API APPLICATION PENTESTING

RESEARCH ON API AND ITS TYPES

APIs (Application Programming Interfaces) are crucial for enabling software applications to interact and integrate with each other. Here's a detailed overview of APIs, including their purpose, key concepts, types, and usage.

Overview of APIs

Definition

An API is a set of rules and protocols that allows different software applications to communicate with each other. It defines methods and data formats for requests and responses, enabling developers to use predefined functions or services without having to understand their internal workings.

Purpose

Interoperability: APIs allow different systems, applications, or components to work together, regardless of their underlying technologies.

Abstraction: They provide a simplified interface to complex functionalities, hiding the implementation details.

Modularity: APIs support modular design by allowing separate components to interact, promoting reusable code and scalable architecture.

Key Concepts

1. Endpoints: Specific URLs provided by an API where requests are sent. Each endpoint represents a particular function or resource.

2. Methods: Actions performed on endpoints. In RESTful APIs, common methods include:

GET: Retrieve data.

POST: Create new data.

PUT: Update existing data.

DELETE: Remove data.

3. Request and Response:

Request: Consists of the HTTP method, endpoint, headers, and body (if applicable).

Response: Consists of a status code (indicating the success or failure of the request), headers, and a body (containing the requested data or an error message).

4. Authentication and Authorization:

Authentication: Verifying the identity of a user or application (e.g., through API keys, OAuth tokens).

Authorization: Determining what an authenticated user or application is allowed to do.

5. Rate Limiting: Mechanism to control the number of requests a user or application can make to an API within a certain time frame, preventing abuse and ensuring fair usage.

6. Error Handling: APIs provide error codes and messages to help developers diagnose and handle issues. Common error codes include 400 (Bad Request), 401 (Unauthorized), 404 (Not Found), and 500 (Internal Server Error).

Types of APIs

1. Web APIs:

REST (Representational State Transfer): Uses standard HTTP methods and is stateless. Data is often exchanged in JSON or XML format. RESTful APIs are simple, scalable, and widely used.

SOAP (Simple Object Access Protocol): A protocol using XML for message format and HTTP or SMTP for message transmission. SOAP APIs are known for their strict standards and built-in error handling but can be more complex.

GraphQL: A query language for APIs that allows clients to request exactly the data they need. Developed by Facebook, GraphQL is efficient for complex queries and reduces over-fetching and under-fetching of data.

2. Library APIs:

Standard Libraries: APIs provided by programming languages (e.g., Python's standard library, Java's JDK) that offer built-in functions and classes for common programming tasks.

Third-Party Libraries: APIs from external libraries or frameworks used for specific functionalities (e.g., data analysis, machine learning).

3. Operating System APIs:

Provide access to OS functionalities such as file management and system calls. Examples include Windows API, POSIX, and macOS APIs.

4. Database APIs:

Enable interaction with databases. SQL-based APIs are used for relational databases, while NoSQL APIs are used for non-relational databases (e.g., MongoDB).

5. Hardware APIs:

Provide access to hardware components like sensors or cameras. Used in embedded systems and mobile devices (e.g., Android camera API).

6. Service APIs:

Cloud Service APIs: Allow interaction with cloud services for storage, computing, and other functions (e.g., AWS SDKs, Google Cloud APIs).

Payment Gateway APIs: Facilitate payment processing in applications (e.g., Stripe, PayPal).

7. Internal APIs:

Micro services APIs: Used within micro services architectures to enable communication between different micro services.

Enterprise APIs: Used to integrate internal systems and data sources within an organization.

Use Cases

Integration: Connecting disparate systems, such as integrating a CRM with an email marketing

9platform.

Extension: Adding new functionalities to existing applications, like integrating third-party payment gateways.

Automation: Automating repetitive tasks by leveraging APIs, such as syncing data between applications.

Data Access: Accessing and manipulating data from various sources, including databases and web services.

Best Practices

1. Documentation: Comprehensive and clear API documentation is essential for ease of use and integration.

2. Versioning: Manage changes and updates to APIs by implementing versioning to maintain compatibility.

3. Security: Implement robust security measures, including authentication, authorization, and encryption, to protect data and access.

4. Testing: Thoroughly test APIs for functionality, performance, and security to ensure they meet quality standards.

APIs are fundamental to modern software development, enabling seamless integration and interaction between different systems, applications, and services. They support a wide range of functionalities, from basic data retrieval to complex service interactions, playing a critical role in the development of scalable and efficient applications.

RESEARCH ON API ARCHITECTURE AND ITS TYPES

API architecture is a framework that defines how APIs are designed, implemented, and interacted with. It encompasses the structural design of APIs, the principles guiding their development, and the methodologies used to ensure they are effective, scalable, and maintainable. Here's an in-depth look at API architecture:

Key Aspects of API Architecture

1. Design Principles

Simplicity: APIs should be intuitive and easy to use. A well-designed API has clear, consistent naming conventions and straightforward endpoints.

Consistency: Adherence to standard conventions and practices across the API ensures that developers can predict behavior and integrate more easily.

Modularity: APIs should be modular, meaning they provide distinct functionalities that can be combined or used independently. This promotes reusability and scalability.

Scalability: The API architecture should support scaling, handling increasing loads and traffic without degrading performance.

Security: Incorporating security best practices, including authentication, authorization, and data encryption, to protect API interactions.

2. Architectural Styles

REST (Representational State Transfer)

Concepts: RESTful APIs are based on stateless operations and standard HTTP methods (GET, POST, PUT, DELETE). They use URIs to identify resources and communicate data in formats like JSON or XML.

Benefits: Simplicity, scalability, and ease of integration with web services. REST is widely used due to its alignment with the principles of HTTP.

Constraints: RESTful APIs are stateless, meaning that each request from a client must contain all the information needed to process the request.

SOAP (Simple Object Access Protocol)

Concepts: SOAP APIs use XML for message formatting and rely on HTTP or other protocols for message transmission. They are based on a strict protocol and often include built-in error handling.

Benefits: Well-defined standards and robust security features. SOAP supports complex transactions and high levels of reliability.

Constraints: Complexity and verbosity due to XML formatting and the need for strict adherence to standards.

GraphQL

Concepts: Developed by Facebook, GraphQL allows clients to specify exactly what data they need. It uses a single endpoint and supports complex queries, real-time updates via subscriptions, and mutations for data changes.

Benefits: Flexibility in querying, reducing over-fetching and under-fetching of data. Ideal for complex systems with multiple interconnected resources.

Constraints: More complex to set up and maintain compared to REST, especially with regard to query optimization and security.

gRPC (gRPC Remote Procedure Calls)

Concepts: Developed by Google, gRPC uses HTTP/2 for transport, Protocol Buffers for serialization, and supports multiple languages. It allows clients and servers to communicate via defined methods and data structures.

Benefits: Efficient communication, support for streaming, and strong typing. Suitable for micro services and environments requiring high performance and low latency.

Constraints: Steeper learning curve and requires the use of Protocol Buffers or other serialization methods.

3. Design Patterns

Resource-Oriented Architecture: Emphasizes the use of resources (e.g., users, orders) and standard HTTP methods to operate on these resources.

Service-Oriented Architecture (SOA): Focuses on designing services that interact over a network. SOA is broader than APIs but often involves API use for service interactions.

Event-Driven Architecture: APIs that interact based on events rather than requests. Useful for real-time applications and asynchronous processing.

Micro services Architecture: Involves breaking down an application into smaller, self-contained services that communicate via APIs. Each micro service handles a specific business function.

4. API Documentation

Purpose: Provides developers with the necessary information to use and integrate with the API. Good documentation includes endpoint descriptions, request/response formats, authentication details, and examples.

Tools: Swagger/OpenAPI, Postman, and Redoc are popular tools for creating and managing API documentation.

5. Versioning

Concepts: API versioning is crucial for maintaining backward compatibility as APIs evolve. Common strategies include URL versioning (e.g., `/api/v1/resource`), header versioning, and query parameter versioning.

Best Practices: Ensure that version changes do not break existing integrations and provide clear deprecation warnings and transition paths.

6. Security Considerations

Authentication: Verifies the identity of users or applications (e.g., OAuth, API keys, JWT tokens).

Authorization: Controls access to resources based on user roles or permissions.

Encryption: Protects data in transit (e.g., using HTTPS) and at rest.

7. Performance Optimization

Caching: Reduces server load and improves response times by storing and reusing responses for identical requests.

Pagination: Handles large datasets by breaking them into smaller chunks, reducing response times and memory usage.

Rate Limiting: Prevents abuse and ensures fair usage by limiting the number of requests from a client within a specified timeframe.

8. Testing and Monitoring

Testing: Ensure APIs are robust, reliable, and meet performance expectations. Includes unit tests, integration tests, and load tests.

Monitoring: Track API performance, availability, and usage metrics to identify and resolve issues proactively. Tools like Prometheus, Grafana, and New Relic are commonly used.

Summary

API architecture encompasses the design principles, styles, patterns, and best practices for creating effective APIs. Understanding and implementing these aspects helps in building APIs that are functional, secure, and scalable, catering to diverse integration needs and ensuring seamless interaction between systems.

TYPES OF API ARCHITECTURE

API architecture refers to the design and structure of how APIs are created, organized, and interacted with. Various architectural styles and patterns guide the development of APIs, each suited to different needs and use cases. Here’s an overview of the most prominent types of API architecture:

1. REST (Representational State Transfer)

Concepts:

1. **Stateless Communication:** Each request from a client to server must contain all the information needed to understand and process the request.
2. **Resource-Based:** Interactions are based on resources (e.g., users, orders) identified by URIs (Uniform Resource Identifiers).
3. **HTTP Methods:** Commonly uses GET (retrieve), POST (create), PUT (update), and DELETE (remove) methods.

Benefits:

1. Simplicity and ease of use with standard HTTP methods.
2. Scalable and stateless, which helps in performance and reliability.
3. Widely supported and easily integrated with web services.

Constraints:

1. Limited to the capabilities of HTTP methods.
2. Not ideal for complex query requirements, which can result in over-fetching or under-fetching data.

2. SOAP (Simple Object Access Protocol)

Concepts:

1. Protocol-Based: Uses XML-based messaging protocol and relies on HTTP or other application protocols.
2. Strict Standards: Enforces a formal structure with strict rules, including message formats and error handling.

Benefits:

1. Well-defined standards for security and transaction management.
2. Reliable for complex transactions and enterprise-level integrations.

Constraints:

1. Verbosity and complexity due to XML formatting.
2. Overhead associated with strict adherence to standards and protocols.

3. GraphQL

Concepts:

1. Query Language: Allows clients to specify exactly what data they need, minimizing over-fetching and under-fetching.
2. Single Endpoint: Operates through a single endpoint that handles queries and mutations (data modifications).
3. Real-Time Capabilities: Supports subscriptions for real-time updates.

Benefits:

1. Flexibility in querying and efficient data retrieval.
2. Better suited for complex queries and dynamic data requirements.

Constraints:

1. More complex to implement compared to REST.
2. Requires careful planning to avoid performance issues with complex queries.

4. gRPC (gRPC Remote Procedure Calls)

Concepts:

1. HTTP/2 Transport: Utilizes HTTP/2 for efficient communication and supports multiplexed streams, header compression, and more.
2. Protocol Buffers: Uses Protocol Buffers (protobuf) for serialization, which is efficient and language-neutral.
3. Service Definition: Defines APIs using service definitions and remote procedure calls.

Benefits:

1. High performance and low latency due to binary serialization and efficient transport.
2. Supports bidirectional streaming and multiplexing.

Constraints:

1. Requires using Protocol Buffers for data serialization, which may introduce a learning curve.
2. Less human-readable compared to JSON used in REST.

5. RPC (Remote Procedure Call)

Concepts:

1. Procedure Call: APIs are defined in terms of methods or functions that can be invoked remotely.
2. Synchronous or Asynchronous: Supports both synchronous and asynchronous communication styles.

Benefits:

1. Simplifies interactions by exposing methods that can be called directly.
2. Can be more straightforward for certain types of applications, such as internal services.

Constraints:

1. Less flexible compared to REST and GraphQL for interacting with web resources.
2. May not be as widely supported or standardized as other approaches.

6. OData (Open Data Protocol)

Concepts:

1. REST-Based: Extends REST with query options and metadata capabilities.
2. Data Querying: Provides a standard way to query and manipulate data using URL conventions.

Benefits:

1. Standardized querying capabilities and data manipulation.
2. Integration with various data sources and services.

Constraints:

1. Adds complexity compared to basic REST.
2. Less commonly used compared to REST, GraphQL, or gRPC.

7. Event-Driven Architecture

Concepts:

1. Event-Based: APIs operate based on events rather than direct requests. Events trigger actions or responses.
2. Asynchronous: Suitable for systems that require real-time updates and processing.

Benefits:

1. Ideal for real-time applications and asynchronous processing.
2. Decouples producers and consumers, allowing for flexible and scalable interactions.

Constraints:

1. Complexity in managing event flows and ensuring message delivery.
2. Requires robust handling of event states and potential retries.

8. Microservices Architecture

Concepts:

1. Service-Oriented: Breaks down an application into smaller, self-contained services that interact via APIs.
2. Independent Deployability: Each microservice can be developed, deployed, and scaled independently.

Benefits:

1. Promotes modularity and scalability.
2. Allows for independent development and deployment of services.

Constraints:

1. Increased complexity in managing inter-service communication and data consistency.
2. Requires comprehensive monitoring and coordination between services.

Summary

API architecture plays a critical role in defining how APIs are structured and interact with clients and other services. Each architectural style or pattern—REST, SOAP, GraphQL, gRPC, RPC, OData, event-driven, and microservices—offers distinct advantages and is suited to different types of applications and integration needs. Choosing the right architecture depends on factors like the complexity of data interactions, performance requirements, and specific use cases.

EXPLAINATION OF REST AND SOAP API

Certainly! REST and SOAP are two popular architectural styles for designing web APIs, each with its own characteristics, advantages, and use cases. Here's a detailed explanation of each:

REST (Representational State Transfer)

Concepts:

1. Resource-Based: In REST, resources (e.g., users, products) are identified by URLs (Uniform Resource Locators). Each URL represents a specific resource.
2. HTTP Methods: RESTful APIs use standard HTTP methods to perform operations on resources:
   GET: Retrieve data from the server.
   POST: Create a new resource on the server.

PUT: Update an existing resource on the server.
DELETE: Remove a resource from the server.

3. Stateless: Each request from a client to the server must contain all the information needed to understand and process the request. The server does not store any state about the client between requests.
4. Representation: Resources can be represented in various formats, commonly JSON or XML. The representation is what is transferred between client and server.
5. HATEOAS (Hypermedia as the Engine of Application State): A RESTful API can provide links to other related resources or actions that the client can perform. This principle helps in navigating the API dynamically.

Advantages:

1. Simplicity: REST APIs are easy to understand and use due to their reliance on standard HTTP methods and status codes.
2. Scalability: Stateless interactions and cacheable responses contribute to the scalability of REST APIs.
3. Flexibility: Supports multiple data formats and is not tied to a specific protocol.

Disadvantages:

1. Limited Functionality: Basic REST does not define a formal standard for complex transactions or security, which may require additional implementation.
2. Over-fetching and Under-fetching: Clients may receive more data than needed (over-fetching) or less data than required (under-fetching), especially with complex resource structures.

Example:

GET /users/123: Retrieve user data with ID 123.

POST /users: Create a new user with the data provided in the request body.

PUT /users/123: Update user data with ID 123.

DELETE /users/123: Delete user data with ID 123.

SOAP (Simple Object Access Protocol)

Concepts:

1. Protocol-Based: SOAP is a protocol for exchanging structured information using XML. It relies on a strict messaging format and can operate over various protocols like HTTP, SMTP, or more.
2. Envelope: A SOAP message is encapsulated in an XML envelope, which includes:
3. Header: Contains metadata and control information (optional).
4. Body: Contains the actual message or request/response payload.
5. Fault: An optional part of the envelope for error reporting.
6. WSDL (Web Services Description Language): SOAP services are described using WSDL, an XML-based language that defines the service's operations, parameters, and data types.
7. Stateful or Stateless: SOAP can be either stateful or stateless, depending on how the service is designed. The protocol itself does not enforce statelessness.
8. Formal Contracts: SOAP services define strict contracts and standards for communication, including message structure, data types, and error handling.

Advantages:

1. Robust Standards: SOAP includes built-in error handling, security, and transaction management features.
2. Strong Typing: The use of XML and WSDL ensures that data types and operations are clearly defined and enforced.
3. Interoperability: SOAP's strict standards ensure interoperability between different platforms and programming languages.

Disadvantages:

1. Complexity: SOAP's formal standards and XML messaging can be complex and verbose.
2. Performance Overhead: XML-based messaging can introduce additional overhead compared to more lightweight formats like JSON.

Example:

Request: A SOAP request to retrieve user information might look like this:

```xml
<soapenv:Envelope xmlns:soapenv="http://schemas.xmlsoap.org/soap/envelope/" xmlns:ex="http://example.com/">
  <soapenv:Header/>
  <soapenv:Body>
    <ex:GetUser>
      <ex:UserId>123</ex:UserId>
    </ex:GetUser>
  </soapenv:Body>
</soapenv:Envelope>
```

Response: A SOAP response might be:

```xml
<soapenv:Envelope xmlns:soapenv="http://schemas.xmlsoap.org/soap/envelope/" xmlns:ex="http://example.com/">
  <soapenv:Header/>
  <soapenv:Body>
    <ex:GetUserResponse>
      <ex:User>
        <ex:Id>123</ex:Id>
        <ex:Name>John Doe</ex:Name>
      </ex:User>
    </ex:GetUserResponse>
  </soapenv:Body>
</soapenv:Envelope>
```

~~~
```
~~~

Summary

REST is widely used for its simplicity, flexibility, and alignment with web standards. It is suitable for applications that benefit from a lightweight, stateless interaction model and can handle varying data formats.

SOAP is preferred in enterprise environments where robust standards, formal contracts, and built-in features like security and transaction management are required. It is well-suited for complex or critical applications where strict adherence to protocols and message formats is important.

RESEARCH ON API PENTESTING PROCESS

API penetration testing (pentesting) is a security assessment process designed to identify vulnerabilities and weaknesses in APIs (Application Programming Interfaces). This process involves simulating attacks to evaluate the security of APIs and ensure they are robust against various types of threats. Here's a detailed overview of the API pentesting process:

1. Preparation

1.1. Define Scope:

Identify APIs: List all APIs that need to be tested, including public and private endpoints.

Understand Functionality: Gather information on the API's functionality, including data it handles, authentication methods, and expected usage.

1.2. Obtain Permissions:

Authorization: Ensure you have explicit permission from the organization to conduct the pentest.

Legal Agreements: Complete any required legal agreements or contracts.

1.3. Gather Documentation:

API Documentation: Collect API documentation such as OpenAPI (formerly Swagger), WSDL for SOAP APIs, and other relevant materials.

Source Code (if available): Review source code to understand the underlying implementation.

2. Information Gathering

2.1. Discover Endpoints:

API Endpoints: Identify all API endpoints through documentation, code reviews, or network traffic analysis.

Dynamic Discovery: Use tools to discover hidden or undocumented endpoints (e.g., web crawling, directory brute-forcing).

2.2. Enumerate Parameters:

Parameters: Identify all input parameters, including path parameters, query parameters, headers, and request bodies.

Data Types and Constraints: Understand the expected data types and constraints for each parameter.

2.3. Understand Authentication and Authorization:

Authentication Methods: Identify authentication mechanisms (e.g., API keys, OAuth, JWT).

Authorization Controls: Determine how access control is implemented and whether there are any role-based or attribute-based access controls.

3. Vulnerability Analysis

3.1. Test for Common Vulnerabilities:

Authentication Flaws: Test for issues such as weak authentication mechanisms, exposed credentials, and session fixation.

Authorization Issues: Verify proper enforcement of access controls and test for unauthorized access to restricted resources.

Input Validation: Assess for issues like SQL injection, cross-site scripting (XSS), and command injection by providing various inputs.

Data Exposure: Check for sensitive data exposure in responses, such as personal information or debug information.

3.2. Assess API Security Features:

Rate Limiting: Ensure rate limiting is implemented to prevent abuse.

Rate Limiting: Ensure mechanisms are in place to prevent abuse.

Encryption: Verify that data is encrypted in transit (using HTTPS) and at rest if applicable.

Error Handling: Assess how errors are handled and whether error messages reveal sensitive information.

4. Attack Simulation

4.1. Fuzz Testing:

Automated Tools: Use fussing tools to provide a large volume of random or malformed data to API endpoints.

Manual Fuzzing: Manually test endpoints with unexpected or malicious inputs.

4.2. Session Management:

Session Hijacking: Test for vulnerabilities related to session management, such as the ability to hijack or reuse sessions.

Token Manipulation: Test for weaknesses in token handling, including token expiry and revocation issues.

4.3. Business Logic Testing:

Workflow Manipulation: Test for vulnerabilities that might arise from business logic flaws, such as improper sequence of actions or incorrect implementation of business rules.

5. Reporting

5.1. Document Findings:

Vulnerabilities: Document all identified vulnerabilities, including their severity, risk, and impact.

Evidence: Provide evidence of vulnerabilities, such as screenshots, logs, and detailed descriptions of the testing process.

5.2. Recommendations:

Mitigation: Offer recommendations for fixing vulnerabilities and improving security.

Best Practices: Suggest security best practices, such as implementing secure coding practices and regular security reviews.

5.3. Present Findings:

Report: Prepare a comprehensive report that outlines findings, recommendations, and action plans.

Presentation: Present the findings to relevant stakeholders, such as developers, security teams, and management.

6. Remediation and Verification

6.1. Fix Vulnerabilities:

Development: Work with developers to address and fix identified vulnerabilities.

Testing: Verify that vulnerabilities have been fixed and that no new issues have been introduced.

6.2. Re-Test:

Follow-Up Testing: Conduct re-testing to ensure that the fixes are effective and that the API is secure.

6.3. Continuous Improvement:

Ongoing Assessment: Implement regular security assessments and updates to maintain API security over time.

Training: Provide training to development teams on secure coding practices and API security.

Summary

API penetration testing is a critical process for ensuring the security and robustness of APIs. It involves preparation, information gathering, vulnerability analysis, attack simulation, reporting, and remediation. By identifying and addressing vulnerabilities, organizations can protect their APIs from potential attacks and ensure the safety of their data and services.

REFERENCES:

https://www.getastra.com/blog/security-audit/api-penetration-testing/

https://www.catchpoint.com/api-monitoring-tools/api-architecture

https://zikazaki.medium.com/top-6-api-architecture-styles-for-modern-software-development-d35752c7f9aa

https://aws.amazon.com/compare/the-difference-between-soap-rest/

Practical

Introduction and Installation of Postman Tool:

Postman is a popular tool used for API development and testing. It provides a user-friendly interface that simplifies the process of sending requests, receiving responses, and analyzing API interactions. Postman is widely used by developers, QA engineers, and other professionals involved in the API lifecycle.

Key Features of Postman

1. Request Building and Sending:

HTTP Methods: Supports various HTTP methods such as GET, POST, PUT, DELETE, PATCH, and more.

Request Composition: Allows users to compose requests with headers, query parameters, request bodies, and authentication credentials.

Environment Variables: Supports the use of variables to manage different environments (e.g., development, testing, and production) and simplify request management.

2. Testing and Automation:

Test Scripts: Users can write test scripts in JavaScript to automate tests and validate API responses. Postman provides a built-in test runner to execute these scripts.

Collection Runner: Allows running a series of requests from a collection with specific parameters and test scripts.

Continuous Integration (CI): Integrates with CI/CD tools to automate API tests as part of the development pipeline.

3. Documentation:

API Documentation: Automatically generates and publishes API documentation from Postman collections. This documentation can be shared with team members or external users.

Collection Export: Users can export collections in various formats (e.g., JSON) to share or integrate with other tools.

4. Collaboration:

Workspaces: Facilitates team collaboration by organizing requests, collections, and environments into shared workspaces.

Version Control: Supports versioning of collections and environments to track changes and collaborate effectively with team members.

5. Monitoring:

Postman Monitors: Allows scheduling and running tests at regular intervals to monitor API performance and availability. Monitors can send notifications if issues are detected.

6. Mock Servers:

Mocking API Endpoints: Users can create mock servers to simulate API responses and test how applications interact with APIs before they are fully developed.

7. Code Generation:

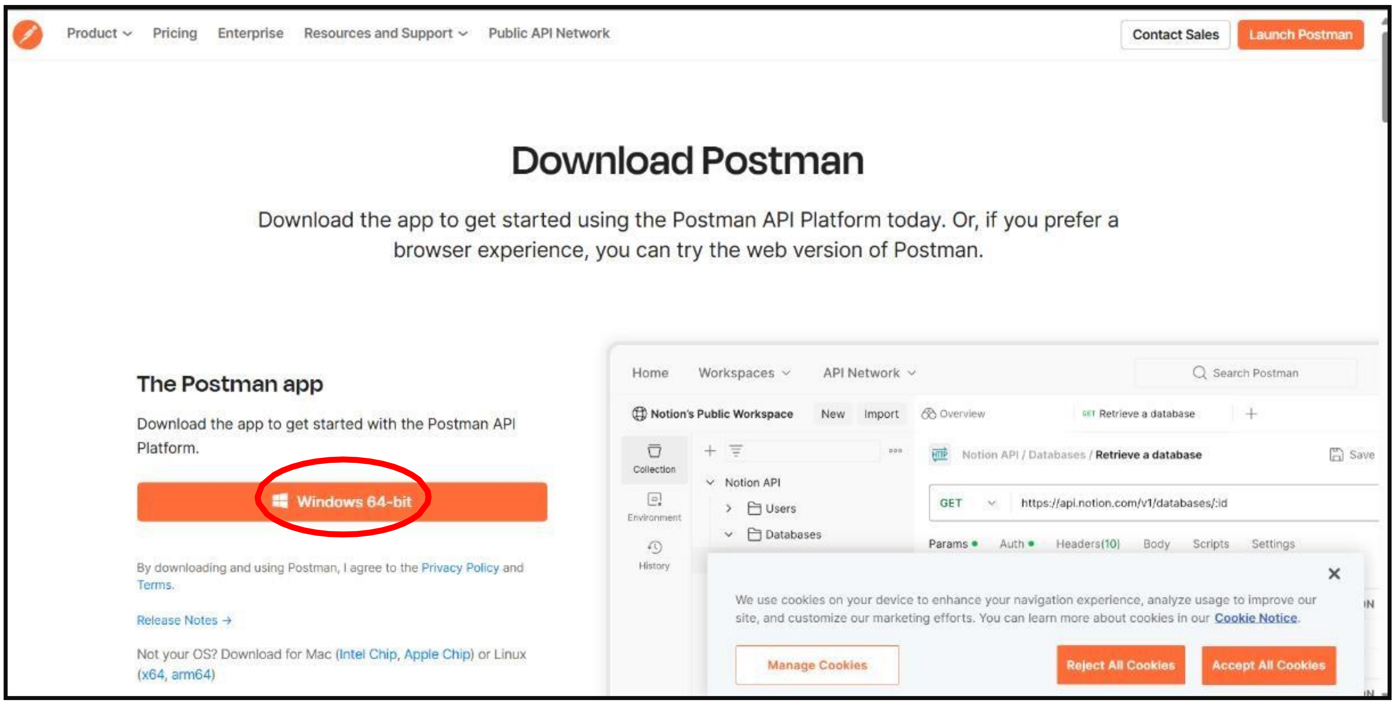
Client Code Snippets: Generates client code snippets in various programming languages (e.g., Python, JavaScript, Java) for integrating API requests into applications.

Summary

Postman is a comprehensive tool designed to facilitate API development, testing, and documentation. Its user-friendly interface, support for various HTTP methods, automation capabilities, and collaborative features make it a valuable resource for developers and teams working with APIs. Whether you're creating new APIs, validating existing ones, or documenting them for future use, Postman provides the tools and functionality needed to streamline the process.

To install Postman tool:

1. Visit this sitehttps://www.postman.com/downloads/
2. Download the Setup file by clicking on the “Windows64-bit” button.


Product Pricing Enterprise Resources and Support Public API Network

Contact Sales Launch Postman

Download Postman

Download the app to get started using the Postman API Platform today. Or, if you prefer a browser experience, you can try the web version of Postman.

The Postman app

Download the app to get started with the Postman API Platform.

Windows 64-bit

By downloading and using Postman, I agree to the Privacy Policy and Terms.

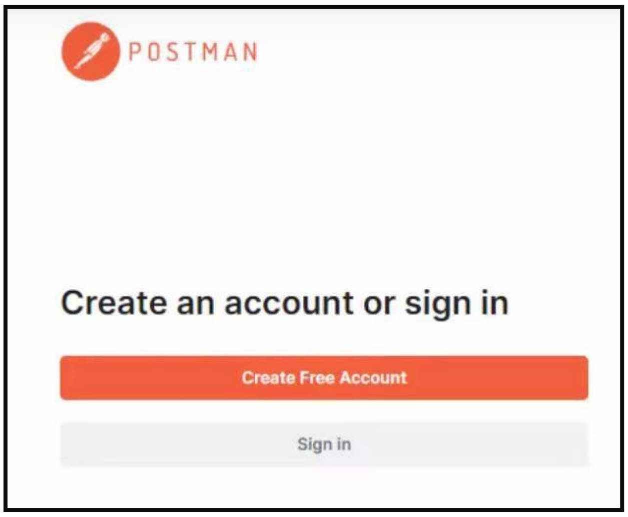
Release Notes →

Not your OS? Download for Mac (Intel Chip, Apple Chip) or Linux (x64, arm64)

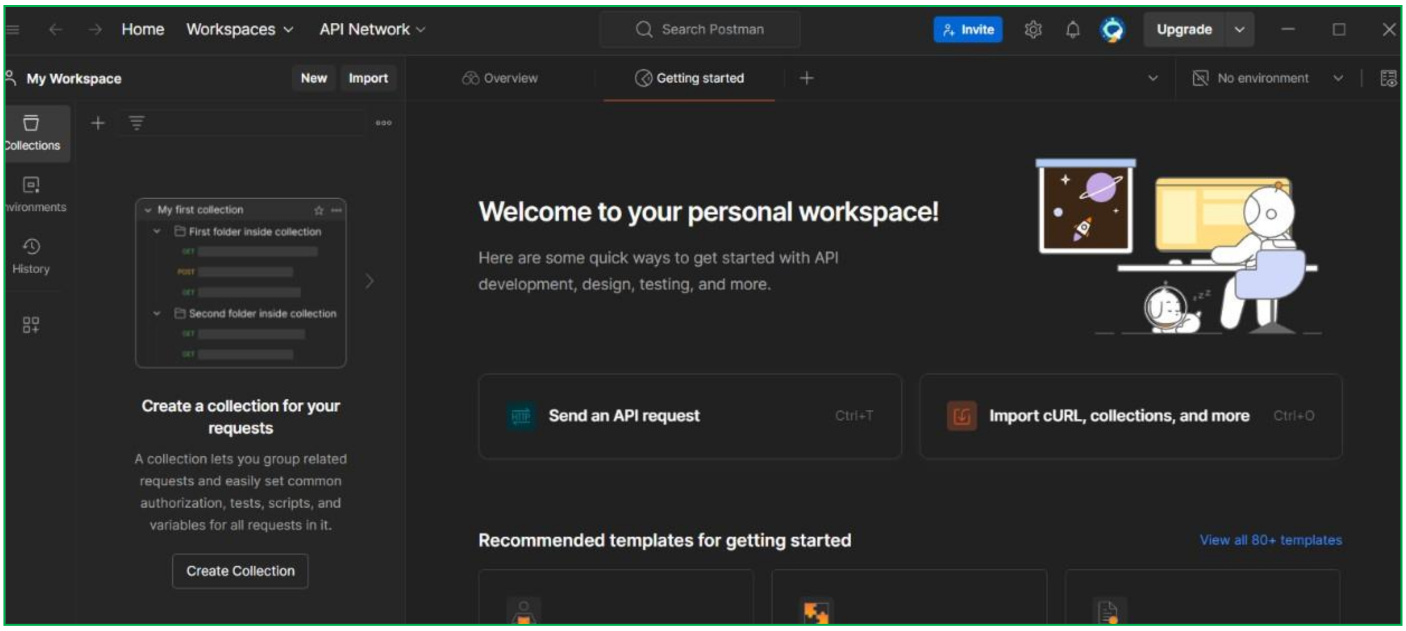
We use cookies on your device to enhance your navigation experience, analyze usage to improve our site, and customize our marketing efforts. You can learn more about cookies in our Cookie Notice.

Manage Cookies | Reject All Cookies | Accept All Cookies

1. Open the download ed file and sign up if you are new user or directly login.



2. You are ready to use.



Welcome to your personal workspace!

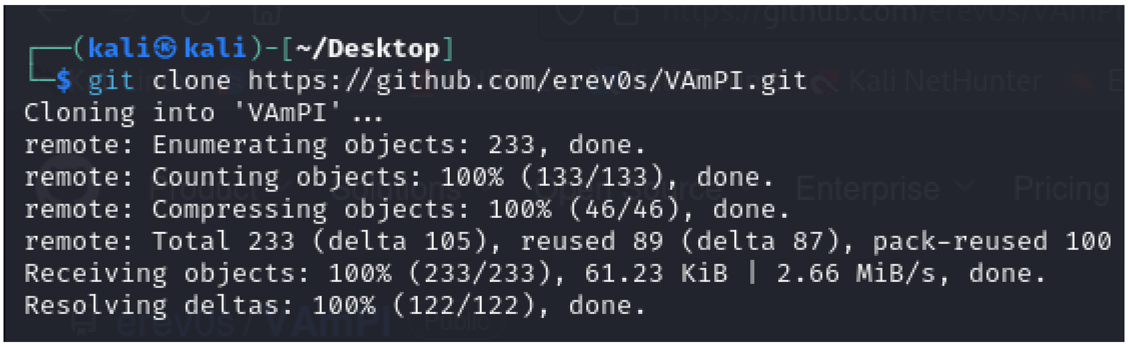
Here are some quick ways to get started with API development, design, testing, and more.

A collection lets you group related requests and easily set common authorization, tests, scripts, and variables for all requests in it.

Recommended templates for getting started

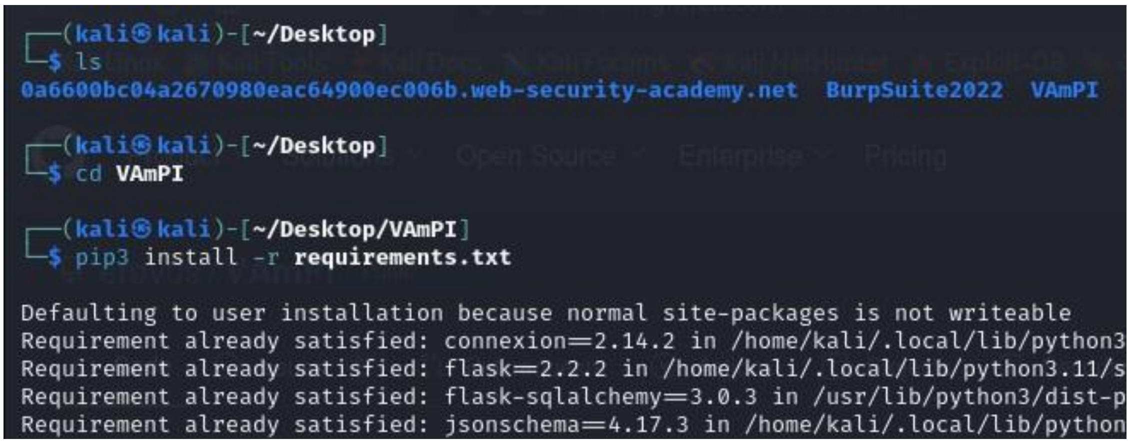
Lab Setup:(Our Lab is https://github.com/erev0s/VAmPI)

1. In the kali terminal git clone this repository https://github.com/erev0s/VAmPI



```
┌──(kali㉿kali)-[~/Desktop]
└─$ git clone https://github.com/erev0s/VAmPI.git
Cloning into 'VAmPI'...
remote: Enumerating objects: 233, done.
remote: Counting objects: 100% (133/133), done.
remote: Compressing objects: 100% (46/46), done.
remote: Total 233 (delta 105), reused 89 (delta 87), pack-reused 100
Receiving objects: 100% (233/233), 61.23 KiB | 2.66 MiB/s, done.
Resolving deltas: 100% (122/122), done.
```

2. Head to the file directory and run the following command to install all the files listed in requirements.txt “pip3install-rrequirements.txt”.



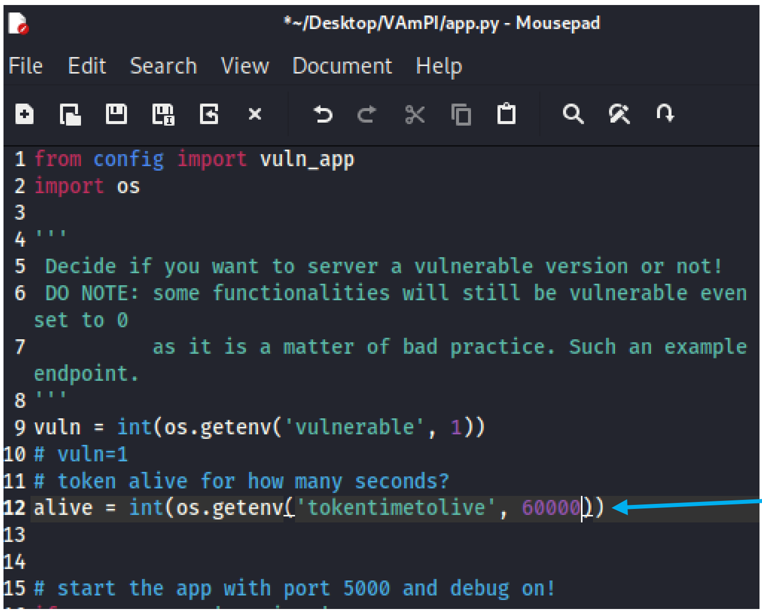
```
┌──(kali㉿kali)-[~/Desktop]
└─$ ls
0a6600bc04a2670980eac64900ec006b.web-security-academy.net  BurpSuite2022  VAmPI

┌──(kali㉿kali)-[~/Desktop]
└─$ cd VAmPI

┌──(kali㉿kali)-[~/Desktop/VAmPI]
└─$ pip3 install -r requirements.txt

Defaulting to user installation because normal site-packages is not writeable
Requirement already satisfied: connexion==2.14.2 in /home/kali/.local/lib/python3
Requirement already satisfied: flask==2.2.2 in /home/kali/.local/lib/python3.11/s
Requirement already satisfied: flask-sqlalchemy==3.0.3 in /usr/lib/python3/dist-p
Requirement already satisfied: jsonschema==4.17.3 in /home/kali/.local/lib/python
```

3. Now in the same directory, edit the token value from 60 to any higher value in app.py file to increase the life span of token.



*~/Desktop/VAmPI/app.py - Mousepad
File Edit Search View Document Help

```
from config import vuln_app
import os

'''
 Decide if you want to server a vulnerable version or not!
 DO NOTE: some functionalities will still be vulnerable even set to 0
          as it is a matter of bad practice. Such an example endpoint.
'''
vuln = int(os.getenv('vulnerable', 1))
# vuln=1
# token alive for how many seconds?
alive = int(os.getenv('tokentimetolive', 60000))


# start the app with port 5000 and debug on!
```

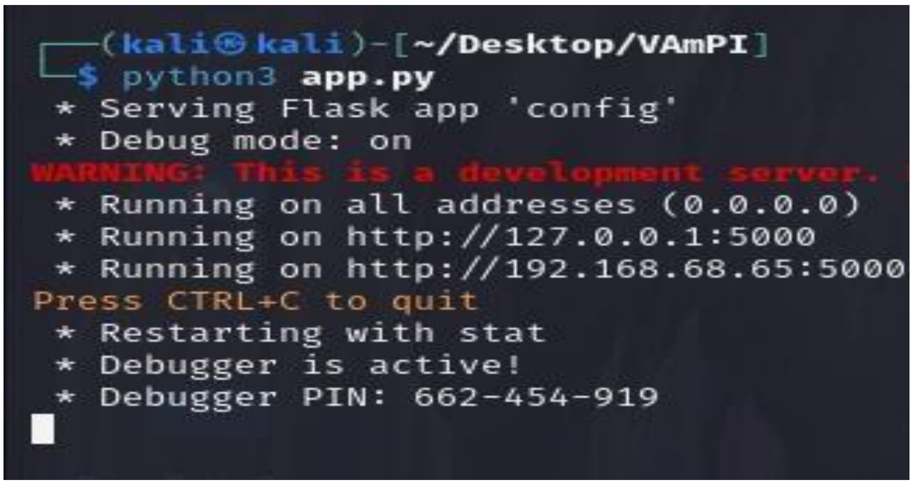
I’veinserted 60000 in place of 60

Tip:

Use this command in terminal to edit files

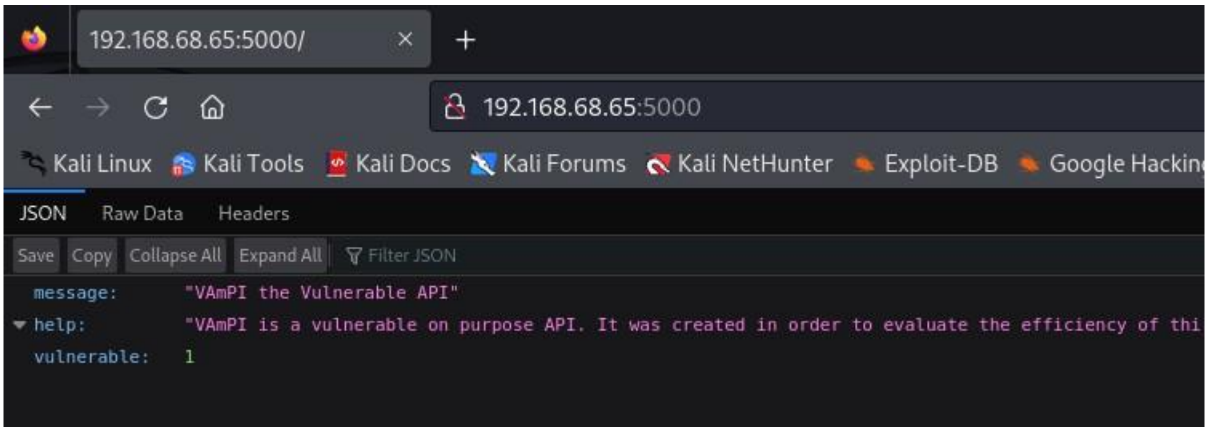
“mousepad<filename>.extension”E.g.:mousepad app.py

4. Now run the following command to run app.py“python3app.py”



```
┌──(kali㉿kali)-[~/Desktop/VAmPI]
└─$ python3 app.py
 * Serving Flask app 'config'
 * Debug mode: on
WARNING: This is a development server.
 * Running on all addresses (0.0.0.0)
 * Running on http://127.0.0.1:5000
 * Running on http://192.168.68.65:5000
Press CTRL+C to quit
 * Restarting with stat
 * Debugger is active!
 * Debugger PIN: 662-454-919
```

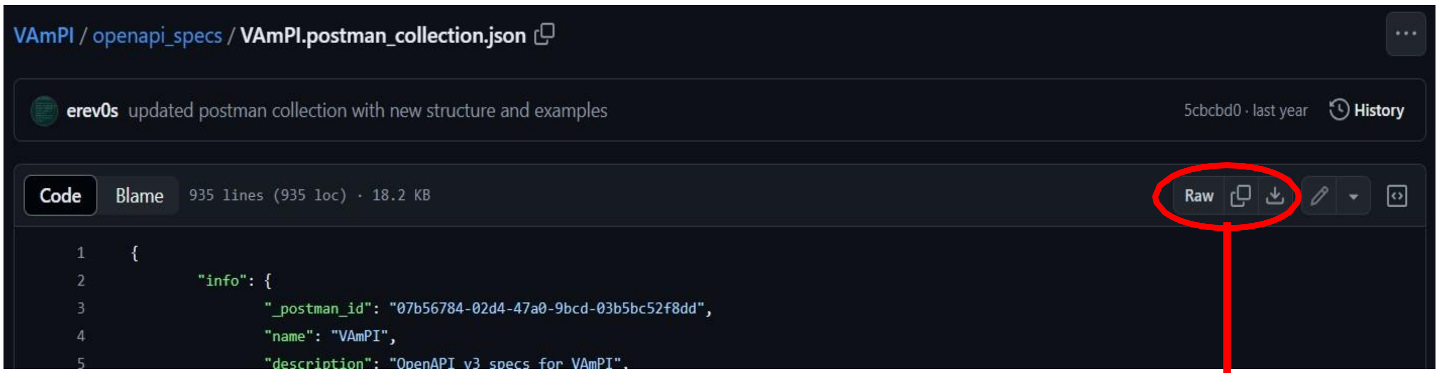
5. Copy the second IP along with the Port its running on & paste it in the browser. You’ll find Vampi API running.



```
message:      "VAmPI the Vulnerable API"
help:         "VAmPI is a vulnerable on purpose API. It was created in order to evaluate the efficiency of thi
vulnerable:   1
```

6. To access it in the postman tool, head back to the Git Hub page and go to openapi_specs folder.

You can either download or copy the raw code from the “.json” file.



VAmPI / openapi_specs / VAmPI.postman_collection.json
erev0s updated postman collection with new structure and examples
5cbcbd0 · last year
History
Code
Blame
935 lines (935 loc) · 18.2 KB
Raw

```
{
        "info": {
                "_postman_id": "07b56784-02d4-47a0-9bcd-03b5bc52f8dd",
                "name": "VAmPI",
                "description": "OpenAPI v3 specs for VAmPI",
```

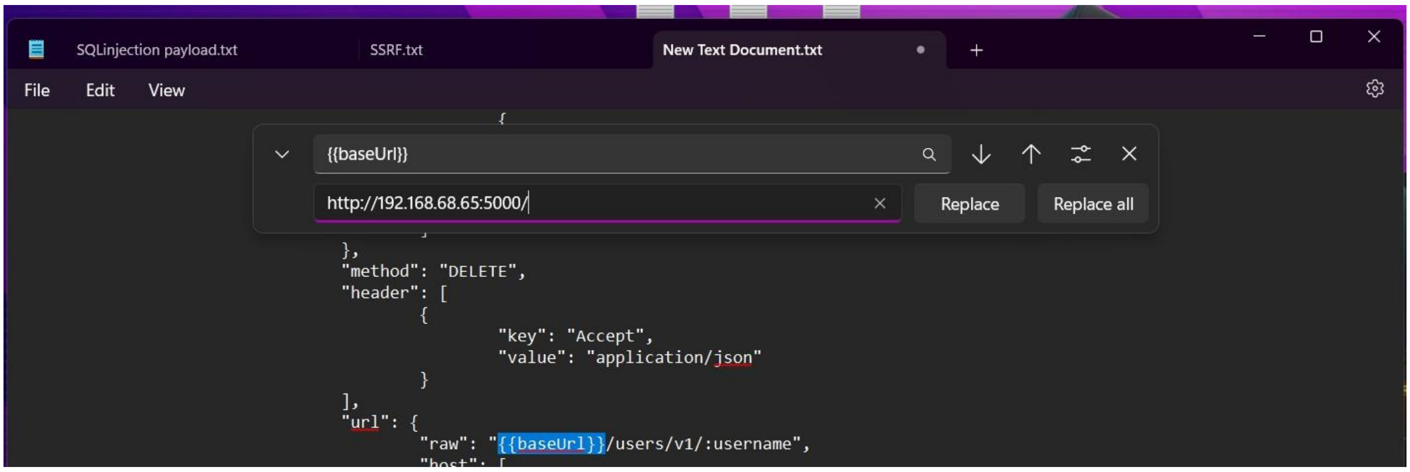
ownloadorcopy here

7. Create a new text file anywhere in your desktop and paste the copied code into it.

(Do this step only if you didn't download the file.)

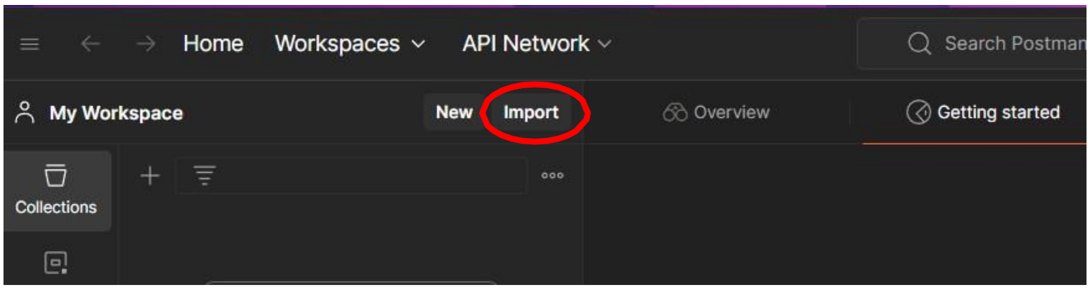
8. Now highlight this code"{{baseUrl}}"in the file and replace all with the IP & port number that we pasted in browser earlier.

(Highlight the textand pressctrl+h, you'll geta pop-up boxasgivenin the image below)

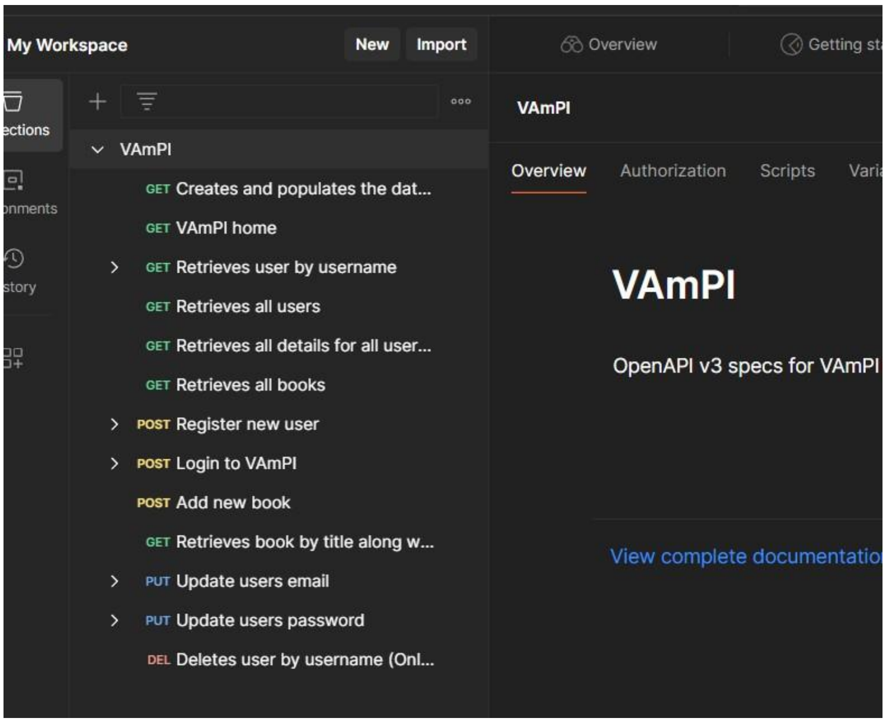


```
                },
                "method": "DELETE",
                "header": [
                        {
                                "key": "Accept",
                                "value": "application/json"
                        }
                ],
                "url": {
                        "raw": "{{baseUrl}}/users/v1/:username",
```

9. Save the file in " .json " format
10. Now go to the postman tool and import ".json" file that was saved earlier.



11. You'll get the file displayed with GETs, PUTs & POSTs, at the left of your console.



My Workspace New Import

VAmPI

- GET Creates and populates the dat...
- GET VAmPI home
- GET Retrieves user by username
- GET Retrieves all users
- GET Retrieves all details for all user...
- GET Retrieves all books
- POST Register new user
- POST Login to VAmPI
- POST Add new book
- GET Retrieves book by title along w...
- PUT Update users email
- PUT Update users password
- DEL Deletes user by username (Onl...

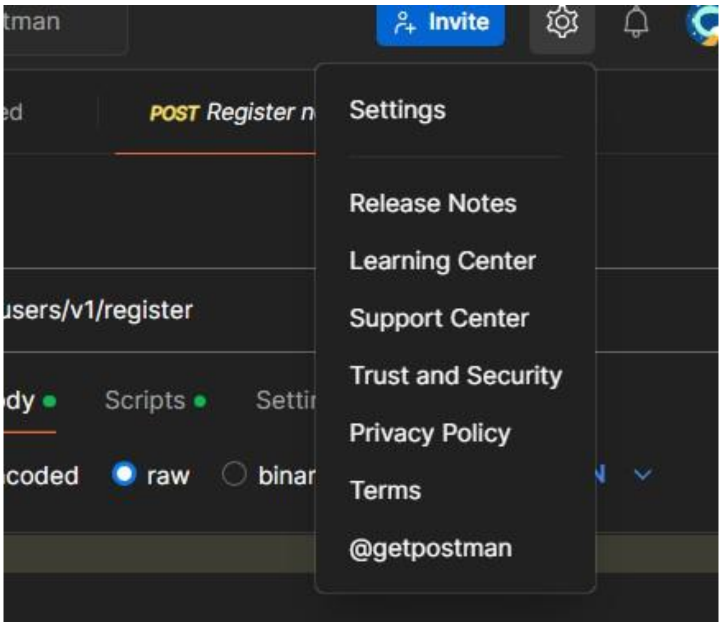
Overview Authorization Scripts

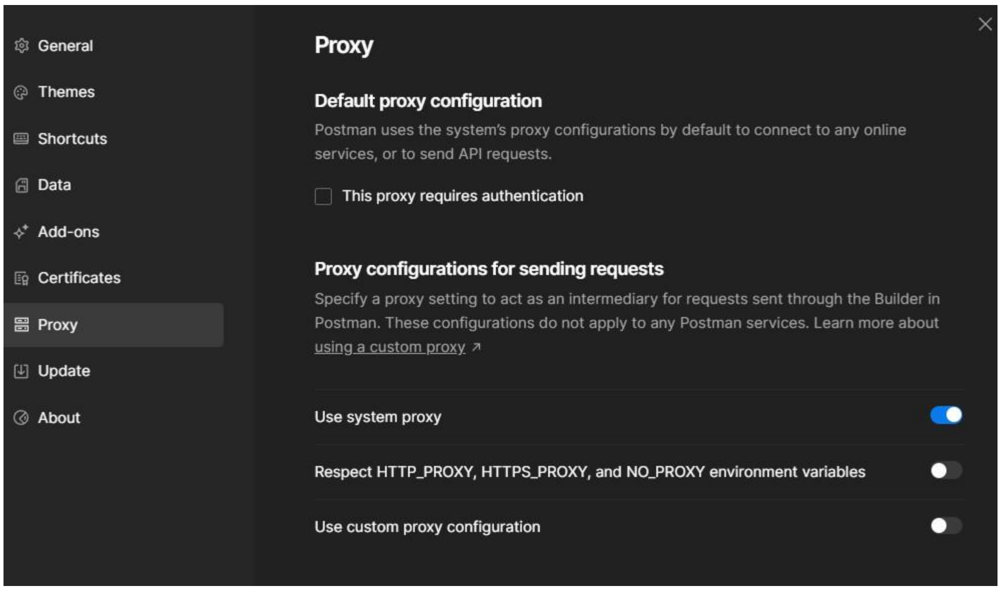
VAmPI

OpenAPI v3 specs for VAmPI

View complete documentatio

12. To capture the requests in burp suite, go to settings at the top and click On "Proxy".





General
Themes
Shortcuts
Data
Add-ons
Certificates
Proxy
Update
About

Proxy

Default proxy configuration

Postman uses the system's proxy configurations by default to connect to any online services, or to send API requests.

☐ This proxy requires authentication

Proxy configurations for sending requests

Specify a proxy setting to act as an intermediary for requests sent through the Builder in Postman. These configurations do not apply to any Postman services. Learn more about using a custom proxy ↗

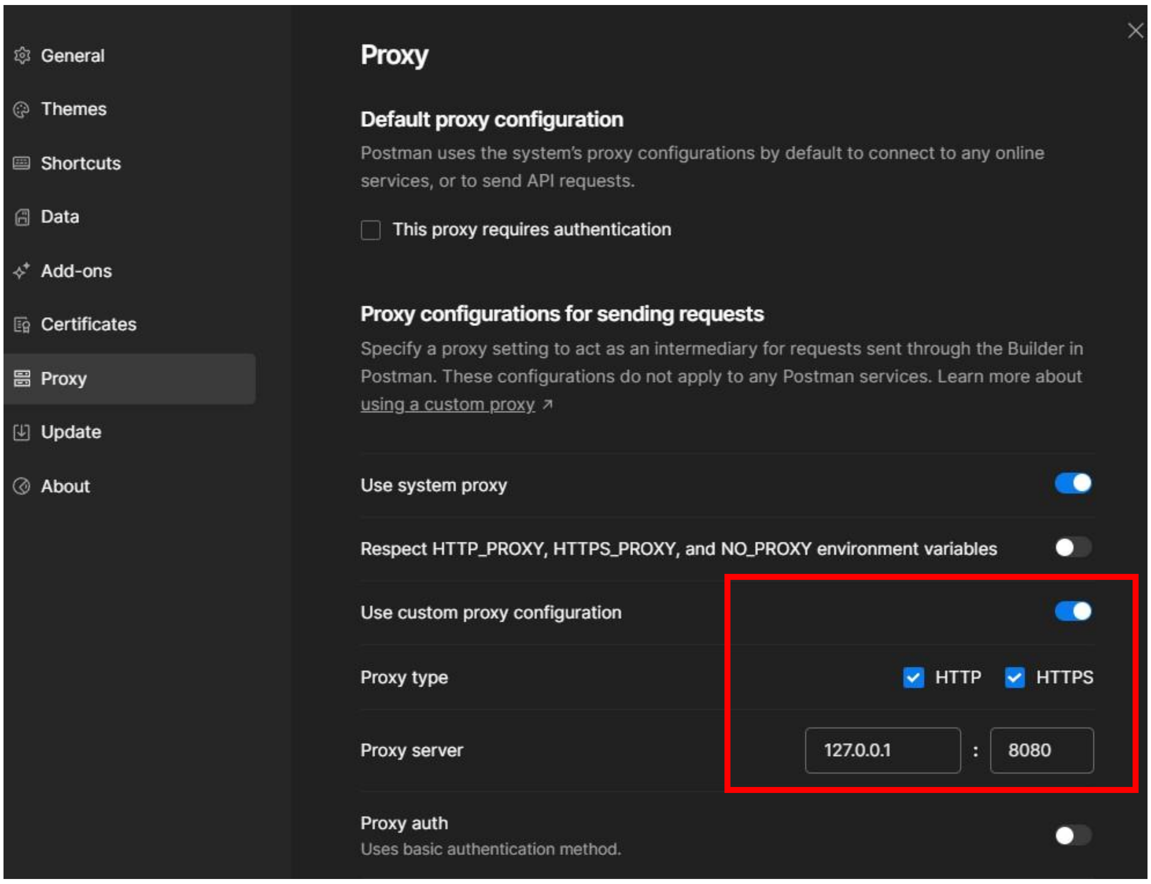
Use system proxy: on

Respect HTTP_PROXY, HTTPS_PROXY, and NO_PROXY environment variables: off

Use custom proxy configuration: off

13. Make sure your custom proxy configuration and both proxy types are enabled.Alsomakesureyourproxy serveris in127.0.0.1andtheport in 8080.

(If you don't enable it burp suite won't be able to capture the request, and the response will be shown directly in Postman tool)



General
Themes
Shortcuts
Data
Add-ons
Certificates
Proxy
Update
About

Proxy

Default proxy configuration

Postman uses the system's proxy configurations by default to connect to any online services, or to send API requests.

☐ This proxy requires authentication

Proxy configurations for sending requests

Specify a proxy setting to act as an intermediary for requests sent through the Builder in Postman. These configurations do not apply to any Postman services. Learn more about using a custom proxy ↗

Use system proxy: on

Respect HTTP_PROXY, HTTPS_PROXY, and NO_PROXY environment variables: off

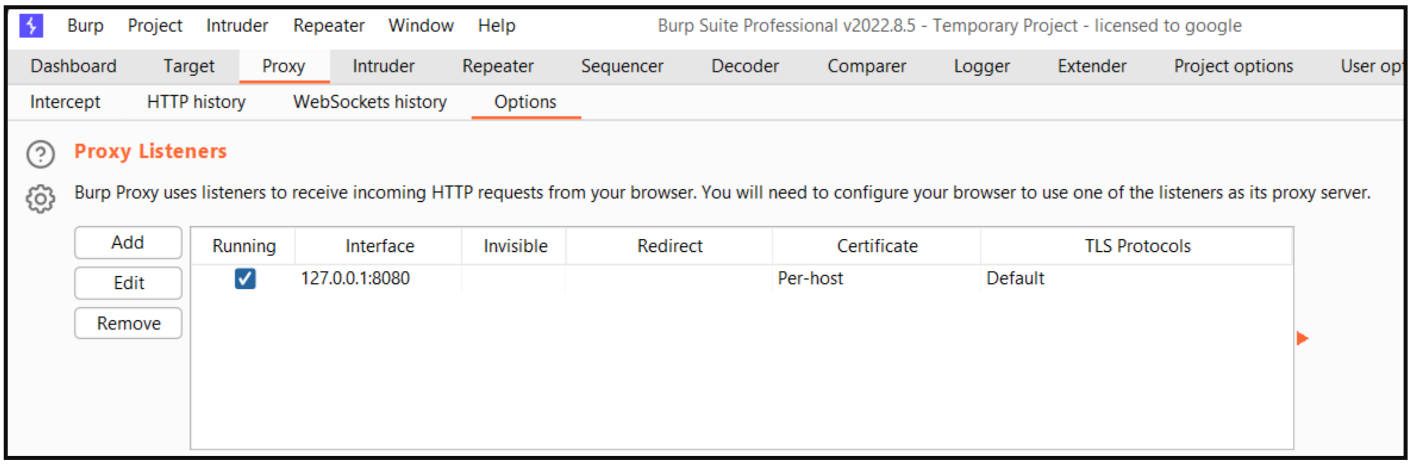
Use custom proxy configuration: on

Proxy type: ✓ HTTP ✓ HTTPS

Proxy server: 127.0.0.1 : 8080

Proxy auth: off
Uses basic authentication method.

14. Now open the burp suite tool, and go to proxy > Options, just to verify the IP and Port in Postman tool and burpsuite tool match.


Burp Project Intruder Repeater Window Help
Burp Suite Professional v2022.8.5 - Temporary Project - licensed to google
Dashboard Target Proxy Intruder Repeater Sequencer Decoder Comparer Logger Extender Project options User op
Intercept HTTP history WebSockets history Options
Add
Edit
Remove

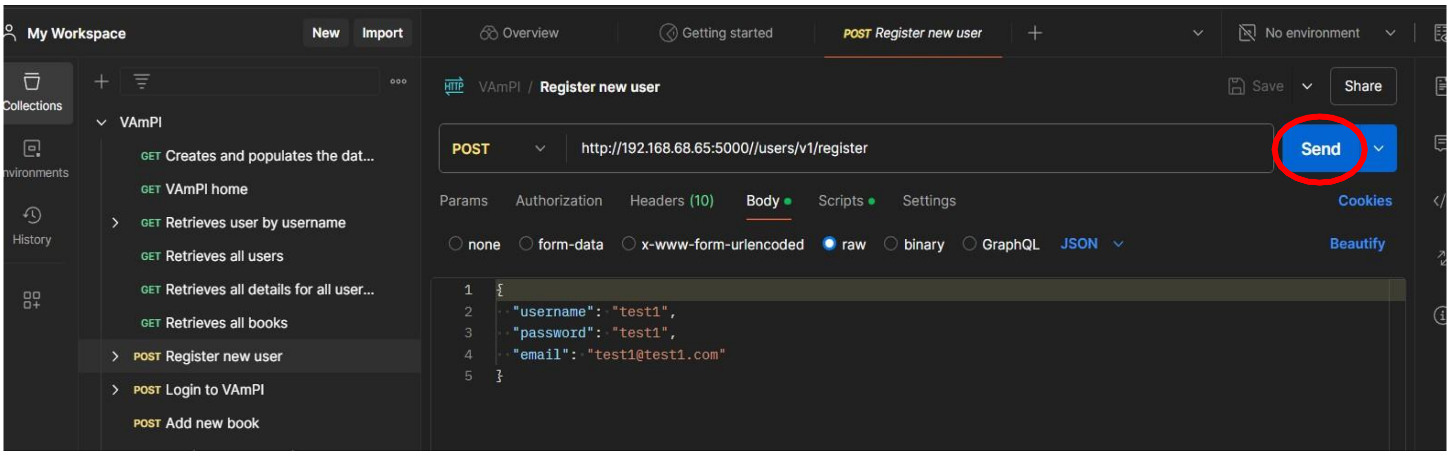
Proxy Listeners

Burp Proxy uses listeners to receive incoming HTTP requests from your browser. You will need to configure your browser to use one of the listeners as its proxy server.

Running	Interface	Invisible	Redirect	Certificate	TLS Protocols
✓	127.0.0.1:8080			Per-host	Default

So, whenever you want to capture the packets just enable the proxy configuration in Postman and turn **ON** the intercept in the burp suite.

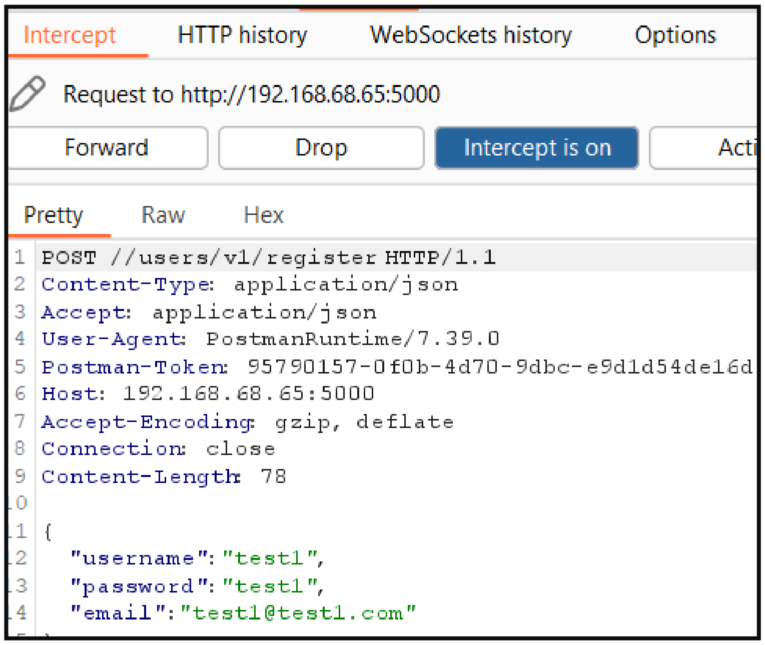
15. To test whether the burpsuite captures the packets or not, go back to postman tool and send any one packet .(make sure your proxy connection is enabled properly and the intercept in burpsuite is **ON**



POST http://192.168.68.65:5000//users/v1/register

```
{
  "username": "test1",
  "password": "test1",
  "email": "test1@test1.com"
}
```

16. Now go to Burp suite you'll find the packet captured there.



Request to http://192.168.68.65:5000

```
POST //users/v1/register HTTP/1.1
Content-Type: application/json
Accept: application/json
User-Agent: PostmanRuntime/7.39.0
Postman-Token: 95790157-0f0b-4d70-9dbc-e9d1d54de16d
Host: 192.168.68.65:5000
Accept-Encoding: gzip, deflate
Connection: close
Content-Length: 78

{
  "username":"test1",
  "password":"test1",
  "email":"test1@test1.com"
```

Further you can edit It through intruder, repeater etc. Your lab is all set.