Science, Engineering & Education, 10, (1), 2025, 83-92

Monitoring and Scanning System for APIs

Lyuboslav Spiridonov, Svetozara Stoyanova*

PGEHT "Prof. Asen Zlatarov", Pleven, 2 Tsar Samuil St.

Received 30 May 2025, Accepted 27 July 2025
DOI: 10.59957/see.v10.i1.2025.11

ABSTRACT

This article presents a comprehensive system for monitoring and scanning APIs, enabling developers, administrators, and IT professionals to oversee performance, availability, and security in real time. The platform integrates ASP.NET microservices, C++ modules for security-critical operations, and a Node.js Backend-for-Frontend built with Express.js and TypeScript, layered with an Angular interface to combine scalability with usability. Security is reinforced through hardware and operating system adaptive Argon2id password hashing, offline verification against a Have I Been Pwned binary dataset, and claims-based access control. The scanning subsystem evaluates TLS certificate validity, framework fingerprinting, header compliance, and open ports, while the monitoring layer collects response times, error rates, and system load. Deployment in Kubernetes with Istio ingress and ArgoCD pipelines provides resilience and automation, while Argo Workflows orchestrate unit and integration testing. By unifying monitoring, scanning, and adaptive protection, the system delivers a reliable and technically robust approach for maintaining secure API infrastructures in modern digital environments.

Keywords: API monitoring, performance analysis, security, Argon2id, cloud-native, backend-for-frontend.

INTRODUCTION

APIs are the backbone of modern software, enabling services to communicate and integrate across diverse platforms. However, as applications scale, API endpoints become vulnerable to performance degradation, misuse, and security threats. Existing monitoring solutions often provide limited visibility, delayed anomaly detection, or lack integrated security mechanisms, leaving critical systems exposed.

This project addresses these challenges by developing a comprehensive API monitoring and scanning system capable of continuous performance tracking, anomaly detection, and security enforcement. By collecting metrics such as response times, error rates, request throughput, and system load, the system enables timely detection of operational issues. Security is reinforced through adaptive password hashing, offline credential verification, and claims-based access control. The platform combines microservices architecture with a Backend-for-Frontend to provide a scalable, user-centric interface.

**Correspondence to: Svetozara Stoyanova, PGEHT "Prof. Asen Zlatarov", e-mail: zaritooo@abv.bg*

By bridging performance monitoring with advanced security features, this system offers a practical approach to maintaining resilient and reliable API infrastructures in modern software environments.

OBJECTIVE AND OVERVIEW OF THE PROBLEM DOMAIN

This project aims to develop and implement a system for continuous monitoring and scanning of APIs, providing in-depth analysis of their operational state while minimizing disruption to services. The solution is designed to enhance stability, security, and performance by detecting anomalies such as latency spikes, abnormal error rates, TLS certificate expiry, and unusual traffic patterns.

In contemporary software environments, monitoring API endpoints presents several challenges. Existing solutions often provide limited visibility, fail to detect issues in real time, or offer inadequate integration with security and performance analysis. Detecting performance bottlenecks, handling dynamic workloads, and maintaining fine-grained access control requires a flexible, scalable, and integrated approach.

The system addresses these challenges by combining a microservices-based server-side architecture with a Backend-for-Frontend for the end user. Real-time event streams enable continuous metrics collection, while specialized C++ modules handle security-critical operations. Continuous integration pipelines, orchestrated through Argo Workflows, automatically validate and test changes, while continuous deployment with ArgoCD ensures that updates reach production with minimal manual intervention. These processes run on Kubernetes, where containerized services can be scaled dynamically. Together, this architecture delivers scalability, resilience, and efficient resource utilization, providing a dependable solution capable of meeting the dynamic demands of modern API-driven applications.

CHOICE OF PROGRAMMING LANGUAGES, FRAMEWORKS, AND DEVELOPMENT ENVIRONMENT

The system integrates multiple programming languages, frameworks, and specialized tooling to achieve security, performance, and scalability.

Backend and core services

- C# with ASP.NET Core forms the backbone of the system's API layer, implementing business logic orchestration, claims-based access control, and secure endpoint exposure [1]. NuGet packages extend functionality:
- DnsClient.NET - advanced DNS resolution in endpoint scanning.
- Humanizer - readable audit logs.
- SemaphoreSlim - bounded parallelism in request processing, preventing thread starvation.
- System.Reactive (Rx.NET) - event-driven scanning of endpoints and API checks.
- System.Threading.Channels - efficient producer-consumer pipelines for high-volume metric streams without back-pressure bottlenecks.
- Task Parallel Library (TPL) - high-throughput asynchronous workloads.
- C++ Modules handle security-critical operations, including hardware-adaptive Argon2id password hashing and offline verification against a Have I Been Pwned binary dataset [2].

These modules communicate with ASP.NET via PInvoke, isolating low-level cryptographic logic while preserving performance. Unit testing is performed with **Catch2**, integrated via CTest into continuous integration pipelines.

Backend-for-Frontend (BFF) and Frontend

- Node.js BFF (Express.js + TypeScript) mediates requests between the Angular client and backend services [3]. Its toolchain reinforces reliability and performance:
- Zod - runtime schema validation for payload

integrity.
- p-memoize - caches repeated calls to reduce redundant processing.
- http-proxy-middleware - controlled, secure routing to backend APIs.
- Pino - structured, high-throughput JSON logging compatible with centralized observability pipelines.
- Jest - unit and integration testing for BFF logic.
- Angular with TypeScript and SCSS delivers a responsive, component-driven frontend [3]. Development leverages Angular CLI for scaffolding, AOT compilation, and lazy-loading optimizations. RxJS is used extensively for reactive data streams, integrating seamlessly with backend metrics and alerts. Unit and integration testing is performed with **Jest** to ensure frontend stability.

Data and messaging layer
- MariaDB stores user data for fine-grained access control [4].
- Redis serves as a distributed cache and transient state store to reduce backend load.
- Kafka and NATS provide reliable, scalable inter-service communication, balancing durability (Kafka) and low-latency messaging (NATS).

Logging and observability
- ASP.NET APIs send logs and telemetry to a dedicated logging API, centralizing event storage and standardizing log formats [2].
- BFF and frontend components emit structured logs with Pino and Humanizer, allowing audit trails to remain human-readable.
- Observability integrates metrics, logs, and traces into Prometheus and Grafana, providing full-stack visibility into performance, errors, and system load.

Orchestration, deployment, and security
- The system is containerized and deployed in Kubernetes, with separate namespaces for development, staging, and production [5].
- HashiCorp Vault manages secrets, configuration, and environment-specific credentials, eliminating hardcoded sensitive information [2].
- Continuous integration is automated with Argo Workflows, executing xUnit tests for ASP.NET, Catch2 for C++, and Jest for Node.js BFF [5].
- Continuous deployment is driven by ArgoCD**,** ensuring declarative, reproducible, and minimal-intervention rollouts [5].
- Istio serves as ingress, managing routing, TLS termination, and traffic observability [5].
- NGINX functions as a reverse proxy and static file server for Angular assets [5].

Development environment and tooling

Development consistency and productivity are maintained through JetBrains IDEs and specialized tools [1]:
- Rider for ASP.NET Core APIs, with integrated NuGet management, debugging, and xUnit support.
- CLion for C++ modules with Catch2, CMake, and low-level debugging.
- WebStorm for Node.js BFF and Angular frontend, including ESLint, Prettier, Jest, and Angular CLI.
- DataGrip for MariaDB and Redis exploration, schema management, and query optimization.
- Valgrind for memory profiling, leak detection, and runtime analysis in C++ modules.

DESCRIPTION OF THE WEB APPLICATION

The API monitoring system integrates robust functionality with usability, fostering active interaction between all users while ensuring high security and reliability [2]. It continuously monitors API endpoints critical to the web application, performing checks for availability, response time, throughput, data integrity, and

TLS compliance, while promptly alerting users to anomalies or potential issues. Metrics collection occurs via event-driven streams, allowing the system to detect bottlenecks, abnormal error rates, or sudden latency spikes in real time [1].

During user registration, strict password requirements are enforced: passwords must be at least 12 characters long, include lowercase letters, uppercase letters, digits, and at least one special character. Each password is validated against an offline Have I Been Pwned binary dataset[5] to prevent the use of compromised credentials. The system employs hardware-adaptive Argon2id hashing, tuning memory, iterations, and parallelism based on detected CPU cores and available RAM to maximize security without degrading performance. Upon successful registration, users are redirected to the login page. After authentication, each user receives a session uniquely tied to their IP address, with associated cookies signed and encrypted to maintain integrity and confidentiality [2].

A web-based command-line interface (CLI) provides advanced interaction with the system. Users can execute standard commands such as clear (clear console), help (view available commands), whoa mi (inspect account details), and logout (terminate session). Operational commands extend the system's oversight capabilities:

- metrics retrieves API statistics, including request counts, response time percentiles, error rates, and throughput, aggregated across endpoints and services.
- threats list recent security incidents, categorized by IP, type, and severity, with optional correlation to known blacklists.
- scan <url> [method] [key] conducts in-depth API analysis, reporting TLS certificate issuer and expiry, framework fingerprinting, open ports, API type (REST, GraphQL, gRPC, SOAP, etc.), and analysing HTTP headers (present or missing). For any missing headers, the system provides actionable suggestions for critical security and performance headers. For example, it checks for:
- Strict-Transport-Security (HSTS) - ensures HTTPS-only connections to prevent downgrade attacks.
- Content-Security-Policy (CSP) - mitigates XSS and data injection attacks by restricting sources.
- X-Frame-Options / Frame-Options - prevents clickjacking by controlling iframe embedding.
- X-Content-Type-Options - stops MIME type sniffing, reducing risk of content injection.
- Referrer-Policy - controls how much referrer information is shared, enhancing privacy.
- Cache-Control / Pragma - enforces proper caching policies to avoid sensitive data leakage.
- Permissions-Policy - limits use of browser features such as geolocation or camera access.

The system not only flags missing headers but provides recommended values based on OWASP best practices and industry standards. This allows developers and administrators to quickly remediate insecure or incomplete configurations, ensuring the API endpoints maintain both robust security posture and optimal compliance.

DNS resolution records (A, AAAA, CNAME, MX, TXT) are thoroughly analyzed. The scanning subsystem leverages DnsClient.NET for advanced DNS queries, event-driven reactive pipelines (Rx.NET) for orchestrating asynchronous lookups, and SemaphoreSlim with Channels to safely throttle and parallelize requests. DNS responses are cached respecting TTL values, reducing redundant queries while ensuring fresh data for critical endpoints. This combination enables high-throughput, resilient, and non-blocking operation, allowing the system to accurately resolve domains and detect DNS anomalies even under heavy load or large-scale endpoint scans [1].

Security is reinforced through ASP.NET middleware, including FirewallRequest, blacklists, whitelists, and adaptive rate limiting

[2]. Repeated lockouts dynamically reduce allowed requests per IP while maintaining legitimate access, with subnet-level blacklisting applied for abusive traffic patterns. Requests and audit logs are routed through a dedicated logging API, with Humanizer ensuring structured, human-readable output compatible with centralized observability pipelines for backend and BFF layers [2].

The system ensures secure communication and data exchange between all components. Claims-based access control enforces fine-grained permissions on both endpoints and backend services [4]. All inter-service messaging uses Kafka and NATS, balancing durability and low-latency event propagation [1]. Deployment leverages Kubernetes namespaces for development, staging, and production, while Argo Workflows execute automated testing (xUnit for ASP.NET, Catch2 for C++, Jest for Node.js BFF) and ArgoCD handles continuous deployment [5]. Istio manages ingress, TLS termination and mutual TLS (mTLS) across all pods ensuring safe communication across every aspect, and traffic observability, while NGINX acts as a reverse proxy and static asset server for Angular [1].

By combining real-time monitoring, reactive event-driven scanning, detailed performance analysis, advanced security features, and a user-friendly interface, the platform guarantees uninterrupted operation of the web application while sustaining robust protection, optimal efficiency, and full-stack observability [1, 2, 5].

Fig. 1. illustrates the login page, where users authenticate themselves using email and password. The interface includes a "Register" button directing users to the registration form, which enforces secure password policies and provides a "Remember Me" option for persistent sessions. Invalid login attempts trigger descriptive error messages, while successful authentication redirects users to the homepage[5].

Fig. 2. displays the user menu, enabling full user data management through create, read, update, and delete (CRUD) operations. The menu presents information such as username and roles, allows changing usernames, and provides the ability to block users by IP address or delete a user profile. All actions are logged for auditing, including username changes, password updates, and account deletions, with users receiving notifications for each modification [1].

Fig. 3. presents the command-line interface (CLI) homepage. The CLI allows users to execute standard commands such as clear, help, whoa mi, and logout. Operational commands extend system oversight, including metrics for retrieving API statistics, threats for reviewing recent security incidents, and scan <url> [method] [key] for in-depth API analysis. This analysis includes SSL/TLS certificate details, framework detection, open ports, API type, HTTP headers, and DNS resolution 32].

PERFORMANCE EVALUATION

The performance of the offline Have I Been Pwned (HIBP) engine was benchmarked using a diverse set of passwords, measuring minimum, median, and maximum lookup times in microseconds. As illustrated in Fig. 4, the median lookup times consistently remain below 660 microseconds across most passwords, demonstrating that the system can verify credentials with minimal latency. Maximum times occasionally spike above 1 200 microseconds for certain inputs, likely due to hash collisions or disk access patterns when reading from the binary dataset.

The results indicate that the system achieves a predictable and low-latency performance profile suitable for integration with real-time registration and authentication workflows. Even under worst-case conditions, the maximum lookup time does not exceed 1.3 milliseconds, which is negligible compared to the overall user registration process. The minimum times, shown in purple, reflect the efficiency of the memory-mapped dataset access

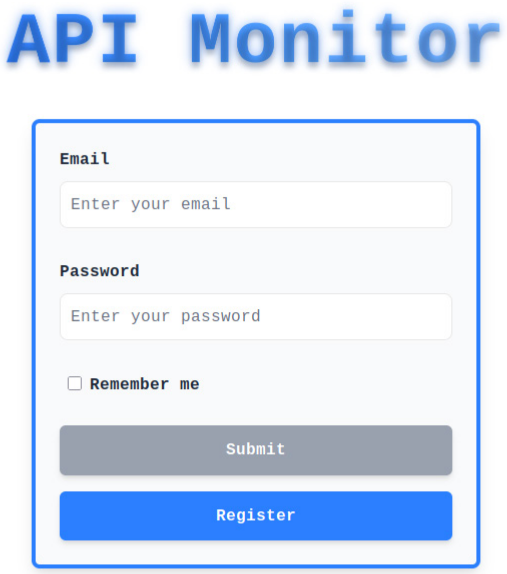
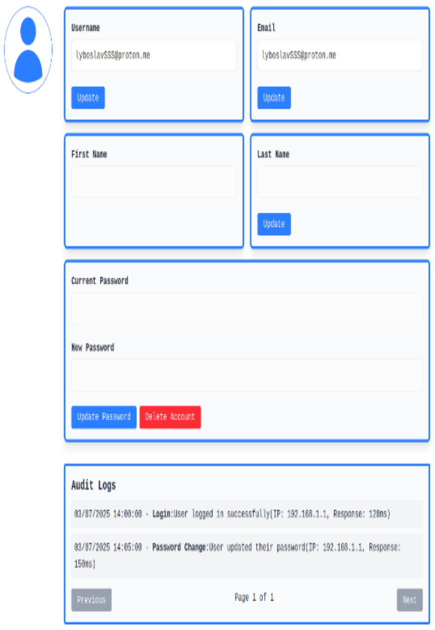


Fig. 1. Login page.


Username: lyboslavSSS@proton.me
Email: lyboslavSSS@proton.me
First Name
Last Name
Current Password
New Password

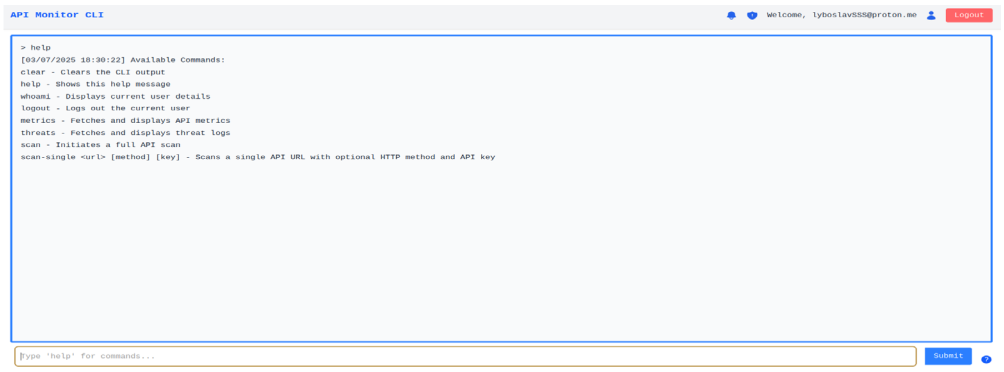
Audit Logs

03/07/2025 14:00:00 - **Login**:User logged in successfully(IP: 192.168.1.1, Response: 120ms)

03/07/2025 14:05:00 - **Password Change**:User updated their password(IP: 192.168.1.1, Response: 150ms)

Page 1 of 1

Fig. 2. User menu.



```
> help
[03/07/2025 18:30:22] Available Commands:
clear - Clears the CLI output
help - Shows this help message
whoami - Displays current user details
logout - Logs out the current user
metrics - Fetches and displays API metrics
threats - Fetches and displays threat logs
scan - Initiates a full API scan
scan-single <url> [method] [key] - Scans a single API URL with optional HTTP method and API key
```

Fig. 3. Command Line Interface.

and optimized search algorithms implemented in the C++ module.

Overall, the benchmark confirms that the HIBP engine is highly efficient, enabling offline password verification without compromising user experience. These results validate the system's ability to enforce strong security policies while maintaining high throughput, supporting the broader goal of securing API endpoints and user accounts.

In addition to credential verification performance, the scanning subsystem was evaluated across multiple high-availability domains to measure the execution times of its core analysis tasks. As illustrated in Fig. 5, the benchmark records the latency of DNS lookups, SPF record parsing, TLS certificate inspection and multi-port scanning operations for openai.com, stackoverflow.com, google.com, github.com and microsoft.com.

DNS resolution remains consistently low at approximately 10 - 30 ms due to the use of DnsClient.NET with reactive pipelines and TTL-aware caching, ensuring that repeated queries do not introduce unnecessary overhead. SPF extraction and evaluation incur slightly higher costs, averaging 40 - 80 ms, because of additional TXT record parsing and include-directive expansion; concurrency limits enforced by SemaphoreSlim maintain stability under heavy workloads. TLS certificate inspection averages 100 - 150 ms per host as the subsystem retrieves, validates and analyses certificate chains.

Port scanning dominates the total scan time at roughly 2.4 s per host. Even though the implementation leverages asynchronous sockets and producer-consumer channels to pipeline probes, network latency and handshake timeouts make this step inherently slower than the other checks. By executing port scanning in a dedicated event-driven pipeline, the system allows faster DNS, SPF and TLS checks to return results promptly without waiting for full port enumeration to complete.

This benchmark confirms that the scanning subsystem delivers predictable latency characteristics across heterogeneous endpoints. Lightweight checks complete within tens or hundreds of milliseconds, while long-running operations are isolated to prevent bottlenecks. Combined with bounded parallelism and back-pressure management, the system sustains high throughput and real-time feedback, validating its ability to perform in-depth API analysis without compromising responsiveness.

LIMITATIONS

- The offline HIBP dataset is static; it requires regular updates to reflect newly compromised credentials, which may introduce brief windows of reduced security.
- Extreme-scale scanning of more than 10 000 endpoints simultaneously could introduce network-bound latency or congestion, potentially requiring distributed orchestration and load balancing.
- The frontend interface has not been fully stress-tested under ultra-high load; dashboard responsiveness and rendering efficiency may require optimization for very large deployments.
- Certain security checks, such as mTLS handshake validation for every service-to-service request, may impose minor overhead in highly dynamic microservice environments.

Future extensions of the API monitoring system can explore several directions to enhance both security and operational intelligence. One avenue is the integration of a dedicated email server under a managed subdomain, enabling automated notifications, account verification, and secure authentication workflows through OpenID Connect. This approach would facilitate single sign-on and federated identity management across multiple services, improving usability and reducing administrative overhead.

Another promising direction involves advanced geolocation and latency analysis. By combining measured round-trip times (RTTs) with GeoIP data, the system could triangulate endpoint locations, identify geographic anomalies, and detect hosts whose reported location differs from the observed network characteristics. This capability would improve threat detection, support regulatory compliance, and enhance performance-aware routing decisions.

Passive data enrichment offers further insight by leveraging WHOIS, RDAP, and reverse ASN lookups. Pulling registrant information, abuse contacts, and AS operator reputation would allow the system to contextualize endpoints within broader network ecosystems, enabling proactive risk assessment and automated threat correlation.

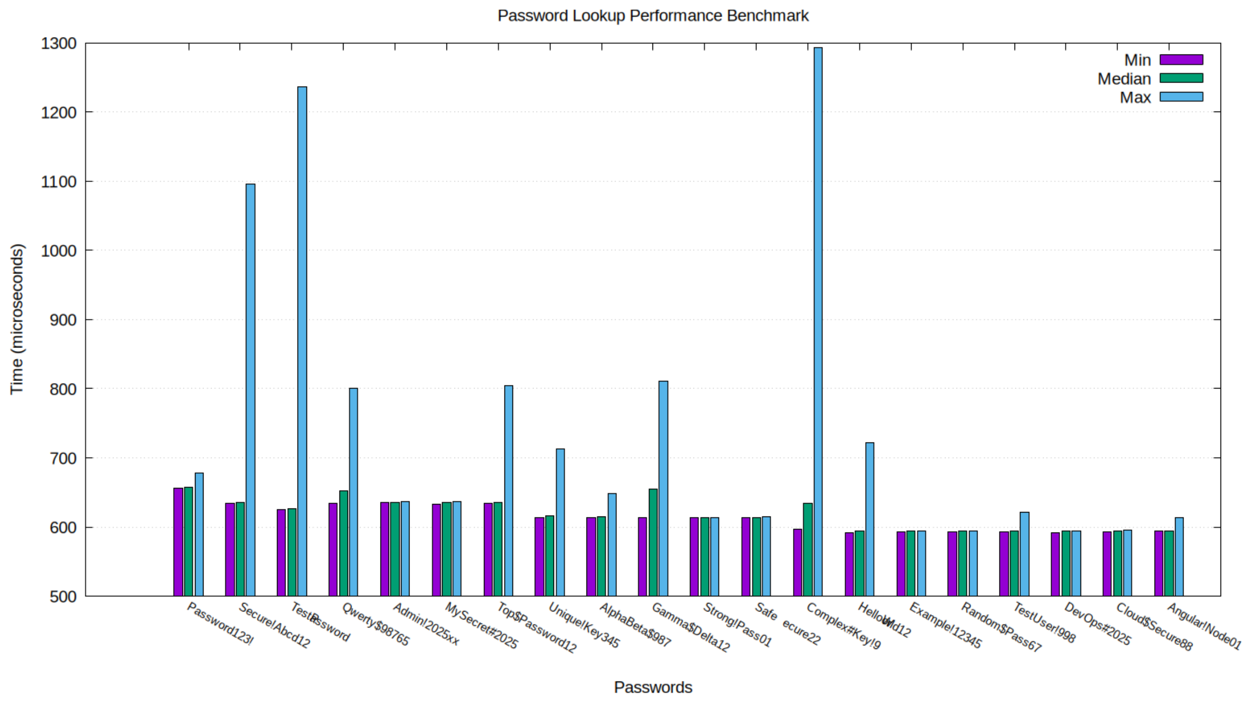


Fig. 4. Have I been PWNED engine.

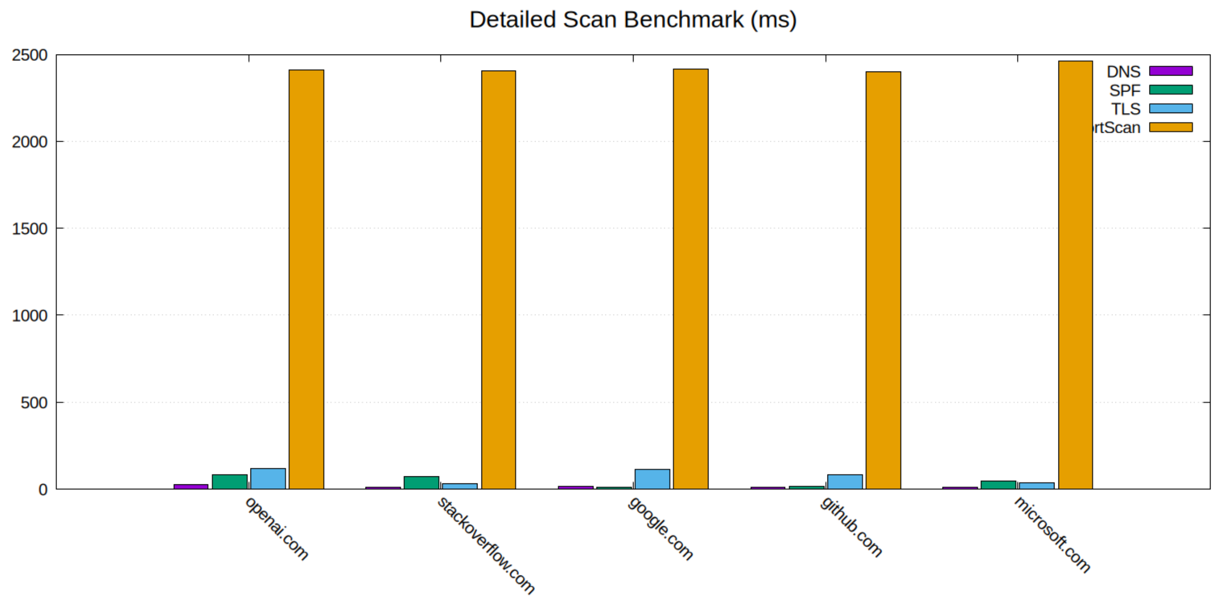


Fig. 5. Scanning performance.

Enhancements to email security monitoring include DMARC parsing and reporting readiness. Validating DMARC policies, alignment modes, and reporting URIs (ruf and rua) ensures endpoints are correctly configured to handle forensic and aggregate reports. Verifying that these endpoints accept mail enables administrators to maintain visibility into unauthorized email activity and prevent domain spoofing.

Finally, active OS and TCP stack fingerprinting could be integrated as an optional module. Techniques such as analysing TTL, TCP window size, options order, IP ID behaviour, and timestamp responses can reveal low-level characteristics of remote hosts, including NAT traversal or load balancer presence. While subtle and potentially noisy, this information can provide a deeper understanding of network infrastructure, correlate suspicious endpoints, and strengthen both operational intelligence and threat detection.

Collectively, these future enhancements would extend the system beyond endpoint monitoring and performance analysis, creating a proactive, intelligence-driven framework capable of supporting advanced security, compliance, and operational insights across distributed API ecosystems.

CONCLUSIONS

The API monitoring and scanning system described in this article demonstrates not only a robust, secure and scalable approach to overseeing critical API endpoints but also the ability to sustain predictable performance under diverse operational conditions. By combining ASP.NET Core microservices, C++ security modules and a Node.js Backend-for-Frontend with Angular, the platform integrates heterogeneous technologies into a single cloud-native architecture orchestrated through Kubernetes, Istio and ArgoCD. This layered design enables real-time metrics collection, event-driven anomaly detection and fine-grained claims-based access control while maintaining low operational overhead.

Performance benchmarks of the offline Have I Been Pwned verification engine (Fig. 4) show sub-millisecond lookup times even in worst-case scenarios, confirming that strong credential policies and hardware-adaptive Argon2id hashing can be enforced without degrading the user experience during registration and authentication. Likewise, detailed scan benchmarks (Fig. 5) demonstrate that DNS, SPF and TLS check complete within tens or hundreds of milliseconds, while longer port-scanning operations are isolated in dedicated pipelines to avoid bottlenecks. Together these results validate the system's ability to deliver comprehensive security and performance assessments of API endpoints while preserving responsiveness and throughput.

The integration of bounded parallelism, producer-consumer channels, reactive pipelines and structured observability further ensures that monitoring and scanning scale to high-volume workloads without thread starvation or back-pressure. Centralized logging, secrets management with Vault and automated testing and deployment pipelines reinforce the reliability and reproducibility of each release, reducing the risk of configuration drift or manual error.

Collectively, these architectural and operational choices provide a technically rigorous framework for sustaining resilient API infrastructures. The system unifies monitoring, scanning and adaptive protection into a single platform that can evolve with future enhancements such as federated identity management, passive data enrichment and advanced geolocation analysis. In doing so, it offers a practical, extensible and intelligence-driven approach to maintaining the performance, security and stability of modern web applications.

REFERENCES

1. A. Vasilev, C# - basics of the language in examples, 2018.
2. API Security in Action - Neil Madden, Manning, 2020.

3. M. Masse, REST API Design Rulebook, O'Reilly Media, 2011.
4. D. Kolisnichenko, SQL - practical programming, Asenevtsi, 2018.
5. E. Wilde, M. Medjaoui, Continuous API Management, O'Reilly Media, 2018.