**Simple HTTP Server with Basic Security Features**

**Project Description**

This project demonstrates a simple HTTP server written in Python that serves as a foundational step towards building a Web Application Firewall (WAF). It includes basic functionalities such as handling GET and POST requests, logging request details, and managing error responses. While it provides essential web server capabilities, it can be extended with additional security features to detect and prevent common web attacks.

**Features**

* **GET Request Handling**: The server responds to search queries by parsing the URL and returning relevant search results.
* **POST Request Handling**: The server processes form submissions by reading and parsing the request body and returning a confirmation response.
* **404 Error Handling**: The server returns a 404 error for any paths that are not explicitly handled and logs these occurrences.
* **Logging**: Comprehensive logging of all incoming requests, including request method, path, headers, and body content for POST requests, providing valuable monitoring and debugging information.

**Usage**

**Using Postman for Testing**

To test the server functionalities, Postman can be used to simulate GET and POST requests. Below are the steps to use Postman for testing:

1. **GET Request**:
   * Open Postman.
   * Create a new GET request.
   * Enter the URL: http://localhost:8080/search?q=test.
   * Click on Send.
   * You should see a response: Search results for: test.
   * The request and response will also be logged by the server.
2. **POST Request**:
   * Open Postman.
   * Create a new POST request.
   * Enter the URL: http://localhost:8080/submit.
   * Go to the Body tab.
   * Select x-www-form-urlencoded.
   * Add a key-value pair, for example: name: John.
   * Click on Send.
   * You should see a response: Form submitted!.
   * The request and response will also be logged by the server.
   * If you enter a wrong path, for example: http://localhost:8080/invalid, you will see a 404 error response.

**Steps to Run the Server**

1. Save the provided Python script as server.py.
2. Open a terminal and navigate to the directory containing server.py.
3. Run the server with the command: python server.py.
4. The server will start on port 8080 and will log incoming requests and errors.

**Future Enhancements for WAF Capabilities**

1. **SQL Injection Detection and Prevention**:
   * Detect and log suspicious SQL patterns in incoming requests.
   * Use parameterized queries to safeguard against SQL injections.
2. **Cross-Site Scripting (XSS) Protection**:
   * Sanitize and validate user inputs to remove or encode potentially harmful scripts.
   * Implement Content Security Policy (CSP) headers to mitigate XSS risks.
3. **Cross-Site Request Forgery (CSRF) Prevention**:
   * Use CSRF tokens in forms to ensure requests originate from authenticated users.
4. **Advanced Logging and Alerting**:
   * Enhance logging mechanisms to detect and log suspicious activities.
   * Implement an alerting system to notify administrators of potential security threats.

**Libraries and Tools Used**

1. **Python**: The core programming language used to implement the HTTP server.
2. **http.server**: A standard library module in Python used for creating basic HTTP servers.
3. **urllib.parse**: A module from Python's standard library for parsing URLs and query strings.
4. **logging**: Python’s built-in logging library used to record request details, errors, and other significant events.
5. **Postman**: A tool used for testing API requests and responses, used to simulate GET and POST requests to the server.