**API Security Study Plan**

It will cover what you need to learn to excel in API security. And please keep in mind that it would require knowledge of:

1. How website works
2. How API endpoints are defined and its request and response
3. You know basics of coding to write your own APIs for testing.
4. OWASP Top 10 for web
5. OWASP Top 10 for API

I will explain very basic or just the overview so that you can start learning those concepts from there.

**ToC:**

1. API Fundamentals
   1. API endpoints
   2. Types of microservices
   3. microservices from API Security perspective
   4. [Cloud-Native API](https://github.com/jassics/security-study-plan/blob/main/api-security-study-plan.md#4-cloud-native-api)
2. API Security Understandings
   1. API Security is not web security.
   2. What is API Security
   3. Why API Security
   4. AuthN and AuthZ
   5. Rate Limit
   6. API Gateway
3. API Security Labs and Practices
4. API Security Tools
5. Books
6. Videos
7. Courses
8. Certifications
9. Interview Questions

**API Fundamentals**

**1. API endpoints**

**Let's try to understand API endpoints in layman terms.**

Imagine you're ordering food from a restaurant using a food delivery app. You open the app and select the items you want to order. When you're done, you click the "Place Order" button. Behind the scenes, the app needs to communicate with the restaurant's system to process your order. This is where API endpoints come into play.

API endpoints are like specific doors or entrances at the restaurant that allow the app and the restaurant's system to exchange information. Each endpoint serves a particular purpose or action. For example, there might be an endpoint for submitting an order, another one for retrieving the menu, and yet another one for tracking the status of your order.

When you click "Place Order" in the app, it sends a request to a specific API endpoint designated for order submission. This request contains all the necessary information, such as the items you ordered, your delivery address, and any special instructions. The endpoint receives this request and processes it. It may check if the items are available, calculate the total cost, and initiate the delivery process.

Similarly, if you want to check the status of your order later, the app sends a request to a different API endpoint dedicated to order tracking. This endpoint retrieves the relevant information about your order, such as whether it's being prepared, out for delivery, or already delivered. The endpoint then sends this information back to the app, which displays it to you.

**In summary,** API endpoints act as communication channels between different systems, allowing them to exchange information and perform specific actions. They serve as designated entry points that receive requests from one system and provide the necessary responses based on the requested action.

**2. Types of microservices**

Try to understand what types of microservices exist from application architectural view. Then it will be easy for you to understand microservices patter. Once you know these concepts, it will help you to the API security assessment accurately.

Microservices are an architectural approach to building software applications by breaking them down into small, loosely coupled, and independently deployable services. Each microservice focuses on a specific business capability or function. Here are some common types of microservices:

1. User Service: This microservice is responsible for managing user-related functionalities such as user registration, authentication, profile management, and access control. It handles tasks like user creation, login, password management, and user-specific settings.
2. Product Service: The product service handles tasks related to managing products or items in an application. It includes functions like product catalogue management, inventory management, pricing, and product search. This microservice allows the application to efficiently handle product-related operations.
3. Order Service: This microservice deals with order processing and management. It handles tasks such as creating new orders, updating order status, handling payments, and generating invoices. The order service enables seamless management of the entire order lifecycle.
4. Payment Service: This microservice focuses on payment processing. It integrates with payment gateways and handles tasks like accepting and verifying payment information, processing transactions, and ensuring secure payment processing.
5. Notification Service: The notification service is responsible for sending out various types of notifications, such as email notifications, SMS alerts, or push notifications. It enables the application to communicate important information to users or other systems in a reliable and timely manner.
6. Analytics Service: This microservice deals with collecting, processing, and analysing data to provide insights and metrics about the application and its usage. It helps in monitoring system performance, identifying patterns, and making data-driven decisions.
7. Integration Service: The integration service facilitates communication and integration between different systems or external services. It enables the microservices-based application to interact with external APIs, third-party services, or legacy systems by providing adapters, connectors, or translation layers.
8. Image/Video Processing Service: This microservice focuses on image or video processing capabilities. It may handle tasks such as resizing images, generating thumbnails, applying filters, or encoding/decoding videos. This service is especially useful in applications that involve multimedia content.

These are just a few examples of the types of microservices that can exist in an application. The specific types and number of microservices can vary depending on the requirements of the application and the domain it serves. The goal is to decompose the application into small, manageable services, each responsible for a specific business capability, allowing for scalability, agility, and independent development and deployment.

**3. Try to understand Microservices from API Security perspective**

When it comes to API security in the context of microservices, there are a few important types of microservices that developers and security professionals should be aware of. Let's explore them:

1. Authentication Service: This microservice handles user authentication and identity management. It verifies the identity of users, usually through username/password combinations, tokens, or other authentication mechanisms. It ensures that only authorized users can access protected resources and performs actions like user registration, login, and logout.
2. Authorization Service: This microservice is responsible for controlling access to various resources within the system. It determines what actions a user or system can perform based on their identity and assigned permissions. The authorization service enforces access control policies and ensures that users can only access the resources they are authorized to access.
3. API Gateway: The API gateway acts as the entry point for external requests and provides a single point of contact for clients. It handles tasks like request routing, load balancing, caching, and protocol translation. From a security perspective, it can also enforce security policies, such as rate limiting, authentication, and encryption.
4. Logging and Monitoring Service: This microservice collects and analyzes logs and metrics from various components of the system. It helps detect anomalies, track system behavior, and identify security events. It plays a crucial role in identifying potential security breaches or attacks by monitoring and analyzing the API traffic and system logs.
5. Encryption Service: This microservice focuses on data protection by providing encryption and decryption capabilities. It ensures that sensitive information, such as user credentials or personal data, is securely transmitted and stored. The encryption service may handle tasks like generating and managing encryption keys, encrypting data at rest or in transit, and decrypting data when necessary.
6. Threat Intelligence Service: This microservice gathers information about known threats and vulnerabilities from various sources. It helps detect and mitigate potential security risks by analyzing incoming requests and comparing them against known threat patterns. The threat intelligence service can provide real-time threat alerts, block suspicious activities, or trigger additional security measures.

Understanding these types of microservices from an API security perspective allows developers and security professionals to design, implement, and maintain secure microservice architectures. It helps them identify the specific security concerns associated with each microservice and apply appropriate security measures to protect the system and its APIs from potential threats and attacks.

**4. Cloud native API**

You can't deny the fact that Cloud is everywhere and understanding how API is being used in cloud would be an aded advantage for you. A cloud-native API refers to an API that is designed, developed, and deployed with a cloud-native approach, taking full advantage of cloud computing capabilities and principles. Cloud-native APIs are specifically tailored for cloud environments, enabling scalability, resilience, and flexibility. Here are a few examples of cloud-native APIs:

1. RESTful API: Representational State Transfer (REST) is a widely used architectural style for building APIs. RESTful APIs are designed to be stateless and use standard HTTP methods such as GET, POST, PUT, and DELETE to interact with resources. They are well-suited for cloud-native environments as they leverage the HTTP protocol and can be easily consumed by different clients, including web browsers, mobile apps, and other services.
2. Event-Driven APIs: Event-driven APIs enable asynchronous communication and are often used in cloud-native architectures. They allow services to exchange information and trigger actions based on events. For example, when a new user registers on a platform, an event-driven API can publish an event that triggers other services to send welcome emails, update user profiles, or perform other related actions. Cloud-native event-driven APIs often utilize message queues, event brokers, or streaming platforms to facilitate event processing.
3. GraphQL API: GraphQL is a query language and runtime for APIs that allows clients to request specific data in a flexible and efficient manner. It enables clients to retrieve only the data they need, reducing the amount of network traffic and improving performance. GraphQL APIs are popular in cloud-native environments where microservices often need to interact with multiple data sources. They provide a single-entry point for clients to fetch data from various services and aggregate responses.
4. Serverless APIs: Serverless computing allows developers to build and deploy applications without managing the underlying infrastructure. Serverless APIs, often implemented using serverless computing platforms like AWS Lambda or Azure Functions, enable developers to focus solely on writing the API logic while leaving the infrastructure management to the cloud provider. Serverless APIs scale automatically based on demand, and developers only pay for the actual execution time of the API functions.
5. OpenAPI (formerly Swagger): OpenAPI is a specification that defines and documents RESTful APIs. It provides a machine-readable format for describing the API's endpoints, request/response payloads, and authentication mechanisms. OpenAPI allows developers to generate code stubs, automatically generate API documentation, and even test and mock APIs. It promotes collaboration and interoperability between different teams and tools in a cloud-native development ecosystem.

These examples illustrate different types of cloud-native APIs that leverage cloud infrastructure and principles to deliver scalable, resilient, and flexible solutions. Cloud-native APIs enable organizations to take full advantage of cloud computing benefits, such as elasticity, cost efficiency, and rapid deployment, while building robust and modern applications.

**API Security Understandings**

Now, as you understand basic blocks of API, types of API, microservices, cloud-native APIs etc. Let's understand few basic concepts while dealing with API Security like AuthN, AuthZ, OAuth, API Gateway, why API security is different from web security and so on.

**1. API Security is not web security!**

When I talk about API Security and say that web security is different. Many either get confused or don't agree. But ideally API security and web application security are two different concepts, although they are related. Web application security refers to the measures taken to protect web applications, including the servers, databases, and user interfaces that are involved in delivering web content. API security, on the other hand, is about protecting the API itself and the data that it transmits.

**Here are some of the key differences between API security and web application security:**

1. Focus: Web application security focuses on securing the user interface, server, and database parts of a web application. API security, however, concentrates on securing the API and the data it transmits.
2. Attack Vectors: Web applications are often targeted by attacks such as SQL injection, cross-site scripting (XSS), cross-site request forgery (CSRF), and file inclusion. API security, on the other hand, is typically targeted by attacks such as API spoofing, parameter manipulation, and man-in-the-middle (MitM) attacks.
3. Authentication and Authorization: Web application security often relies on username and password authentication, session management, and access control to protect web applications. API security, on the other hand, typically uses OAuth, API keys, or JSON Web Tokens (JWTs) for authentication and authorization.
4. Usage: Web applications typically provide a user interface to interact with, whereas APIs are back-end components that allow different systems to communicate. As a result, API security focuses on securing APIs that expose application functionality to other systems.

Overall, API security and web application security are equally important and require different measures to ensure they remain safe and secure. While web application security focuses on the user interface, server, and database, API security focuses on the API itself, the data it transmits, and the ways in which different systems interact with it.

**2. What is API Security**

API security refers to the measures taken to protect an Application Programming Interface (API) from unauthorized access, modification, and exploitation. APIs are a set of rules and protocols that allow various software applications to interact with each other. Therefore, they need to be secure to ensure that only authorized parties have access to them.

**API security can be ensured through the following measures:**

1. Encryption: This involves the use of encryption techniques to protect the data being transmitted between API endpoints. For example, HTTPS (HyperText Transfer Protocol Secure) is commonly used for secure data transmission.
2. Authentication: This involves verifying the identity of the user or application accessing the API. Authentication can be achieved using usernames and passwords or the use of tokens that expire after a certain period.
3. Authorization: This involves determining what actions a user or application can perform after they have been authenticated. For example, some users may have access to read data, while others may have access to modify data.
4. Rate Limiting: This involves limiting the number of requests that a user or application can make to the API. This helps to prevent Denial of Service attacks by limiting the number of resources that can be consumed by a user.

**Practical Examples:**

* Google Maps API: Google Maps API requires an API key for authentication. This key is specific to the developer's account and is used to restrict access to the API to authorized users. It also uses HTTPS to encrypt the data being transmitted between the API endpoints.
* Stripe API: The Stripe API uses authentication and API keys to restrict access to authorized users. It also uses rate limiting to prevent excessive API usage and offers real-time fraud detection to prevent fraudulent transactions.
* Twilio API: The Twilio API uses authentication and authorization to restrict access to authorized users. It also uses HTTPS to encrypt data and provides rate throttling to prevent excessive API usage. Additionally, it offers access control features to restrict API access to specific IP addresses or domains.

**3. Why API Security**

API security is crucial for several reasons:

1. Data Protection: APIs often handle sensitive data, such as user credentials, personal information, financial data, or business secrets. Securing APIs ensures that this data remains confidential and is not compromised or accessed by unauthorized entities.
2. Authorization and Access Control: APIs provide access to various resources and functionalities. Implementing proper security measures ensures that only authenticated and authorized users or systems can access and perform actions on those resources. It helps prevent unauthorized access, data breaches, and misuse of the API.
3. Trust and Reputation: Security breaches or vulnerabilities in APIs can damage the trust and reputation of an organization. Users and clients expect their data to be handled securely. Demonstrating a commitment to API security enhances trust, improves customer satisfaction, and helps maintain a positive reputation.
4. Compliance with Regulations: Many industries are subject to regulatory requirements regarding data protection and privacy, such as GDPR (General Data Protection Regulation) or HIPAA (Health Insurance Portability and Accountability Act). Ensuring API security helps organizations comply with these regulations and avoid legal consequences or penalties.
5. Prevention of Attacks: APIs can be targeted by various types of attacks, including injection attacks, cross-site scripting (XSS), cross-site request forgery (CSRF), or denial-of-service (DoS) attacks. Implementing security measures mitigates the risk of these attacks and protects the API and underlying systems from being compromised.
6. Secure Integration: APIs are often used to integrate different systems, services, or third-party applications. Proper security measures ensure that these integrations are performed securely, protecting both the API provider and the entities consuming the API from potential vulnerabilities or data leaks.
7. Monitoring and Auditing: API security enables organizations to monitor and audit API activities, such as tracking API usage, detecting abnormal behaviour, or identifying potential security incidents. This visibility allows for timely response and remediation, reducing the impact of security breaches.

In summary, API security is essential to protect sensitive data, ensure authorized access, maintain trust and reputation, comply with regulations, prevent attacks, secure integrations, and enable monitoring and auditing. It is a fundamental aspect of building robust and secure applications and systems in today's interconnected digital landscape.

**4. AuthN and AuthZ**

Authentication (AuthN) and authorization (AuthZ) are two essential concepts in API security. Let's explore each of them with practical examples:

**Authentication (AuthN):** Authentication is the process of verifying the identity of a user or system. It ensures that the entity trying to access a resource or perform an action is who they claim to be. Here are a few practical examples of authentication:

1. Username/Password Authentication: When you log into an application or website using a username and password, the system checks whether the entered credentials match the stored values associated with that user. If the credentials are valid, you are authenticated and granted access to the protected resources.
2. Token-Based Authentication: Many APIs use tokens for authentication. For instance, when you log into a mobile app using your social media account, the app receives an access token. The app includes this token with subsequent API requests to prove your identity and access protected resources. The API server verifies the token's validity before allowing access.
3. Two-Factor Authentication (2FA): 2FA adds an extra layer of security to authentication. In addition to a username and password, you may be required to provide a second factor, such as a unique code generated by an authentication app or received via SMS. This method ensures that even if someone obtains your password, they still cannot access the protected resource without the second factor.

**Authorization (AuthZ):** Authorization is the process of granting or denying access to specific resources or actions based on the authenticated user's privileges or permissions. Here are some practical examples of authorization:

1. Role-Based Access Control (RBAC): In RBAC, permissions are assigned based on roles. For example, in a content management system, an administrator role may have access to create, edit, and delete content, while a regular user role may have read-only access. When a user is authenticated, their role determines the actions they can perform.
2. Attribute-Based Access Control (ABAC): ABAC grants access based on various attributes associated with the user or the resource. For instance, in a healthcare system, a doctor may have access to medical records of their assigned patients based on their role as a doctor and the specific patient's attribute.
3. Scope-Based Access Control: APIs often implement scope-based access control, where each authenticated user is granted specific scopes or permissions. For example, in an email API, a user with "read" scope can only retrieve emails, while a user with "read" and "send" scopes can both read and send emails.
4. Fine-Grained Access Control: Fine-grained access control allows for precise control over individual resources. For instance, in a file-sharing application, you can define permissions at the file or folder level, specifying which users or groups can read, write, or delete specific files.

In summary, authentication (AuthN) is the process of verifying a user's identity, while authorization (AuthZ) determines the permissions and access rights granted to an authenticated user. These two concepts work together to ensure that only authenticated users with appropriate privileges can access the desired resources or perform specific actions.

**5. Rate limit**

API rate limiting is, in a nutshell, limiting access for people (and bots) to access the API based on the rules/policies set by the API's operator or owner. We can think of rate limiting as a form of both security and quality control. This is why rate limiting is integral for any API product's growth and scalability.

**6. API Gateway**

**API Security Labs and Practices**

1. [OWASP crAPI](https://github.com/OWASP/crAPI): Completely Ridiculous API (crAPI) can help teams understand the ten most important security aspects of an API within a mock environment. crAPI has implemented almost every security loophole that APIs should not have—this offers a good model that showcases how not to secure APIs.

crAPI uses a microservices architecture and is composed of several services which are developed using the following:

* Identity: user and authentication endpoints
* Web: main Ingress service
* Community: community blogs and comments endpoints
* Mailhog: mail service
* Workshop—vehicle workshop endpoints
* Postgres—SQL Database
* Mongo—NoSQL Database