ENSURING API SECURITY IN WEB SERVICES DEVELOPED IN PYTHON

Qurbonov Behruz Amrulloyevich
Tashkent University of Information Technologies
named after Muhammad al-Khwarizmi 3rd year student
Faculty of Software Engineering
Recipient of the Muhammad al-Khwarizmi scholarship
Abdumalikov Nurmuxammad Sherzod o‘g‘li
Tashkent University of Information Technologies
named after Muhammad al-Khwarizmi2nd year student
Faculty of Software Engineering

Abstract: This article discusses the key points of developing a secure RESTful web service API for keeping a student achievement journal. The relevance of using web services has been analyzed. The classification of web applications is given. The features of the Single Page Application architecture were considered. Comparative characteristics of architectural styles of application programming interfaces are given. Requirements to be met by RESTful API services are considered. The basic principles of API security were analyzed. A list of the main vulnerabilities that may appear during the development of the REST API is given. An overview of popular authentication schemes (methods) is given. Comparative characteristics of web frameworks of the Python programming language are given. The main tools used in the development of web API applications are listed. The process of creating a secure prototype of a RESTful web service API in Python using the Flask microframework and a tool for describing the Swagger specifications is presented. The process of configuring the application was examined in detail. The main recommendations for securing a web application, database and web server settings are listed. The key points of ensuring the protection of the developed web application are considered. The results obtained were analyzed.

Keywords: RESTFull API , Artificial Intelligence (AI) , Machine Learning (ML) , Deep Learning , Natural Language Processing (NLP) , Real-Time Analysis.

A web application is no longer a pre-assembled collection of HTML pages, style files, scripts, and media files. Web applications are based on a client-server architecture based on the separation of a functional block and appearance). The business logic of the application is performed on a special dedicated server (back-end), while the client interacts through the user interface (UI) in his Internet browser (front-end). This approach allows to generate HTML markup content in real time, with a minimum number of reloads on the

viewed web page. Almost all modern e-commerce sites, social networks, instant messengers, online programs, forums, and search engines are web applications.

By the type of main components, web applications can be classified into several types: backend, frontend and SPA. Backend applications (server-side applications) collect the main functionality and logic of the application, interact with the database. They are developed using a number of programming languages, the most popular of which are: Python, Ruby, PHP, C # (.NET), Node.js. However, it is almost impossible to implement production-ready solutions without the use of special tools called web frameworks. Each of the programming languages listed above has its own set of frameworks for creating web applications: Python (Django, Flask, FastAPI, Tornado), PHP (Laravel, Simfony, Yii2), Ruby (Ruby on Rails, Sinatra), Node.js (Express .js), C # (ASP .NET Core).

Frameworks take on some of the developer's responsibilities and help in ensuring application security, working with typical redirection, authentication, registration and other functionality. The application is deployed on a specially configured web server and independently generates the HTML representing pages (however, the browser will reload the page in most cases).

Frontend applications (client-side applications) work directly in the user's web browser. They are developed using JavaScript, TypeScript and Vue.js, ReactJS, AngularJS programming languages. They are applicable in cases when the application does not work with the database and does not store client information for longer than one session. They are fully responsible for the presentation of the HTML code. SPA (Single Page Application) is a combination of solutions, dividing a web application into two (frontdend and backend) and organizing interaction between them. The backend part deals with logic, processes request, works with the database; the frontend part forms an external view based on the data received from the backend and displays the result in the browser. The exchange of information between the client and the server parts of the application, as a rule, occurs through a specially developed application programming interface (API). Almost all modern web applications are based on the SPA concept. API is a set of methods and rules by which applications exchange messages and transfer data. As a rule, these are classes, functions, structures of one program, interacting with other programs. API not only helps organize interactions; it also plays an important role in securing that interaction. Such an interface hides the specifics of the service implementation and works within the framework of what the developers provide. Using the program interface of the application, the operations that are available to the user are defined, restrictions are imposed, input and output data are indicated. In addition, API has a number of architectural styles that reflect certain aspects of the interface work. Applications of this kind have complex logic and consist of many components, which noticeably affect the security of the system. There are many factors to consider when organizing security. In addition to the front and back-ends

security, one should focus on developing a secure API. Intrusion or an accidental client error can result in service unavailability, data loss, or complete application operability failure. This can critically affect the user experience, and as a result, bring massive losses to business. Therefore, ensuring security is a topical issue when developing any web service. This article is devoted to the development of a secure RESTful API web service for generating a list of student achievements using the Python3 programming language and the Flask web microframework.

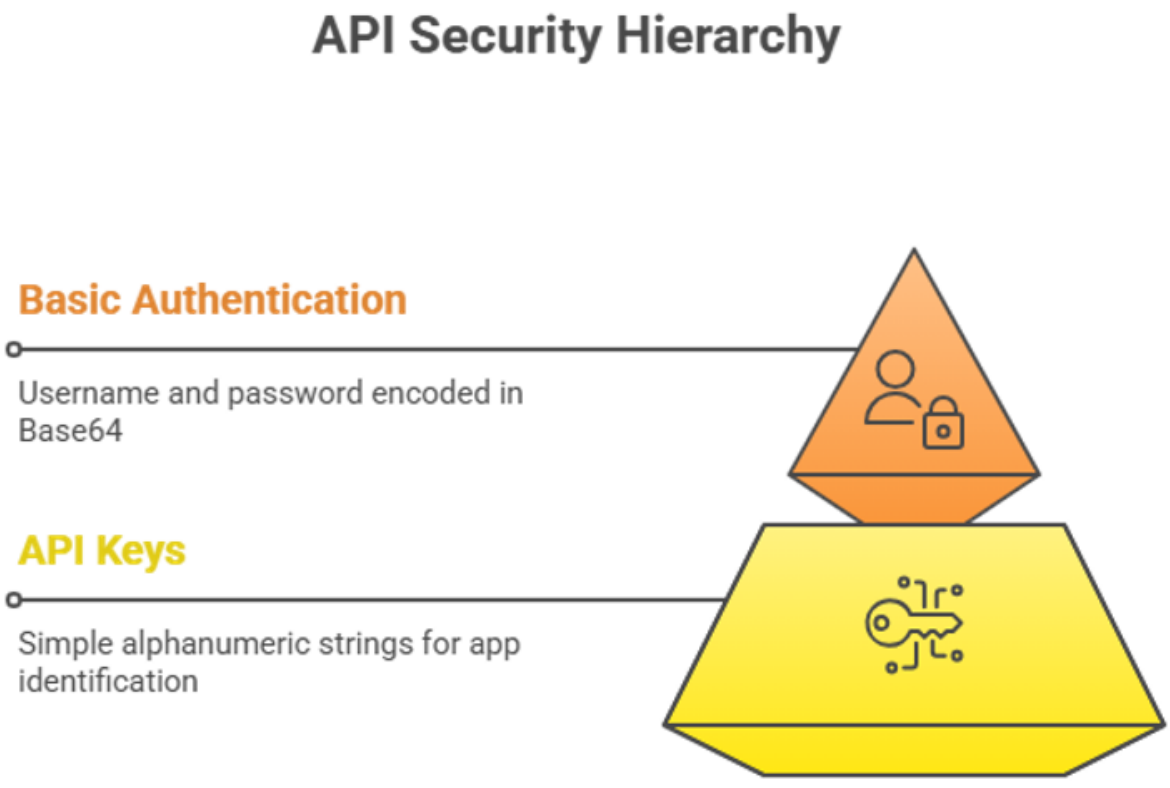
Core Problem: Traditional network security systems are reactive and signature-based. They can only defend against known threats, failing to detect novel attacks or zero-day exploits. As a result, organizations remain vulnerable to data breaches, ransomware, and advanced persistent threats (APTs).

Proposed Solution: AI-Driven Network Security

Artificial Intelligence (AI), with its ability to learn from vast data and identify hidden patterns, provides a proactive and intelligent defense mechanism. AI enhances network security by:

- Detecting anomalous behavior
- Predicting potential cyberattacks
- Automating threat responses
- Identifying previously unseen malware variants

The use of AI shifts the paradigm from rule-based static defense to adaptive and predictive security models.



Key Technologies Used

- **Machine Learning (ML):** Supervised and unsupervised models identify normal vs. abnormal network traffic.
- **Deep Learning:** CNNs and RNNs detect patterns in packet data, user behavior, and logs.
- **Natural Language Processing (NLP):** Analyzes phishing emails and threat intelligence feeds.
- **Reinforcement Learning:** Optimizes firewalls and intrusion prevention systems (IPS).

AI-Based Threat Detection Architecture

1. **Data Collection:** Logs from firewalls, routers, endpoints, and user activity.
2. **Preprocessing:** Feature extraction (e.g., IP addresses, ports, protocols, time windows).
3. **Model Training:** ML models trained on labeled datasets (attack vs. normal).
4. **Real-Time Analysis:** Incoming traffic is classified in real time.
5. **Alerting & Response:** When an anomaly is detected, alerts are generated, or automatic mitigation occurs.

Mathematical Formulation: Anomaly Detection

Let $X = \{x_1, x_2, ..., x_n\}$ represent network activity features (e.g., traffic size, protocol type, source IP).

The anomaly score is calculated as: $A(x) = \frac{|x-\mu|}{\sigma}$ Where:

- μ: mean of historical data
- σ: standard deviation

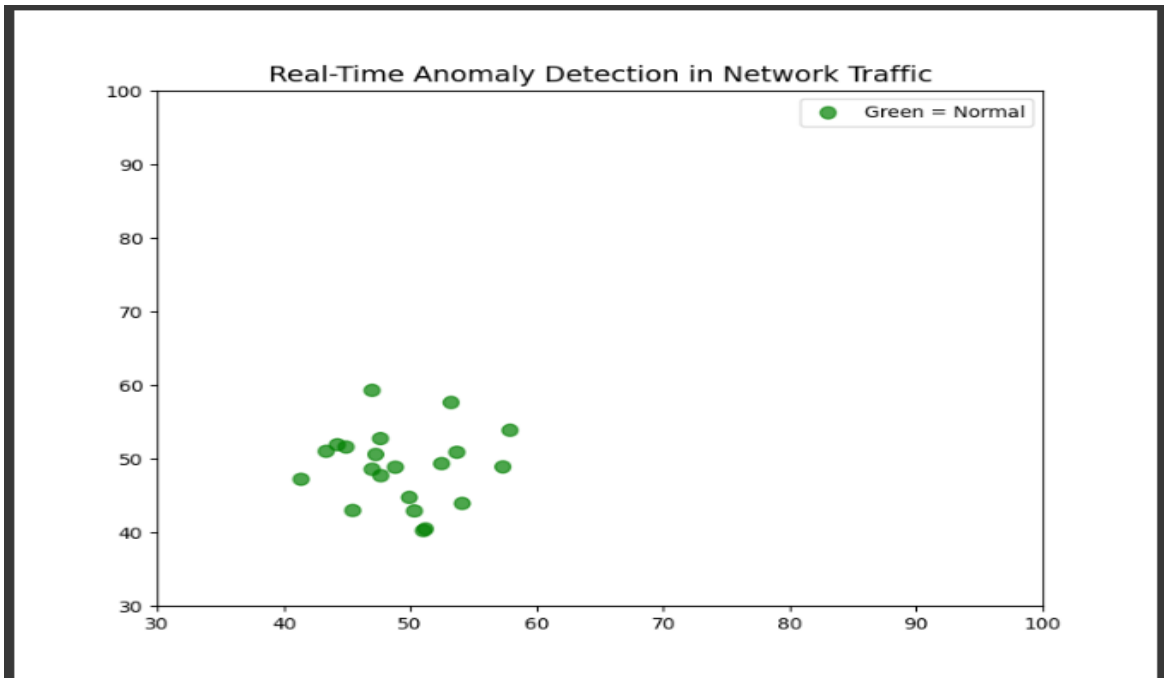
If $A(x) > \text{threshold}$, the behavior is flagged as anomalous.

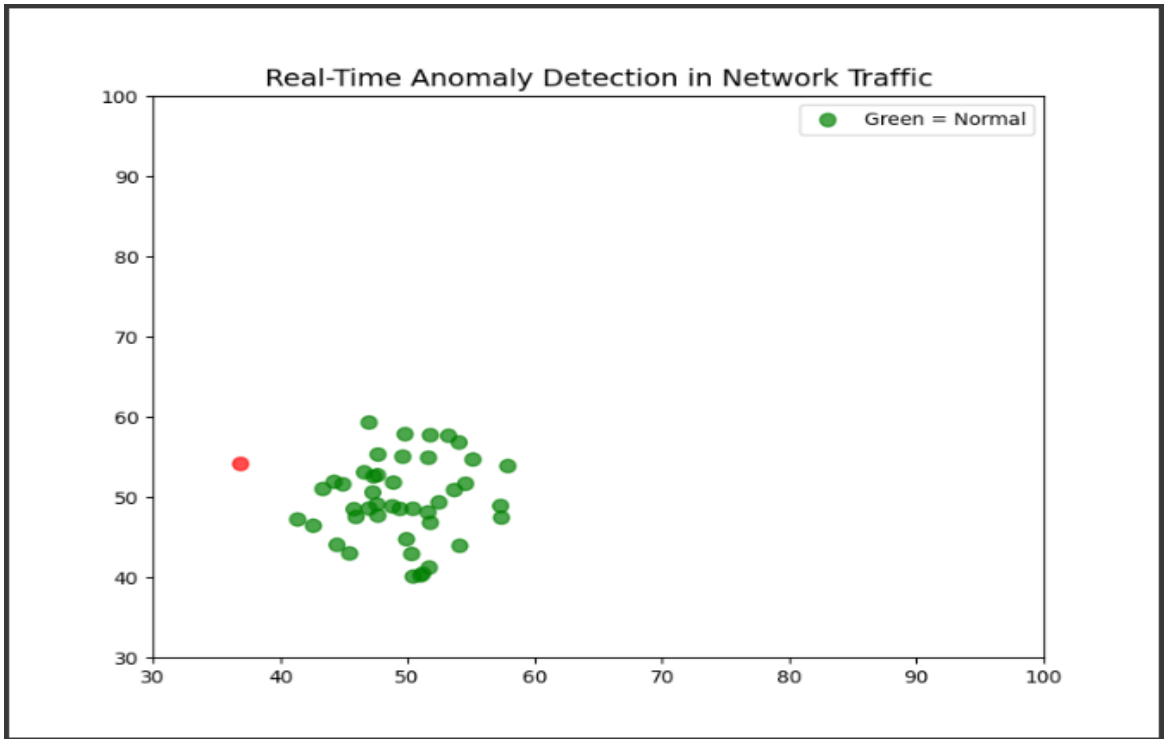
Alternatively, we can use a One-Class SVM or Isolation Forest for unsupervised anomaly detection: $f(x) = \begin{cases} 1 & \text{normal} \\ -1 & \text{anomaly} \end{cases}$

The server on which the developed web application will be launched is also a potentially vulnerable point for attackers. To reduce the risk of hacking, you need to disable root logins of the linux server by editing the file / etc / ssh / sshd_config. In the PermitRootLogin option, change the value from yes to no. The next step is installation and configuration of the firewall. It is necessary to close access to all ports except 80 (HTTP), 443 (HTTPS) and 22 if we work with server via SSH: sudo ufw allow ssh hhtp 443 / tcp – force enable.

In addition, you should replace the standard Flask application server with a more advanced WSGI server, gunicorn. It is a robust production server that many developers take as a standard when developing Python web applications. Configure the service to wait for queries from private port 8000: gunicorn -b localhost: 8000 -w 4 restapiservice: app. However, application web traffic can only go through ports 80 and 443 open on the firewall. At the same time, to ensure the security of the application, all traffic arriving on HTTP port 80 must be redirected to encrypted port 443 (HTTPS). To work with the HTTPS protocol, you need to obtain an SSL security certificate (for example, Let's Encrypt): sudo Apt install python-certbot-nginx / sudo certbot –nginx -d <application_domain> -d www. <application_domain> An Nginx proxy will be used to redirect traffic. Listening port settings, HTTPS connection, and traffic redirection are done by setting the configuration file / etc / nginx / sites-enabled / default.The default SQLite database is also not an acceptable solution. The application will use the reliable and efficient PostgreSQL

DBMS, which is the de facto standard on the Python + Django / Flask technological stack. However, no DBMS can efficiently service the large number of independent processes that call for it. Too many queries (from real users or those created by an intruder) will lead to the failure of the database. To solve this problem, we will use the PgBouncer connection puller. The puller proxies all incoming queries in small chunks, creating specialized queues.





REFERENCES

1. Kai Hwang, Zhiwei Xu – *Distributed and Cloud Computing: From Parallel Processing to Big Data*
2. Murat Kantarcioglu – *Artificial Intelligence for Cybersecurity: A Comprehensive Guide*
3. Charles Brooks – *Cybersecurity Issues and AI Solutions in Modern IT Environments*
4. Roman V. Yampolskiy – *Artificial Intelligence Safety and Cybersecurity*
5. Ali Dehghantanha, Reza M. Parizi – *Machine Learning Applications in Cybersecurity*
6. Sumeet Gupta, Manoj Singh Gaur, Vijay Laxmi – *AI-Based Network Intrusion Detection Systems: A Survey*
7. MIT CSAIL - Artificial Intelligence & Cybersecurity Research – https://www.csail.mit.edu/
8. IEEE Xplore Digital Library – AI and Cybersecurity – https://ieeexplore.ieee.org/
9. Springer Journal of Cybersecurity and AI Integration – https://www.springer.com/journal/144
10. Ponemon Institute Reports on AI in Cybersecurity – https://www.ponemon.org/