**API for an ML-based auto security and vulnerability detector framework**

Creating an API for an ML-based auto security and vulnerability detector framework involves defining the endpoints, request and response formats, and the underlying functionality to process security-related data. Here's a general outline of the API parameters you might consider:

**\*\*1. Endpoint for Scanning:\*\***

- Endpoint: `/scan`

- HTTP Method: POST

- Description: This endpoint initiates the security scanning process.

**\*\*2. Request Parameters:\*\***

- `target`: The target URL or file path to be scanned.

- `scan\_type`: Type of scan (e.g., web application, network, codebase).

- `scan\_options`: Additional options for configuring the scan (e.g., deep scan, quick scan).

**Example JSON request:**

**```json**

**{**

**"target": "https://example.com",**

**"scan\_type": "web\_application",**

**"scan\_options": {**

**"deep\_scan": true**

**}**

**}**

**```**

**\*\*3. Response Parameters:\*\***

- `scan\_id`: Unique identifier for the scan session.

- `status`: Current status of the scan (queued, scanning, completed, failed).

- `results`: Detected vulnerabilities and security issues.

**Example JSON response:**

**```json**

**{**

**"scan\_id": "12345",**

**"status": "completed",**

**"results": [**

**{**

**"severity": "high",**

**"vulnerability\_type": "SQL Injection",**

**"description": "A potential SQL injection vulnerability was detected...",**

**"location": "https://example.com/login",**

**"recommendations": "Apply input validation and parameterized queries..."**

**},**

**{**

**"severity": "medium",**

**"vulnerability\_type": "Cross-Site Scripting (XSS)",**

**"description": "A possible XSS vulnerability...",**

**"location": "https://example.com/contact",**

**"recommendations": "Sanitize user inputs and use content security policies..."**

**}**

**]**

**}**

**```**

**\*\*4. Error Handling:\*\***

Handle potential errors, such as invalid input, connection issues, or internal server errors. Provide appropriate error codes and error messages in the response.

**\*\*5. Authentication and Security:\*\***

Implement proper authentication mechanisms (e.g., API keys, OAuth tokens) to secure the API. Additionally, consider rate limiting to prevent abuse.

**\*\*6. Documentation:\*\***

Create detailed documentation explaining how to use the API, including example requests and responses, and any required headers or authentication methods.

This is a high-level overview, and the actual parameters and structure of your API may vary based on your specific implementation and the technologies you're using. Also, security-related tools and frameworks are complex, so ensure that your API design aligns with best practices to provide effective and reliable security scanning capabilities.

**Input Parameters:**

**Code/Input Text**: The code or input text must be analyzed for security vulnerabilities.

**Language/Framework:** The programming language or framework is used. This can help the system identify language-specific vulnerabilities.

**Configuration Parameters:**

**Scan Type:** Specify whether you want a quick scan, deep scan, or a specific type of scan (SQL injection, cross-site scripting, etc.).

**Sensitivity Level**: Adjust the sensitivity of the detection, which can affect the number of false positives/negatives.

**Whitelist/Blacklist**: Specify certain patterns or keywords to be ignored or focused on during the scan.

Exclude Paths: Paths or files that should be excluded from the scan.

**Output Parameters:**

**Detected Vulnerabilities:** Information about the vulnerabilities found, including type, severity, location, and recommendations for remediation.

**Risk Score**: An overall risk score for the analyzed code.

**Report Format**: Choose the format for the generated report (JSON, XML, HTML, etc.).

**Actions**: Optionally, actions to be taken automatically (e.g., quarantine code, send notifications) upon detecting vulnerabilities.

**Authentication/Authorization:**

**API Key/Token:** An authentication token to access the API.

**User Permissions**: Depending on the user's role, certain functionalities might be restricted.

**Integration/Output Options:**

**Integration with CI/CD:** Parameters to integrate the API with continuous integration and continuous deployment pipelines.

**Webhooks/Notifications:** Parameters to configure notifications or callbacks when vulnerabilities are detected.

**Export Options:** Parameters to export the results to other tools or systems.

**Versioning/Backward Compatibility:**

**API Version:** Specify the API version being used to ensure compatibility.

**Deprecation Notices:** Information about deprecated parameters or features.