domain names separated by commas (can be used multiple times)	oam_subs -d example.com
-demo	Censor output to make it suitable for demonstrations	oam_subs -demo -d example.com
-df	Path to a file providing root domain names	oam_subs -df domains.txt
-ip	Show the IP addresses for discovered names	oam_subs -show -ip -d example.com
-ipv4	Show the IPv4 addresses for discovered names	oam_subs -show -ipv4 -d example.com
-ipv6	Show the IPv6 addresses for discovered names	oam_subs -show -ipv6 -d example.com
-names	Print just discovered names	oam_subs -names -d example.com
-o	Path to the text output file	oam_subs -names -o out.txt -d example.com
-show	Print the results for the enumeration index + domains provided	oam_subs -show
-src	Print data sources for the discovered names	oam_subs -show -src -d example.com
-summary	Print just ASN table summary	oam_subs -summary -d example.com

3. The **oam_track** command checks the difference between enumerations that include the same target(s) for monitoring the target surface. The flags for this tool are -d, where we define the domain, and -df, the path to the file providing the root domain. Lastly, -since is for excluding all enumerations before a spec-

ified date (format: 01/02 15:04:05 2006 MST) [2].

4. The **oam_viz** generates illuminating network graph visualizations that enhance the organization of collected information. This operation can utilize the SQLite file generated through enumerations or the remote graph database settings specified in the configuration file.

   The following displays options to display DNS and infrastructure findings in the form of a network graph:

Table 4.1.1.5: The oam_viz flag for showing network graph

Flag	Description	Example
-d	Domain names separated by commas (can be used multiple times)	oam_viz -d3 -d example.com
-d3	Output a D3.js v4 force simulation HTML file	oam_viz -d3 -d example.com
-df	Path to a file providing root domain names	oam_viz -d3 -df domains.txt
-dot	Generate the DOT output file	oam_viz -dot -d example.com
-gexf	Output to Graph Exchange XML Format (GEXF)	oam_viz -gexf -d example.com
-o	Path to a pre-existing directory that will hold output files	oam_viz -d3 -o OUTPATH -d example.com
-oA	Prefix used for naming all output files	oam_viz -d3 -oA example -d example.com

4.1.2 Active

In active reconnaissance, the attacker directly interacts with the target and gathers information for later stages of attacks without disruption or noticeable impact [30]. It's like softly knocking on a system's digital doors and windows, discreetly seeking to understand its vulnerabilities and weaknesses. This subtle approach leaves minimal traces to avoid raising alarms or suspicions. By employing active reconnaissance techniques effectively, security professionals can gain valuable insights into a system's configuration and potential entry points while maintaining a low profile to ensure undetected operation.

4.1.2.1 Port Scanning

The initial step of each scanning is port scanning, which enables us to gather more information about the ports in our client server. We do not know what type of applications are running in the system. I will review the most common open-source tools as Nmap and Rustscan for port scanning.

1. **Nmap**

   Nmap is widely used for port scanning. It offers a variety of flags to suit different needs but can be quite disruptive, so be sure to obtain permission before use on your target. See Table 4.1.2.1 for a breakdown of the various flags, a description of those flags, and their respective functions.

Table 4.1.2.1: Nmap flags and use cases

Command	Description	Use Case
nmap target	Basic host discovery and open port scanning	Discover hosts and identify open ports
nmap -F target	Fast scan (top 100 ports)	Quickly identify common open ports
nmap -p 1-65535 target	Scan all 65,535 ports	Thoroughly enumerate open ports
nmap -A target	Aggressive scan with OS and service detection	Gather detailed information about target
nmap -sV target	Service version detection	Identify running services and their versions
nmap -O target	OS detection	Determine the target's operating system
nmap -T4 target	Set timing template (fast)	Speed up the scanning process
nmap -T5 target	Set timing template (insane)	Fastest scan with minimal delay
nmap -sU target	UDP scan	Identify open UDP ports
nmap -p- target	Scan all 65,535 ports (TCP) and common UDP ports	Thoroughly enumerate all ports, including UDP
nmap -sn target	Ping scan (disable port scan)	Check host online status without scanning ports
nmap -Pn target	Skip host discovery (assume all hosts are up)	Force scan without host discovery
nmap -oA output target	Output results to files	Save scan results in various formats
nmap –script <script>target	Run NSE scripts	Execute Nmap Scripting Engine (NSE) scripts

   For API pentesting, we need to run the scan twice - once for general detection and once for all ports. The general method scans for default scripts and services against a target.

```
1
```

```
nmap -sC -sV <target address or network> -oA nameOfOutput.txt
```

The -sC flag runs the default script, while -sV checks the service version. The second scan checks all 65,535 TCP ports. See the command below.

```
nmap -p- <target address or network> -oA allports.txt
```

For example, using Nmap, I performed a general detection on the website crapi.apisces.ai. First, I ran nslookup to obtain the website's IP address, which we have permission to conduct pentesting. I ran a general scan, which can be viewed in figure 4.1.2.1. The scan revealed that the server had open ports for SSH, HTTP, and other services.

```
❯ nmap -sC -sV 5.161.86.65
Starting Nmap 7.94 ( https://nmap.org ) at 2023-09-19 16:13 PDT
Nmap scan report for static.65.86.161.5.clients.your-server.de (5.161.86.65)
Host is up (0.095s latency).
Not shown: 992 closed tcp ports (conn-refused)
PORT     STATE    SERVICE          VERSION
22/tcp   open     ssh              OpenSSH 8.2p1 Ubuntu 4ubuntu0.9 (Ubuntu Linux; protocol 2
.0)
| ssh-hostkey:
|   3072 97:0b:a8:24:ef:d6:2d:b2:ed:d4:80:f1:26:6b:52:09 (RSA)
|   256 1c:2a:3f:3e:91:eb:a3:ff:85:88:62:4c:fa:c5:87:2f (ECDSA)
|_  256 9c:31:d2:2d:4b:e8:de:51:43:b3:92:41:45:f9:2e:de (ED25519)
80/tcp   open     http             OpenResty web app server 1.17.8.2
|_http-title: crAPI
|_http-server-header: openresty/1.17.8.2
389/tcp  filtered ldap
636/tcp  filtered ldapssl
1025/tcp open     smtp             Postfix smtpd
|_smtp-commands: Hello static.65.86.161.5.clients.your-server.de, PIPELINING, AUTH PLAIN
1099/tcp filtered rmiregistry
3268/tcp filtered globalcatLDAP
3269/tcp filtered globalcatLDAPssl
Service Info: Host: mailhog.example; OS: Linux; CPE: cpe:/o:linux:linux_kernel
```

Figure 4.1.2.1: Nmap for crapi.apisec.ai

2. **Rustscan**

Rustscan is a rapid port scanning tool that uses adaptive learning to improve over time. This port scanner is high-speed; it can scan 65000 ports in less than

3 seconds. Rustscan now features a scripting engine that supports Python, Lua, and Shell. Figure 4.1.2.2 indicated the helper page for the Rustscan.

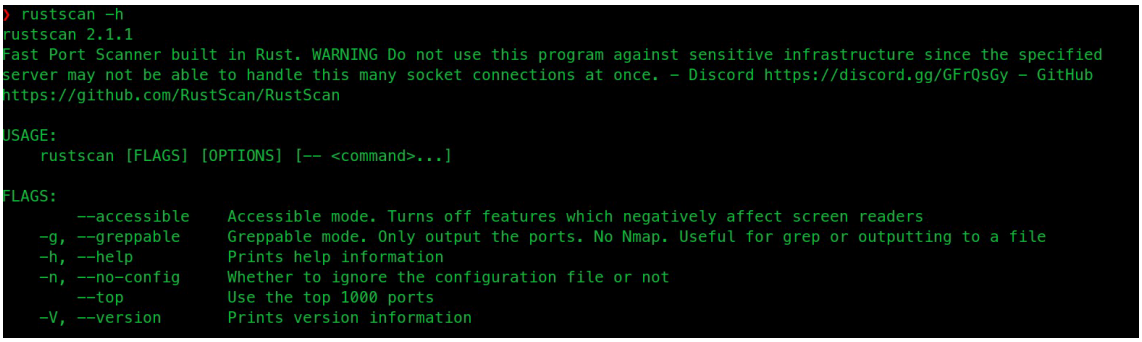


Figure 4.1.2.2: Rustscan help page

Rustscan provides different flags to do port scanning. For example, I ran the following command. The address can be specified using the -a command, while Nmap flags can be used after "–", such as -sC for the default script and -sV for service version checking.

```
1
2                rustscan [FLAGS] [OPTIONS] [-- <command>...]
3                rustscan -a scanme.nmap.org -- -sC -sV
4
```

When the command -b is used with a specified number, such as -b 10, 10 ports can be scanned simultaneously with a default timeout of 1000 or 1 second. To increase the timeout to 5 seconds, the command -T 5000 can be used.

4.1.2.2 Burp Suite

Users can activate FoxyProxy on their browser to intercept communication and switch to Burp Suite mode. In the Burp Suite, the user must go to the proxy and set the

intercept on. After that, they can attempt to authenticate with the web server and observe how the data is transmitted to the system by seeing it in the Burp Suite.

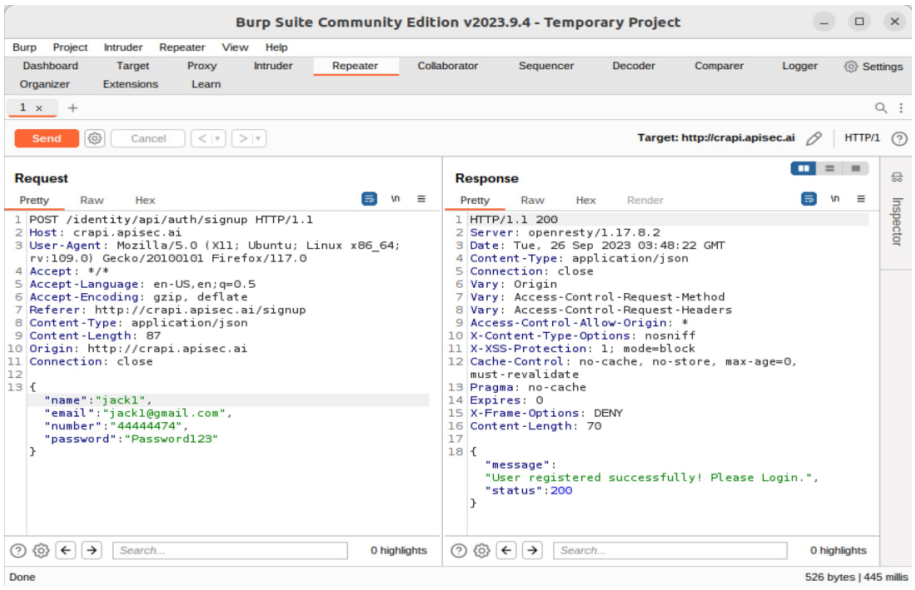


Figure 4.1.2.3: Burp Suite intercept creating account

Figure 4.1.2.3 shows that the user tried to create a new account crapi.apisec.ai, a vulnerable server for pentesting. We can see the server is trying to do POST requests to what endpoints. This information is essential for our pentesting.

4.1.3 Gobuster

Gobuster is a tool with many functions, including directory brute force, which can reveal available directories on a target. It can also perform DNS subdomain brute-forcing to discover additional subdomains associated with the target. Additionally, Gobuster includes a command for detecting S3 buckets and a host option to identify other subdomains hosted by the system. It can also fuzz for brute-forcing passwords by modifying the FUZZ keyword. To execute the command, follow this example.

```
1
```

```
2                gobuster dir -u 127.0.0.1 -w /opt/SecLists/Discover/Web-Content/
   ↪ common.txt --threads 10 -delay 1s
3
```

This command performs directory brute-forcing using the **dir** argument. The **-u** command is used to specify the URL, and the **-w** argument is for the wordlist. The number of threads and delay are used to send ten requests per second.

4.2 Authentication

Authentication involves confirming the identity of an entity, ensuring that it genuinely corresponds to its claimed identity [33]. Technology for verification controls system entry by validating whether the provided credentials align with those stored in an authorized user database or a data authentication server. Through this, verification ensures the security of systems, processes, and overall enterprise information. Throughout this section, I will be covering three types of pentesting on brute force password-based authentication, one-time password (OTP) authentication, and JSON web token, "JWT" .

The following describes the different types of authentication:

1. **Password-based authentication:** is a popular method for verifying user identity. However, it has vulnerabilities that cybercriminals can exploit. Brute-force and phishing attacks can compromise the system's security. Additionally, users' tendency to reuse passwords across multiple platforms can result in unauthorized access if a password is leaked.

2. **Two-factor (2FA)/multifactor authentication:** To provide a higher level of user authentication security, this method requires users to execute one or

more additional verification measures. These could include the usage of a pin sent to the recipient's email through a message or using a physical key such as Yubikey.

3. **Biometric authentication:** Biometric authentication uses unique personal characteristics to identify, including fingerprint scanning, facial recognition, iris recognition, and behavioral biometrics [34].

4. **Single sign-on (SSO):** In this way, the user has an account trusted by the identity provider (IdP) and tells the application via cookies or tokens that the user verified. This reduces the number of credentials a user needs to remember and strengthens security.

5. **Token-based authentication:** Token authentication allows individuals to access their accounts using a physical device like a smartphone, security key, or smart card. This approach can be integrated into Multi-Factor Authentication (MFA) or a password-less login. It permits users to verify their identity once within a given timeframe, decreasing the necessity for multiple logins [34]. This method makes it challenging for attackers to gain access to an account because they need physical access to the token.

4.2.1 Password-based Authentication Brute-Force Attack

This attack is on the authentication API endpoints used for generating targeted password lists, we can use https://github.com/sc0tfree/mentalist or common User Password Profiler https://github.com/Mebus/cupp.

4.2.1.1 Wfuzz

Wfuzz is a password brute-forcing tool. It searches for the keyword "FUZZ" in your command and continues to brute-force until it finds the password. To fuzz, you can use "-d" to send post data in the body. If you want to hide responses with a specific code, use the "–hc" flag. To set the payload for each FUZZ keyword, use "-z". In our example, we used a file and provided a wordlist.

```
1
2          wfuzz options -z payload, params url
3          # -d => fuzz the content that is sent in the body of the post
4          # curly => post request body
5          # --hc => hides responses with certain response code
6
7          wfuzz -d '{"email":"pogba006@example.com", "password": "FUZZ"}' --hc 405
   ↪  -H 'Content-Type:application/json' -z file,'/home/ranger/Downloads/rockyou
   ↪ .txt'  http://100.95.131.79:8888/api/v2/auth
8
```

Listing 4.2.1.1: Wfuzz

Figure 4.2.1.1 shows how I ran crapi.apisec.ai. The 200 responses indicate that we achieved cracking the password.

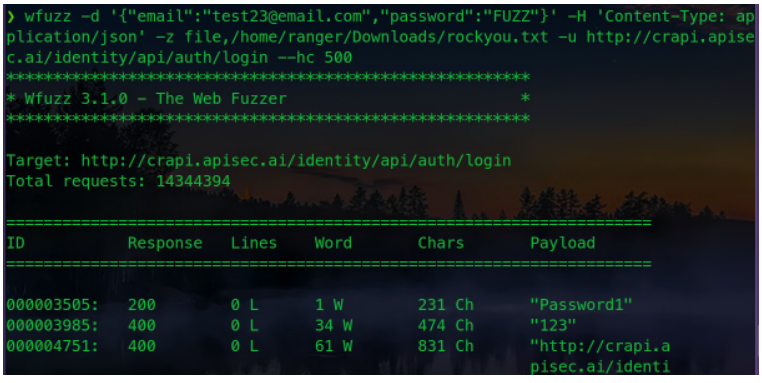


Figure 4.2.1.1: Crack the user password and get 200 response

4.2.1.2 Ffuf

In the ffuf we need to copy the request with the help of Burpsuite. For example, in figure 4.2.1.2, you can see the post request intercepted by Burp Suite and the email and password change with the keywords FUZZ and WFUZZ. We changed the email and password because when we are brute-forcing the ffuf, look for keywords that have some FUZZ and then use that for brute-forcing.

Pretty Raw Hex

```
1  POST /identity/api/auth/login HTTP/1.1
2  Host: crapi.apisec.ai
3  User-Agent: Mozilla/5.0 (X11; Ubuntu; Linux x86_64; rv:109.0) Gecko/20100101 Firefox/118.0
4  Accept: */*
5  Accept-Language: en-US,en;q=0.5
6  Accept-Encoding: gzip, deflate, br
7  Referer: http://crapi.apisec.ai/login
8  Content-Type: application/json
9  Content-Length: 47
10 Origin: http://crapi.apisec.ai
11 Connection: close
12
13 {
     "email":"FUZZ",
     "password":"WFUZZ"
   }
```

Figure 4.2.1.2: Post request intercepted by Bursp Suite

Here's a guide on using ffuf to brute force basic authentication. We need to specify the file we copied to the request using Burp Suite in the "-request" field. The protocol we use is set in the "-request-proto" field, which is HTTPS by default but can be changed to HTTP, like in our example. The "-mode" field allows us to choose from multiple methods for brute forcing. By default, ffuf uses clusterbomb, which iterates through a different payload set for each position [35]. This results in a combination of sets. Another method is pitchfork, which iterates through a different payload set for each defined position. The last method is sniper, which places each payload position in turn [35]. The "-w" we specify the wordlist and the keyword they need to look for changes. Last, the "-mc" to match HTTP status code in our request.

```
1          # -request File containing the raw http request
```

```
2         # -request-proto Protocol to use along with raw request (default: https)
3         # -mode Multi-wordlist operation mode. Available modes: clusterbomb,
    ↪ pitchfork, sniper (default: clusterbomb)
4         # -w Wordlist file path and (optional) keyword separated by colon. eg.
    ↪ '/path/to/wordlist:KEYWORD'
5         # -c color
6         # -mc Match HTTP status codes, or "all" for everything. (default:
    ↪ 200,204,301,302,307,401,403,405,500)
7         ffuf -request ./requirement.txt -request-proto http -mode pitchfork -w
    ↪ ./emails.txt:HFUZZ -w ./pass.txt:WFUZZ -c -mc 200
8
9
```

Listing 4.2.1.2: FFUF similar attack to Wfuzz

Figure 4.2.1.3 indicates how we use ffuf for brute-forcing the password on a CrAPI vulnerable server. We cracked a couple of users' emails and passwords to log in to the server.

4.2.2 One Time Password (OTP) Brute-Force Attack

To ensure secure access to digital services like banks, one-time password authentication is commonly used. This method generates random strings or numbers the user must authenticate with each time they log in to their account. However, it can be vulnerable if the server has no rate limit and can be subjected to brute-force attacks. For example, in CrAPI, as shown in figure 4.2.2.1.

```
❯ ffuf -request ./requirement.txt -request-proto http -mode pitchfork -w ./emails.txt:HFUZZ -w ./pass.txt:WFUZZ -c -mc 200

       /'___\  /'___\           /'___\
      /\ \__/ /\ \__/  __  __  /\ \__/
      \ \ ,__\\ \ ,__\/\ \/\ \ \ \ ,__\
       \ \ \_/ \ \ \_/\ \ \_\ \ \ \ \_/
        \ \_\   \ \_\  \ \____/  \ \_\
         \/_/    \/_/   \/___/    \/_/

       v2.0.0-dev
________________________________________________

 :: Method           : POST
 :: URL              : http://vapi.apisec.ai/vapi/api2/user/login
 :: Wordlist         : HFUZZ: /home/ranger/Downloads/emails.txt
 :: Wordlist         : WFUZZ: /home/ranger/Downloads/pass.txt
 :: Header           : Postman-Token: afd16479-558d-4d25-a0ec-91ad24b3210f
 :: Header           : Host: vapi.apisec.ai
 :: Header           : Accept-Encoding: gzip, deflate
 :: Header           : Connection: close
 :: Header           : Content-Type: application/json
 :: Header           : User-Agent: PostmanRuntime/7.29.0
 :: Header           : Accept: */*
 :: Data             : {
    "email":"HFUZZ",
    "password":"WFUZZ"
}
 :: Follow redirects : false
 :: Calibration      : false
 :: Timeout          : 10
 :: Threads          : 40
 :: Matcher          : Response status: 200
________________________________________________

[Status: 200, Size: 89, Words: 1, Lines: 1, Duration: 3067ms]
    * HFUZZ: savanna48@ortiz.com
    * WFUZZ: zTyBwV/9

[Status: 200, Size: 89, Words: 1, Lines: 1, Duration: 2892ms]
    * HFUZZ: hauck.aletha@yahoo.com
    * WFUZZ: kU-wDE7r

[Status: 200, Size: 89, Words: 1, Lines: 1, Duration: 3157ms]
    * HFUZZ: harber.leif@beatty.info
    * WFUZZ: kU-wDE7r

:: Progress: [1000/1000] :: Job [1/1] :: 12 req/sec :: Duration: [0:01:21] :: Errors: 0 ::
```

Figure 4.2.1.3: Crack the user password and get 200 responses

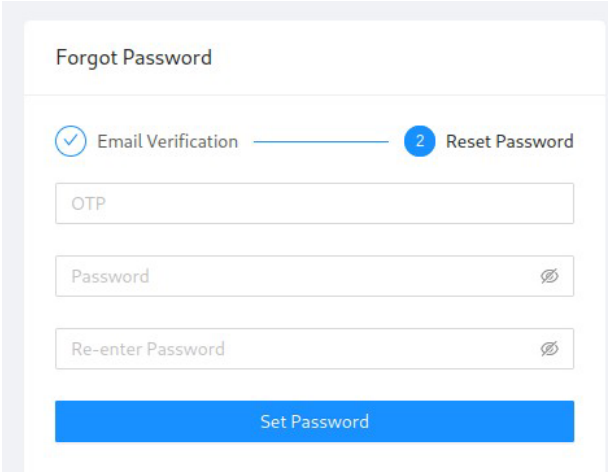


Figure 4.2.2.1: CrAPI OTP password reset

To retrieve the OTP password request, you can intercept the post request from Foxyproxy and Burp Suite. Figure 4.2.2.2 displays the results, showing that the application uses v3 and is not susceptible to our attack. However, if the version is downgraded to v2, it becomes an endpoint management vulnerability. This will enable us to use v2 to brute-force and obtain the OTP.

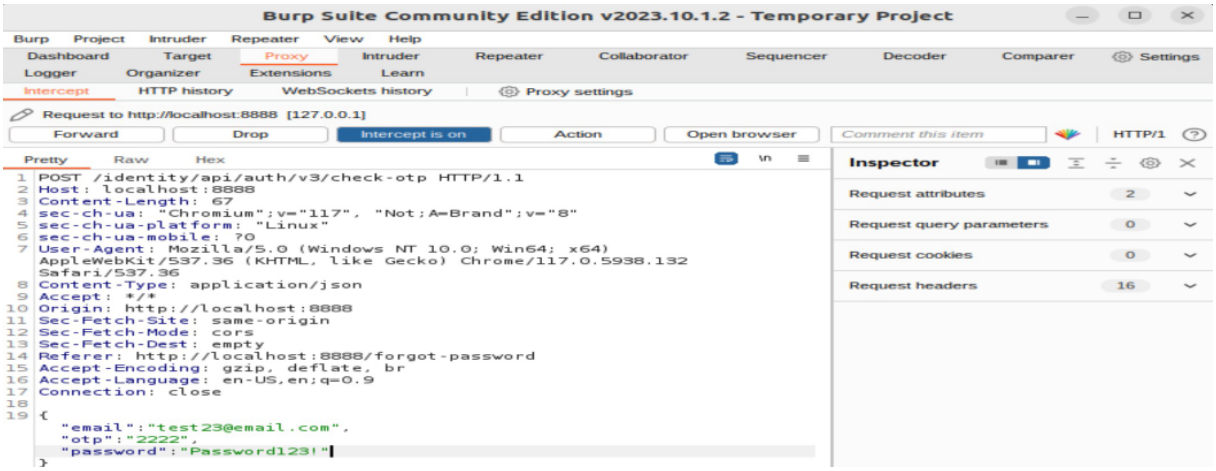


Figure 4.2.2.2: OTP Burp Suite intercept

There are two methods to obtain an OTP pin using Wfuzz or Ffuf. Figure 4.2.2.2 indicates that the endpoints are located in /identity/api/auth/v2/check-otp. To fuzz the desired location, we use the FUZZ key and create a POST request body with -d. In Wfuzz, using –hc hides the response 500, which prevents us from receiving too many 500 responses. Instead, we only received the desired 200 responses. Similarly, in Ffuf, -mc is used to show only the 200 responses.

```
1        wfuzz -d '{"email":"ha@email.com", "otp": "FUZZ", "password": "
   ↪ Password123!"}' --hc 500 -H 'Content-Type:application/json' -z range
   ↪ ,0000-9999  http://localhost:8888/identity/api/auth/v2/check-otp

2

3            ffuf -w 4-digit.txt  -u http://localhost:8888/identity/api/auth/v2
   ↪ /check-otp  -H 'Content-Type:application/json' -X POST  -d '{"email":"
   ↪ ha@email.com", "otp": "FUZZ", "password": "Password123!"}'  -mc 200
```

```
4
```

Listing 4.2.2.1: Wfuzz and FFUF OTP

Figure 4.2.2.3 shows that we successfully brute-forced the OTP and used the Wfuzz command to discover the four-digit PINs.

Figure 4.2.2.3: Brute-force OTP

4.2.3 JSON Web Tokens (JWT)

A JSON Web Token (JWT) serves as an open standard (RFC 7519), outlining a concise and self-contained method to securely convey information between involved parties in the form of a JSON object [36]. The credibility of this information is upheld through digital signatures. JWTs can undergo signing processes, employing either a secret (via HMAC algorithm) or a public/private key pair through RSA or ECDSA [36].

Although JWTs provide encryption and secrecy, our focus is on signed tokens. The signed key ensures the integrity of our information, while encrypted tokens hide it from other parties. There are two reasons why we use JWTs: for user authorization when logged in, for including JWT tokens in every request, and for securely transmitting information.

JWT is structured in three parts, separated by dots. These three parts included **header, payload, and signature.** The structure will look like:

```
xxxxx.yyyyy.zzzzz

```

Listing 4.2.3.1: JWT token

The **header** consists of two parts: the type of token, that is, JWT, and the signing algorithm, which could be HMAC SHA256 or RSA. The example will be:

```
{
    "alg": "HS256",
    "type": "JWT"
    }

```

Listing 4.2.3.2: JWT Header

This is encoded in Bas64Url. The **payload** is a predefined claim that is not mandatory. The **registered claims** are **iss (issuer), exp(expiration), sub(subject), and aud(audience). Public claims** are defined for the using of JWTs but should consider collisions, and for evidence of that, they should defined in IANA JSON Web Token Registery [36]. Last, **private claims** are customer claims for sharing information between parties, which is encrypted as Bas64Url. The payload will look like this:

```
{
      "sub": "1234567890",
      "name": "John Doe",
      "admin": true
    }

```

Listing 4.2.3.3: JWT Payload

To create a **signature**, we need to use an encoded header, an encoded payload, a secret key, and the chosen algorithm for the header. For instance, if we want to use HMAC SHA256, the signature should be generated as follows, according to [36]:

```
1        HMACSHA256(
2            base64UrlEncode(header) + "." +
3            base64UrlEncode(payload),
4            secret)
5
```

Listing 4.2.3.4: JWT Signature

In summary, figure 4.2.3.1 indicates that users sent requests that are marked as one, and the authorization server responded with the token. After that, the user can access the resources on the server.

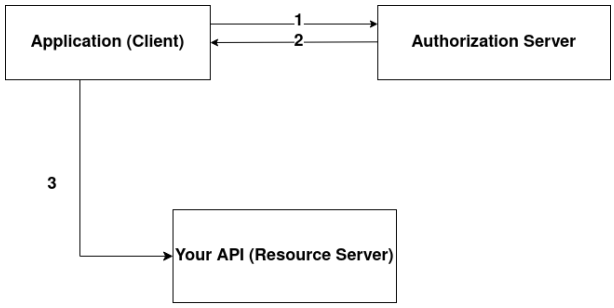


Figure 4.2.3.1: JWT authentication between server and client

4.2.3.1 Attacking JWT

In our example of CrAPI, you get a token response after logging in to the system. When intercepting the log-in page with Burp Suite and getting a response, it looks like the figure 4.2.3.2. As we can see, dots separate the structure token.

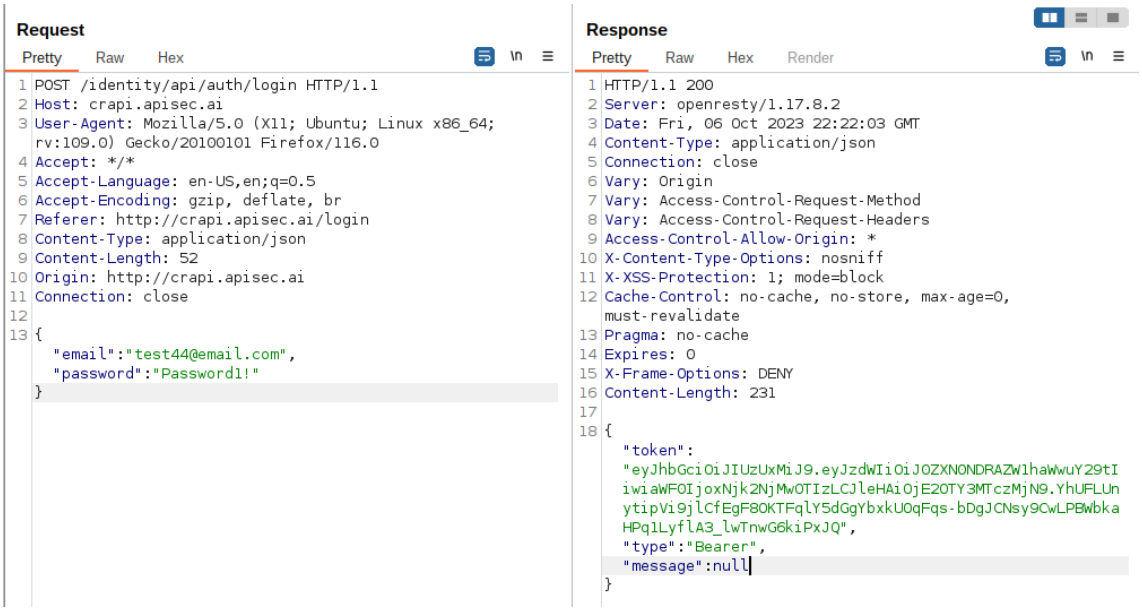


Figure 4.2.3.2: Authentication with website and get token with Burp Suite

To get the information about our token, there are two paths you can take. one is the website jwt.io and the other paths is using the **jwt_tool**. If we use the website in the encoded section, we provide the token and click on **SHARE JWT**. As a result, we get an analysis of each section of the JWT token. On the other hand, we can use the jwt_tool, which is run in the terminal. To analyze our JWT token, we need to run it as **jwt_tool jwt_token**. Figure 4.2.3.3 shows that we got the following results by running that command. Figure 4.2.3.3 illustrates the payload and the header file.

Figure 4.2.3.3: jwt_tool JWT token analysis

Jwt_tool has a different mode of scanning for common vulnerabilities. With **-M** we can choose between:

1. pb = playbook audit
2. er = fuzz existing claims to force errors
3. cc = fuzz common claims
4. at = All Tests!

Below is an example of how to run this for the playbook audit method with specifying **pb**.

```
1       jwt_tool -t http://crapi.apisec.ai/identity/api/v2/user/dashboard -rh "
    ↪ Authorization: Bearer Your_JWT_token" -M pb
2
```

Listing 4.2.3.5: JWT Playbook Audit Attack

Jwt_tool provides different exploits against the JWT token. This can be specified with **-X**. The set after -X makes the algorithm in the header none and generates new

tokens, which we can test to see if API is vulnerable to this token. Below is all the exploit method available for jwt_tool:

1. a = alg:none
2. n = null signature
3. b = blank password accepted in signature
4. s = spoof JWKS (specify JWKS URL with -ju, or set in jwtconf.ini to automate this attack)
5. k = key confusion (specify public key with -pk)
6. i = inject inline JWKS

The command will look like:

```
jwt_tool <Your_JWT_token> -X <options (a, n, b, s)>

```

Listing 4.2.3.6: JWT exploit

JWT Crack Attack:

This method is used to forcefully guess the secret for the signature by attempting to sign a new token with various combinations of usernames or email addresses. For instance, to access the CrAPI signature key, we can use the tool "crunch" to generate multiple vital combinations. To generate all the variety of lengths of five, run the following command.

```
crunch 5 5 -o crAPI.txt

```

Listing 4.2.3.7: Crunch password creates a 5 character combination password

After saving all the combinations in the CrAPI.txt, we can use jwt_tool to brute-force the signed key. Figure 4.2.3.4 shows how to crack with jwt_tool. We need to specify the **-C**, which indicates we want to use crack attack, and **-d** for specifying our all-key combination file.

Figure 4.2.3.4: Crack Attack

Figure 4.2.3.4 shows that the signing key, crapi, was successfully cracked. If we visit jwt.io, we could use this key to generate a new key and change the associated email to gain access to the user account, as depicted in Figure 4.2.3.5.

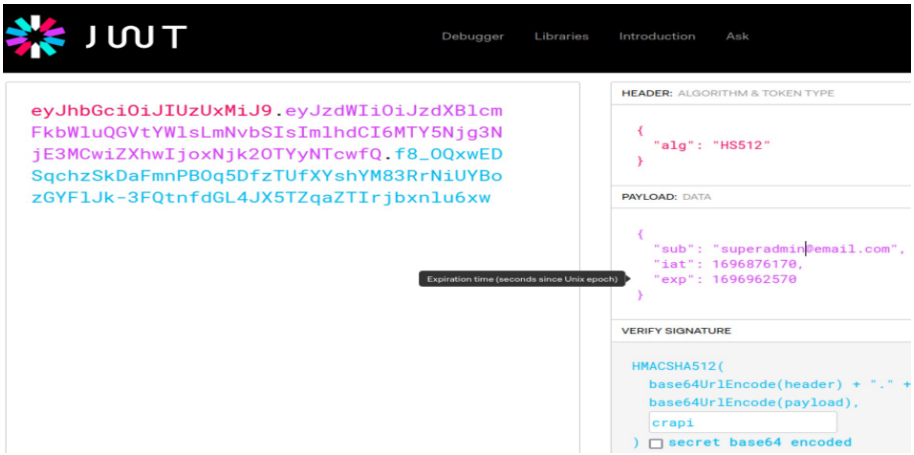


Figure 4.2.3.5: Using jwt.io to generate token

Chapter 5

CONCLUSION & FUTURE WORK

5.1 Conclusion

In this thesis, first I introduced the basics of APIs and the various API protocols used in API communication. Confidentiality, integrity, and availability of API security are discussed. In the course of this thesis, we found that many vulnerabilities come from misconfiguration of the authentications. If the developers correctly configure the system, there's less chance the system will be vulnerable. Additionally, we delved into the integral components of API security, highlighting the crucial elements that form its foundational framework. We emphasized the significance of a triad comprising confidentiality, integrity, and availability as the three indispensable pillars of API security. These pillars serve as the bedrock for robust security measures and play a pivotal role in shaping and fortifying API systems. Most common breaches occur in large companies and expose them to the risk of data breaches, allowing the attacker to access personal identifiable information (PII). Additionally, we discussed OWASP APIs top ten security vulnerabilities and compared the vulnerabilities between 2019 and 2023 to demonstrate the similarities and differences between the years. We covered deployable scripts that install the required pentesting tools to the Ubuntu systems and prepares the system for pentesting for reconnaissance and authentication. The script was written in Ansible, developed by RedHat, enabling the user to deploy the script for system configuration. This script first installs the necessary packages for the Ubuntu system and uses the "includes" function in the local.yml file and all the scripts included in the tasks folder to install necessary cyber security tools and system

configurations, such as background setup, FireFox plugins, terminal setup, Tmux configuration, terminal prompts, coding editor and Vim configurations. Finally, we explored API reconnaissance and authentication, diving deeper into the active and passive approaches to API reconnaissance to fortify API security and authentication pentesting on brute-force attacks against password-based authentication, one-time password, and JWT tokens.

5.2 Future Work

Following the conclusion of this study, there's a multitude of pentesting objectives yet to be studied. First, to improve the deployment of the Ansible script discussed in Chapter three, I recommend using a Docker container, which provides a consistent and reproducible environment to run applications and bundles applications and dependencies into a singular container. Some suitable Docker images like Webtop or Kasmweb allow you to run the operating system in a web browser. You can then utilize a Dockerfile to configure images and install necessary packages with the Ansible script. I faced a challenge when I attempted this approach. Two applications, Burp Suite and Zaproxy, require a graphical user interface for installation, which prevented the use of Docker due to the lack of a user interface during the compilation of the Dockerfile. This limitation posed a hurdle I couldn't overcome in my attempts.

Additionally, it's worth noting that the OWASP API top ten for 2023 emphasizes a deeper exploration of GraphQL. My current coverage, however, is limited to REST API. The industry is presently shifting towards GraphQL as the preferred choice for API development. Unlike REST APIs, where clients receive a fixed data set in a predefined structure, GraphQL allows clients to request only the data they need in

a specified format. This flexibility is more efficient and can handle faster interaction between clients and servers.

Furthermore, all the tests were done on a vulnerable environment, CrAPI, which we know is susceptible to attack. If we have permission to use different companies to test their system, it will provide us the opportunity to work on real-life experiments.

Time constraints only allowed me to cover three areas under authentication pentesting. In the future, it is essential to explore Multi-Factor Authentication (MFA), Open Authorization, and signal sign-on (SSO) to understand the standards used in the industry and increase proficiency in authentication pentesting.

BIBLIOGRAPHY

[1] C. Ball, "Api penetration testing," https://www.apisecuniversity.com/, accessed: July 8, 2023.

[2] Amass OAM, "Amass github," https://github.com/owasp-amass/oam-tools.git, 2021-07-21.

[3] D. C. . R. V. J. F. U. the Power of OWASP Amass. (2023) Jeff foley. YouTube video. [Online]. Available: https://youtu.be/IgxPsv8MXMw?si=WT71Wrmz1A7GhrKL

[4] Cal Poly, "Cal poly github," http://www.github.com/CalPoly, accessed: July 8, 2023.

[5] Jenny Davidse, "Api fundamentals," https://developer.ibm.com/articles/api-fundamentals/, accessed: July 8, 2023.

[6] AWS, "What is an api (application programming interface)?" https://aws.amazon.com/what-is/api/, accessed: July 8, 2023.

[7] J. Greig, "Coinbase pays out largest bug bounty ever for trading interface flaw," https://www.zdnet.com/finance/blockchain/coinbase-pays-out-largest-bug-bounty-ever-for-trading-interface-flaw/, accessed: July 8, 2023.

[8] L. Tung, "Usps finally fixes website flaw that exposed 60 million users' data," https://www.zdnet.com/article/usps-finally-fixes-website-flaw-that-exposed-60-million-users-data/, accessed: July 8, 2023.

[9] Z. Whittaker, "Peloton's leaky api let anyone grab riders' private account data,"
https://techcrunch.com/2021/05/05/peloton-bug-account-data-leak/, accessed: July 8, 2023.

[10] D. Salmon, "I scraped millions of venmo payments. your data is at risk,"
https://www.wired.com/story/i-scraped-millions-of-venmo-payments-your-data-is-at-risk/, accessed: July 8, 2023.

[11] M. GAJANAN, "Instagram says bug gave hackers data on 'high-profile' users,"
https://time.com/4922700/instagram-security-breach-verified-users/, accessed: July 8, 2023.

[12] C. Kanikee, "The optus breach: How bad code keeps happening to good companies," https://securityboulevard.com/2022/09/the-optus-breach-how-bad-code-keeps-happening-to-good-companies/, accessed: July 8, 2023.

[13] Codecademy, "What is rest?"
https://www.codecademy.com/article/what-is-rest, accessed: July 8, 2023.

[14] S. Dinda, "6 types of api architectures and how they work,"
https://www.makeuseof.com/api-architecture-types-how-work/, accessed: July 8, 2023.

[15] Indeed, "What is soap api? (plus comparison to rest api and benefits),"
https://www.indeed.com/career-advice/career-development/what-is-soap-api#:~:text=SOAP%20API%2C%20or%20simple%20object,Extensible%20Markup%20Language%20(XML)., accessed: July 8, 2023.

[16] Baeldung, "Introduction to graphql," https://www.baeldung.com/graphql, accessed: July 8, 2023.

[17] A. Mendoza and G. Gu, "Mobile application web api reconnaissance: Web-to-mobile inconsistencies & vulnerabilities," in *2018 IEEE Symposium on Security and Privacy (SP)*. IEEE, 2018, pp. 756–769.

[18] Bae, "T-mobile hacked to steal data of 37 million accounts in api data breach," https://www.bleepingcomputer.com/news/security/t-mobile-hacked-to-steal-data-of-37-million-accounts-in-api-data-breach/#:~:text=New%20data%20breach%20impacts%2037,the%20API%20one%20day%20later., accessed: July 8, 2023.

[19] T. Keary, "T-mobile data breach shows api security can't be ignored," https://venturebeat.com/security/t-mobile-data-breach-shows-api-security-cant-be-ignored/, accessed: July 8, 2023.

[20] CMS.gov, "Electronic health records," https://www.cms.gov/Medicare/E-Health/EHealthRecords, accessed: July 8, 2023.

[21] R. Sheldon, "passive reconnaissance," https://www.techtarget.com/whatis/definition/passive-reconnaissance, accessed: July 8, 2023.

[22] X. Mendez, "Wfuzz github," https://github.com/xmendez/wfuzz, accessed: July 8, 2023.

[23] Mozilla Add-ons, "ublock origin," https://addons.mozilla.org/en-CA/firefox/addon/ublock-origin/, accessed: July 8, 2023.

[24] OWASP.org, "Owasp api security project," https://owasp.org/www-project-api-security/, accessed: July 8, 2023.

[25] ticarpi, "jwt_tool," https://github.com/ticarpi/jwt_tool.

[26] R. Hat, "How ansible works," https://www.ansible.com/overview/how-ansible-works.

[27] Techgaun, "Github-dorks," https://github.com/techgaun/github-dorks.git.

[28] Trufflesecurity, "trufflehog," https://github.com/trufflesecurity/trufflehog.git.

[29] Michenriksen, "Gitrob," https://github.com/michenriksen/gitrob.git.

[30] A. H, "Active reconnaissance: Overview, methodology and tools," https://securityboulevard.com/2021/07/active-reconnaissance-overview-methodology-and-tools/, 2021.

[31] Cuncis, "How the internet archive wayback machine can help pentesters find hidden vulnerabilities," https://medium.com/@cuncis/how-the-internet-archive-wayback-machine-can-help-pentesters-find-hidden-vulnerabilities-2604fe31ba0c, 2023-04-10.

[32] Tomnomnom, "waybackurls github," https://github.com/tomnomnom/waybackurls.git.

[33] Mary E. Shacklett, "Authentication," https://www.techtarget.com/searchsecurity/definition/authentication.

[34] Kyle Johnson, "Use these 6 user authentication types to secure networks," https://www.techtarget.com/searchsecurity/tip/Use-these-6-user-authentication-types-to-secure-networks.

[35] Portsswigger, "Burp intruder attack types," https://portswigger.net/burp/documentation/desktop/tools/intruder/configure-attack/attack-types.

[36] jwt.io, "Introduction to json web tokens," https://jwt.io/introduction.

[37] Cybersecurity Exchange, "Understanding the basics of footprinting and reconnaissance," https://www.eccouncil.org/cybersecurity-exchange/ethical-hacking/basics-footprinting-reconnaissance/.

[38] Ritu Gill, "What is open-source intelligence?" https://www.eccouncil.org/cybersecurity-exchange/ethical-hacking/basics-footprinting-reconnaissance/.

[39] Baivab Kumar Jena, "What is google dorking? the best google hacker," https://www.simplilearn.com/tutorials/cyber-security-tutorial/google-dorking.

[40] Wayback Machine, "Using the wayback machine," https://help.archive.org/help/using-the-wayback-machine/.

[41] Rowena Johansen, "Ethical hacking code of ethics: Security, risk & issues," https://panmore.com/ethical-hacking-code-of-ethics-security-risk-issues.

[42] University of Denver, "The complete guide to ethical hacking," https://bootcamp.du.edu/blog/the-complete-guide-to-ethical-hacking/.

[43] APIsec, "Burp suite vs. owasp zap - which is better for api security testing?" https://www.apisec.ai/blog/burp-suite-vs-zap.

[44] Prashant Phatak, "A comprehensive comparison of owasp zap and burp suite vulnerability assessment tools (part 1)," https://www.valencynetworks.com/blogs/a-comprehensive-comparison-of-owasp-zap-and-burp-suite-vulnerability-assessment-tools-part-1/.

[45] Ricahrd, "Why you should use rustscan for port scanning," https://onlineblogzone.com/why-you-should-use-rustscan-for-port-scanning/.

[46] V. Aggarwal, D. Kaur, S. Mittal, T. J. S. Prasad, D. Batra, and A. Garg, "A comparative study of directory fuzzing tools," in *2023 International Conference on Circuit Power and Computing Technologies (ICCPCT)*, 2023, pp. 1368–1374.

[47] R. F. Khalil, "Why johnny still can't pentest: A comparative analysis of open-source black-box web vulnerability scanners," 2018. [Online]. Available: https://api.semanticscholar.org/CorpusID:86510564

[48] Zap, "A quick start guide to building zap," https://www.zaproxy.org/docs/developer/quick-start-build/.

[49] Burp Suite, "Burp suite pro," https://portswigger.net/burp/pro.

[50] Rustscan, "False comparison," https://github.com/RustScan/RustScan/issues/141#issuecomment-671308963.

[51] APISEC, "Crapi," http://crapi.apisec.ai/.