Web Service and API testing approach

Agenda

- Why do we need API Testing?
- What is Web Service and API?
- How API Works?
- Example
- OWASP Top 10 API Security
- Attacks on Web services
- Tools for API Testing

Why do we need API testing?

APIs now serve as the primary interface to application logic and because GUI **tests** are difficult to maintain with the short release cycles and frequent changes commonly used with Agile software development and DevOps.

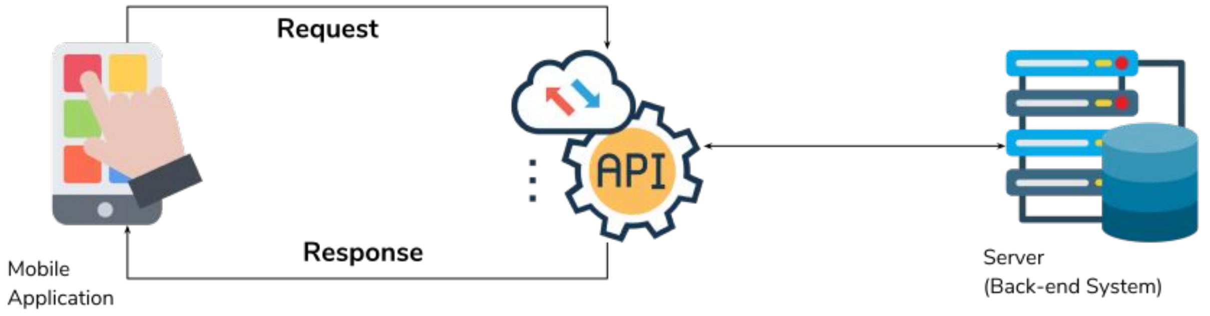
What is Web Service And API?

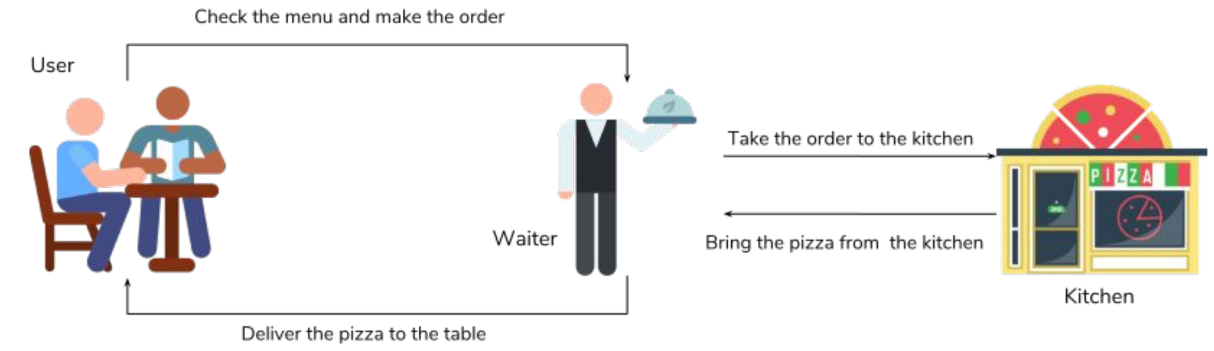
- A Web service is a standardized way of establishing communication between two Web-based applications by using open standards over an internet protocol backbone.
- Generally web applications work using HTTP and HTML, but web services work using HTTP and XML. Which as added some advantages over web applications. HTTP is transfer independent and XML is data independent.
- API is a set of functions and procedures that allow the creation of applications which access the features or data of an operating system, application, or other service

How API Works?

When you are using a mobile application, it connects to the internet and sends your request to the server. The server then collects the data from your request, process them and send back to your mobile phone. The application will convert that data into a readable response and display that through the user interface.

Example -


Request
Response
API
Mobile Application
Server (Back-end System)


Check the menu and make the order
User
Waiter
Take the order to the kitchen
Bring the pizza from the kitchen
PIZZA
Kitchen
Deliver the pizza to the table

OWASP Top 10 API Security

1. Broken Object Level Authorization
   - APIs tend to expose endpoints that handle object identifiers, creating a wide attack surface Level Access Control issue. Object level authorization checks should be considered in every function that accesses a data source using an input from the user.
2. Broken User Authentication
   - Authentication mechanisms are often implemented incorrectly, allowing attackers to compromise authentication tokens or to exploit implementation flaws to assume other user’s identities temporarily or permanently. Compromising a system’s ability to identify the client/user, compromises API security overall.

3. Excessive Data Exposure

   - Looking forward to generic implementations, developers tend to expose all object properties without considering their individual sensitivity, relying on clients to perform the data filtering before displaying it to the user.

4. Lack of Resources & Rate Limiting

   - Quite often, APIs do not impose any restrictions on the size or number of resources that can be requested by the client/user. Not only can this impact the API server performance, leading to Denial of Service (DoS), but also leaves the door open to authentication flaws such as brute force.

5. Broken Function Level Authorization

   - Complex access control policies with different hierarchies, groups, and roles, and an unclear separation between administrative and regular functions, tend to lead to authorization flaws. By exploiting these issues, attackers gain access to other users' resources and/or administrative functions.

6. Mass Assignment

- Binding client provided data (e.g., JSON) to data models, without proper properties filtering based on an allowlist, usually leads to Mass Assignment. Either guessing objects properties, exploring other API endpoints, reading the documentation, or providing additional object properties in request payloads, allows attackers to modify object properties they are not supposed to.

7. Security Misconfiguration

- Security misconfiguration is commonly a result of insecure default configurations, incomplete or ad-hoc configurations, open cloud storage, misconfigured HTTP headers, unnecessary HTTP methods, permissive Cross-Origin resource sharing (CORS), and verbose error messages containing sensitive information.

8. Injection

- Injection flaws, such as SQL, NoSQL, Command Injection, etc., occur when untrusted data is sent to an interpreter as part of a command or query. The attacker's malicious data can trick the interpreter into executing unintended commands or accessing data without proper authorization.

9. Improper Assets Management

- APIs tend to expose more endpoints than traditional web applications, making proper and updated documentation highly important. Proper hosts and deployed API versions inventory also play an important role to mitigate issues such as deprecated API versions and exposed debug endpoints.

10. Insufficient Logging & Monitoring

- Insufficient logging and monitoring, coupled with missing or ineffective integration with incident response, allows attackers to further attack systems, maintain persistence, pivot to more systems to tamper with, extract, or destroy data. Most breach studies demonstrate the time to detect a breach is over 200 days, typically detected by external parties rather than internal processes or monitoring.

Possible Attacks

- SQL INJECTION
- CROSS-SITE SCRIPTING
- BILLION LAUGHS ATTACK / XML BOMB/XML ENTITY EXPANSION (XEE)
- XML EXTERNAL ENTITY (XXE) INJECTION
- COMMAND INJECTION
- AUTHENTICATION BYPASS
- ERROR HANDLING

Tools for API Testing

- Soap UI
- Postman
- Katalon Studio
- Apigee
- API Fortress
- JMeter
- .etc

Any Questions?

Thank You!!