Baku Higher Oil School

***Advanced Network Security***

**PROJECT REPORT**

**Title: Intrusion Detection System (IDS)**

**Student: Rafael Mehdiyev**

**Supervisor: PhD, Mehman Hematyar**

**Project report submitted: 07.12.2024**

Contents

1. Introduction to Network Security Monitoring [3](#_Toc102295834)

[Overview of Network Security 3](#_Toc102295836)

[Importance of Real-time Monitoring 4](#_Toc102295836)

[Modern Security Challenges 4](#_Toc102295836)

Project Objectives [5](#_Toc102295836)

2. [Network Security Monitoring Architecture 6](#_Toc102295835)

[System Architecture Overview 6](#_Toc102295836)

[Core Components 7](#_Toc102295837)

Data Flow and Processing [9](#_Toc102295838)

3. Intrusion Detection System [9](#_Toc102295843)

[Types of Detection Methods 9](#_Toc102295844)

Alert Generation and Management [11](#_Toc102295844)

Threat Detection Algorithms [12](#_Toc102295844)

Response Mechanisms [13](#_Toc102295844)

Log Management [14](#_Toc102295844)

4.  [Implementation and Technical Details 15](#_Toc102295845)

Development Environment  [15](#_Toc102295846)

[Configuration Management 16](#_Toc102295848)

[API Implementation 17](#_Toc102295849)

Conclusion ………………………………………………………………………………………………………………………………………..21

# 1. Introduction to Network Security Monitoring

**Overview of Network Security**

Network security is a fundamental aspect of protecting digital infrastructures from unauthorized access, misuse, or damage. As the digital landscape evolves, organizations face increasing challenges in safeguarding their networks against sophisticated cyber threats. This project aims to develop a comprehensive network security monitoring system that offers real-time intrusion detection capabilities, ensuring robust protection against potential security breaches.

In today's interconnected world, network security involves implementing a range of measures, including firewalls, intrusion detection systems, and encryption protocols, to secure data and maintain the integrity of network operations. The importance of network security cannot be overstated, as it helps prevent data breaches, protects sensitive information, and ensures business continuity.

Key aspects of network security include:

* **Threat Detection and Prevention:** Identifying and mitigating potential threats before they can cause harm is crucial. This involves monitoring network traffic for unusual patterns and deploying countermeasures to neutralize threats.
* **Access Control:** Ensuring that only authorized users have access to network resources is vital for preventing unauthorized data access and potential breaches.
* **Data Integrity and Confidentiality:** Protecting data from unauthorized modifications and ensuring that sensitive information remains confidential are the core objectives of network security.
* **Incident Response:** Establishing protocols for responding to security incidents helps organizations quickly address and recover from breaches, minimizing potential damage.

**Importance of Real-time Monitoring**

Real-time network monitoring is essential for maintaining robust cybersecurity. It provides immediate visibility into network activities, enabling organizations to detect and respond to threats as they occur. This capability is crucial in today's environment, where cyber-attacks are increasingly sophisticated.

Key benefits of real-time monitoring include:

* **Immediate Threat Detection**: By continuously analyzing network traffic, organizations can quickly identify suspicious activities, allowing for prompt intervention.
* **Proactive Security Measures**: Real-time insights enable the implementation of defenses, such as blocking malicious IPs, before threats escalate.
* **Enhanced Incident Response**: With real-time data, security teams can respond more effectively, reducing the time to mitigate threats and recover from attacks.

Additionally, real-time monitoring supports regulatory compliance by providing detailed logs and reports of network activities. This helps organizations meet industry security standards and maintain trust with clients and stakeholders.

**Modern Security Challenges**

The landscape of network security is constantly evolving, presenting new challenges for organizations striving to protect their digital assets. As cyber threats become more sophisticated, traditional security measures are often insufficient to address the complexities of modern attacks.

One significant challenge is the increasing sophistication of cyber-attacks. Hackers are employing advanced techniques, such as phishing, ransomware, and zero-day exploits, to bypass security defenses. These attacks are often targeted and persistent, making detection and prevention more difficult.

Another challenge is the need for real-time threat detection. With the rapid pace of technological advancements, organizations must be able to identify and respond to threats as they occur. This requires robust monitoring systems capable of analyzing vast amounts of data in real-time.

The complexity of network environments also poses a challenge. As organizations expand their digital footprint, they must manage a diverse array of devices, applications, and services. This complexity can create vulnerabilities that attackers may exploit.

Additionally, the growing number of connected devices, often referred to as the Internet of Things (IoT), introduces new security risks. Many IoT devices lack adequate security measures, making them attractive targets for cybercriminals.

Finally, advanced persistent threats (APTs) represent a significant challenge. These threats involve prolonged and targeted attacks, often orchestrated by well-funded adversaries. APTs require sophisticated detection and response strategies to mitigate their impact.

Addressing these challenges requires a comprehensive approach to network security, incorporating advanced monitoring, threat intelligence, and incident response capabilities.

**Project Objectives**

The primary objective of this project is to develop a comprehensive network security monitoring system with advanced intrusion detection capabilities. This system aims to enhance the security posture of organizations by providing real-time insights into network activities and enabling proactive threat mitigation.

The specific objectives of the project include:

Implement Real-time Network Monitoring: Develop a system capable of continuously monitoring network traffic to detect and respond to potential threats in real-time.

Develop Effective Intrusion Detection Mechanisms: Incorporate advanced detection algorithms to identify and mitigate various types of cyber threats, including port scans, DNS anomalies, and protocol-based attacks.

Create a User-friendly Web Interface: Design a web-based dashboard that provides security teams with intuitive access to network monitoring data, alerts, and configuration settings.

Provide Comprehensive Logging and Reporting Capabilities: Implement a robust logging system that captures detailed information about network events, enabling thorough analysis and reporting.

Enable Configurable Alert Systems: Allow users to customize alert settings based on specific security requirements, ensuring timely notification of potential threats.

By achieving these objectives, the project aims to deliver a powerful tool for network security professionals, enhancing their ability to protect digital assets and maintain the integrity of network operations.

# 2. Network Security Monitoring Architecture

**System Architecture Overview**

The network security monitoring system is designed with a modular architecture that facilitates scalability, maintainability, and flexibility. This architecture comprises several key components, each serving a distinct purpose in the overall functionality of the system.

At the core of the architecture is the **Network Sensor Module**, responsible for capturing and analyzing network traffic. Utilizing advanced packet capture techniques, this module processes data in real-time, enabling the detection of potential threats and anomalies.

Complementing the sensor module is the **Processing Engine**, which aggregates and analyzes the data collected by the sensors. This engine employs various algorithms to identify patterns indicative of security incidents, providing actionable insights to security teams.

**The Web Interface** serves as the user interaction layer, offering a user-friendly dashboard for monitoring network activities, configuring settings, and viewing alerts. Built on the Flask web framework, this interface provides real-time updates and visualizations of network data.

Additionally, the **Configuration Management** component allows users to easily manage security rules and settings. This ensures that the system can adapt to evolving threats and organizational needs without requiring extensive reconfiguration.

Finally, the **Logging System** captures detailed records of network events, facilitating thorough analysis and reporting. This component is crucial for compliance with regulatory standards and for conducting post-incident investigations

Overall, the modular architecture of the network security monitoring system not only enhances its effectiveness in detecting and responding to threats but also ensures that it can evolve alongside emerging security challenges.

IDS/

├── .git/                      # Git repository

├── \_\_pycache\_\_/              # Python cache directory

├── logs/                     # Log files directory

├── templates/                # HTML templates

│   ├── config.html          # Configuration page

│   ├── index.html           # Main dashboard

│   └── logs.html            # Log viewer page

├── utils/                    # Utility functions and helpers

│   └── detailed\_logger.py   # Detailed logging implementation

├── config.json              # Configuration file

├── requirements.txt         # Python dependencies

├── sensor.py               # Network sensor implementation

└── server.py             # Main server application

**Core Components**

The network security monitoring system is composed of several core components that work together to provide comprehensive security coverage. Each component plays a vital role in ensuring the system's effectiveness in detecting and responding to cyber threats.

**1. Network Sensor Module (sensor.py)**

The Network Sensor Module is responsible for capturing and analyzing network packets in real-time. Utilizing the Scapy library, this component performs packet sniffing, allowing it to monitor various protocols and detect anomalies in network traffic. The sensor generates real-time statistics and alerts based on predefined rules, ensuring that potential threats are identified promptly.

**2. Processing Engine**

The Processing Engine aggregates data from the Network Sensor Module and applies advanced detection algorithms to analyze traffic patterns. This component is crucial for identifying suspicious activities, such as port scans and unusual data flows. By processing data in real-time, the engine enables quick decision-making and response actions.

**3. Web Interface (server.py)**

The Web Interface serves as the user interaction layer, providing a dashboard for security professionals to monitor network activities, configure settings, and view alerts. Built on the Flask web framework, this component offers a responsive and intuitive user experience, allowing users to access critical information briefly.

**4. Configuration Management (config.json)**

The Configuration Management component utilizes a JSON-based configuration file to store security rules and settings. This modular approach allows for easy updates and modifications to security policies, enabling the system to adapt to evolving threats and organizational requirements.

**5. Logging System**

The Logging System captures detailed records of network events and security incidents. This component is essential for compliance with regulatory standards and for conducting post-incident analyses. By maintaining comprehensive logs, the system facilitates a thorough understanding of network activities and potential vulnerabilities.

Together, these core components form a cohesive network security monitoring system that is capable of detecting, analyzing, and responding to cyber threats effectively.

**Data Flow and Processing**

The data flow within the network security monitoring system is meticulously designed to ensure efficient processing and timely threat detection. This system leverages a structured approach to handle network data, enabling comprehensive analysis and rapid response to potential security incidents.

Initially, network packets are captured by the Network Sensor Module. This module employs advanced packet sniffing techniques to monitor traffic across various protocols. The captured data is then forwarded to the Processing Engine, where it undergoes thorough analysis. The engine utilizes sophisticated algorithms to identify patterns indicative of malicious activities, such as unusual data flows or unauthorized access attempts.

Once analyzed, the data is categorized based on threat levels and processed accordingly. High-priority threats trigger immediate alerts, which are communicated to the security team via the Web Interface. This interface provides a real-time dashboard, allowing security professionals to monitor network activities and respond to incidents swiftly.

Simultaneously, the system logs all network events, capturing detailed information for further examination. These logs are stored in a structured format, facilitating easy retrieval and analysis. The Logging System plays a crucial role in maintaining a comprehensive record of network activities, supporting both real-time monitoring and post-incident investigations.

# 3. Intrusion Detection System

**Types of Detection System**

The network security monitoring system developed in this project employs a multifaceted approach to threat detection, leveraging both **signature-based** and **anomaly-based** methods to ensure comprehensive coverage.

**Signature-based Detection** is implemented through predefined rules that identify known threats by matching network traffic patterns against a database of known attack signatures. This method is particularly effective in detecting well-documented threats, such as common malware and intrusion attempts, providing a first line of defense. This method is implemented in the **check\_ip\_rules** function, where the system checks traffic involving blacklisted IPs or specific IP ranges.

def check\_ip\_rules(self, src\_ip, dst\_ip):

        """Check IP against configured rules"""

        ip\_rules = self.config.get("IP\_RULES", {})

        # Check blacklisted IPs

        if src\_ip in ip\_rules.get("BLACKLISTED\_IPS", []) or dst\_ip in ip\_rules.get("BLACKLISTED\_IPS", []):

            self.add\_alert(f"Detected traffic from/to blacklisted IP - Src: {src\_ip}, Dst: {dst\_ip}", "ALERT")

            return True

        # Check IP ranges

        for ip\_range in ip\_rules.get("IP\_RANGES\_TO\_MONITOR", []):

            if self.ip\_in\_range(src\_ip, ip\_range) or self.ip\_in\_range(dst\_ip, ip\_range):

                self.add\_alert(f"Detected traffic in monitored IP range - Src: {src\_ip}, Dst: {dst\_ip}", "INFO")

                return True

        return False

**Anomaly-based Detection** is achieved by monitoring network traffic for deviations from established baseline patterns. The system utilizes statistical models to identify unusual activities, such as unexpected spikes in traffic or irregular access attempts. This approach is crucial for detecting zero-day attacks and other novel threats that do not yet have known signatures. This method is implemented in the **check\_dns\_rules** function, where the system analyzes DNS queries for blacklisted domains, suspicious top-level domains (TLDs), and potential domain generation algorithm (DGA) domains.

def check\_dns\_rules(self, packet, dns\_packet):

        """Enhanced DNS analysis with new rules"""

        dns\_rules = self.config.get("DNS\_RULES", {})

        if dns\_packet.qr == 0:  # DNS query

            query = dns\_packet.qd.qname.decode('utf-8').lower().rstrip('.')

            # Check DNS blacklist

            if query in dns\_rules.get("DNS\_BLACKLIST", []):

                self.add\_alert(f"Query to blacklisted domain detected: {query}", "ALERT")

                return True

            # Check suspicious TLDs

            for tld in dns\_rules.get("DNS\_MONITORING", {}).get("SUSPICIOUS\_TLD", []):

                if query.endswith(tld):

                    self.add\_alert(f"Query to suspicious TLD detected: {query}", "WARNING")

                    return True

            # DGA detection

            if dns\_rules.get("DGA\_DETECTION", {}).get("ENABLED", False):

                if self.check\_dga(query):

                    self.add\_alert(f"Possible DGA domain detected: {query}", "WARNING")

                    return True

        return False

In addition to these primary detection methods, the system incorporates Real-time Monitoring capabilities, facilitated by the **sensor.py** module. This module captures and analyzes network packets, generating alerts for suspicious activities. The alerts are then processed and displayed through the web interface, allowing security teams to respond promptly to potential threats.

**Alert Generation and Management**

The network security monitoring system includes a robust alert generation and management framework designed to notify security personnel of potential threats in real-time. This framework is integral to the system's ability to respond promptly to security incidents, thereby minimizing potential damage.

**Alert Generation** is triggered by various detection mechanisms embedded within the system. For instance, the system generates alerts when it detects traffic from blacklisted IPs, queries to blacklisted domains, or potential port scanning activities. These alerts are categorized based on severity, ranging from informational messages to critical alerts requiring immediate attention.

The alert generation process is supported by the **sensor.py** module, which continuously monitors network traffic and applies detection rules. When a rule is triggered, an alert is created and logged, providing detailed information about the incident, including the source and nature of the threat.

def add\_alert(self, alert\_msg, alert\_type="INFO"):

        """Enhanced alert handling with configurable actions"""

        try:

            with self.display\_lock:

                timestamp = datetime.now().strftime('%H:%M:%S')

                alert\_settings = self.config.get("ALERT\_SETTINGS", {})

                alert\_level = alert\_settings.get("ALERT\_LEVELS", {}).get(alert\_type, {})

                # Get color based on alert type

                color = getattr(Fore, alert\_level.get("COLOR", "WHITE"))

                formatted\_alert = f"{color}[{timestamp}] {alert\_msg}{Style.RESET\_ALL}"

                # Add to history and increment counter

                self.alert\_history.append(formatted\_alert)

                self.alert\_count += 1

                # Log if enabled

                if alert\_level.get("LOG", True):

                    logging.warning(alert\_msg)

                # Handle notifications

                if alert\_level.get("NOTIFY", False):

                    self.handle\_notification(alert\_msg, alert\_type)

                # Handle auto-blocking

                if alert\_type == "ALERT" and alert\_settings.get("ALERT\_ACTIONS", {}).get("AUTO\_BLOCK", {}).get("ENABLED", False):

                    self.handle\_auto\_block(alert\_msg)

        except Exception as e:

            logging.error(f"Error updating display: {str(e)}")

This function, **add\_alert**, is responsible for generating alerts based on detected security events. It formats the alert message, logs it, and handles notifications and potential auto-blocking actions based on the alert type and configured settings.

**Threat Detection Algorithms**

The network security monitoring system employs a variety of threat detection algorithms designed to identify and respond to potential cyber threats effectively. These algorithms form the backbone of the system's intrusion detection capabilities, ensuring comprehensive security coverage.

**Port Scan Detection** is implemented using a time-based threshold algorithm. This method tracks the number of distinct ports accessed by a single IP address within a specified time window. If the number of accessed ports exceeds the predefined threshold, the system flags the activity as a potential port scan, triggering an alert. This algorithm helps detect reconnaissance activities often used by attackers to identify vulnerable services.

def check\_port\_scan(self, src\_ip, dst\_port):

        """Detect potential port scanning"""

        port\_rules = self.config.get("PORT\_RULES", {}).get("PORT\_SCAN\_DETECTION", {})

        threshold = port\_rules.get("THRESHOLD", 20)

        time\_window = port\_rules.get("TIME\_WINDOW", 60)

        current\_time = time.time()

        if src\_ip not in self.port\_scan\_tracker:

            self.port\_scan\_tracker[src\_ip] = {"ports": set(), "start\_time": current\_time}

        tracker = self.port\_scan\_tracker[src\_ip]

        tracker["ports"].add(dst\_port)

        if current\_time - tracker["start\_time"] <= time\_window:

            if len(tracker["ports"]) >= threshold:

                self.add\_alert(f"Possible port scan detected from {src\_ip} - {len(tracker['ports'])} ports in {time\_window}s", "ALERT")

                return True

        else:

            # Reset tracker after time window

            tracker["ports"] = {dst\_port}

            tracker["start\_time"] = current\_time

        return False

**IP and DNS Blacklist Checks** are integral to the system's signature-based detection approach. The system maintains a list of known malicious IP addresses and domains, which is used to monitor network traffic. Any communication involving these blacklisted entities is immediately flagged, allowing for swift intervention to prevent data breaches or malware infections.

**Anomaly Detection in DNS Queries** is achieved through the analysis of domain query patterns. The system evaluates DNS queries for suspicious characteristics, such as uncommon top-level domains (TLDs) or potential domain generation algorithm (DGA) domains. By identifying anomalies in DNS traffic, the system can detect and respond to emerging threats that do not match known signatures.

**Response Mechanisms**

The network security monitoring system is equipped with several response mechanisms designed to mitigate the impact of detected threats and maintain network integrity. These mechanisms are crucial for the system's ability to respond swiftly and effectively to security incidents.

**Automated Alerting** is a primary response mechanism implemented in the system. The **add\_alert** method in the **sensor.py** module generates alerts based on detected threats and categorizes them by severity. This allows for prioritized responses to critical incidents, ensuring that security personnel are immediately notified and can take appropriate actions.

**Auto-blocking of Malicious IPs** is another implemented feature. The system can automatically block IP addresses involved in suspicious activities, such as port scans or communication with blacklisted entities. This proactive measure helps prevent further unauthorized access attempts and potential data breaches.

**Incident Logging and Reporting** are integral to the system's response strategy. The **sensor.py** module logs all detected threats and response actions, providing a comprehensive record of security events. This logging capability supports post-incident analysis and helps organizations refine their security strategies over time.

**Log Management**

Log management is a critical component of the network security monitoring system, providing a detailed record of network activities and security events. This functionality supports both real-time monitoring and post-incident analysis, ensuring that organizations can maintain a comprehensive understanding of their network security posture.

**Detailed Event Logging** is implemented within the **sensor.py** module. The system logs all detected threats, including alerts generated by the various detection algorithms. Each log entry includes a timestamp and detailed information about the event, enabling security teams to track and analyze incidents over time.

**Configurable Log Levels** allow organizations to customize the granularity of their logging. The system supports different log levels, such as INFO, WARNING, and ALERT, ensuring that only relevant information is recorded based on the organization's security policies and operational needs.

"ALERT\_SETTINGS": {

        "ALERT\_LEVELS": {

            "INFO": {

                "COLOR": "WHITE",

                "LOG": true

            },

            "WARNING": {

                "COLOR": "YELLOW",

                "LOG": true

            },

            "ALERT": {

                "COLOR": "RED",

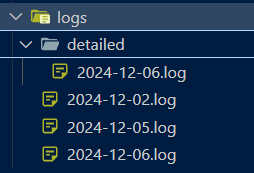
                "LOG": true,

                "NOTIFY": true

            }

        },

**Separate Logs for Different Event Types** are maintained to facilitate efficient log management and retrieval. The system categorizes logs by event type, such as network traffic anomalies or security alerts, allowing for targeted analysis and reporting.



The log management capabilities of the network security monitoring system are designed to support comprehensive security monitoring and incident response. By maintaining detailed and organized logs, the system enables organizations to enhance their threat detection and response capabilities, ensuring robust protection against cyber threats.

# 4. Implementation and Technical Details

**Development Environment**

The development environment for the network security monitoring system is carefully configured to support the project's requirements and ensure seamless integration of its components. This environment leverages a combination of tools and technologies to facilitate efficient development, testing, and deployment.

**Programming Language:** The system is developed using Python 3.11, chosen for its versatility and extensive library support, particularly in the fields of network programming and data analysis.

**Libraries and Frameworks:** Key libraries and frameworks utilized in the project include:

* **Scapy:** Used for network packet manipulation, enabling the system to capture and analyze network traffic effectively.
* **Flask:** Provides the web framework for the system's user interface, facilitating the development of a responsive and user-friendly dashboard.
* **Colorama:** Enhances console output with color coding, improving the readability of alerts and logs in the terminal.
* **Threading:** Supports concurrent processing, allowing the system to handle multiple tasks simultaneously without compromising performance.

**Configuration Management:** The project utilizes a JSON-based configuration file, **config.json**, to manage various settings, including network interfaces, blacklisted IPs, and alert configurations. This approach ensures flexibility and ease of customization, allowing the system to adapt to different network environments and security policies.

**Version Control:** Git is used for version control, providing a robust platform for tracking changes, collaborating with team members, and maintaining a history of the project's development.

**Development Tools:** The development environment is set up on a Windows platform, with tools such as Visual Studio Code used for code editing and debugging. This setup supports efficient development workflows and facilitates the integration of various project components.

The development environment is designed to support the comprehensive requirements of the network security monitoring system, ensuring that the project can be developed, tested, and deployed effectively.

**Configuration Management**

The configuration management system is a vital component of the network security monitoring system, providing flexibility and control over various operational parameters. This system ensures that the network security tools can be adapted to different environments and requirements without altering the underlying codebase.

**Configuration File:** The project utilizes a JSON-based configuration file, **config.json**, to store and manage settings. This file includes parameters such as network interface settings, blacklisted IPs, suspicious ports, and alert configurations. The use of JSON allows for easy readability and modification of configuration settings.

**Dynamic Configuration Loading:** The system is designed to load configuration settings dynamically at runtime. This feature ensures that any changes made to the configuration file are immediately reflected in the system's operation, allowing for real-time adjustments to security policies and monitoring parameters.

**Hierarchical Configuration Structure:** The configuration file is structured hierarchically, organizing settings into categories such as **GENERAL\_SETTINGS**, **IP\_RULES**, **PORT\_RULES**, and **DNS\_RULES**. This structure facilitates easy navigation and management of configuration options, enabling users to quickly locate and modify specific settings.

**Default and Custom Configurations:** The system supports both default and custom configurations. Default settings are provided to ensure the system operates effectively out of the box, while custom configurations allow users to tailor the system to their specific network environment and security requirements.

**Configuration Management Interface:** The web interface includes a configuration managemet section, enabling users to view and modify settings directly through the user interface. This feature enhances usability by providing a convenient and accessible way to manage configurations without requiring direct access to the configuration file.

**API Implementation**

The API implementation in the network security monitoring system is designed to facilitate seamless interaction between the server and client components, enabling efficient data exchange and control operations. This section outlines the key aspects of the API implementation, focusing on its architecture, functionalities, and integration with other system components.

**RESTful Architecture:** The API is built using a RESTful architecture, leveraging HTTP methods to perform operations on resources. This approach ensures scalability and flexibility, allowing the system to support a wide range of client applications and integration scenarios.

**Endpoints and Operations:** The API provides a set of endpoints to manage various aspects of the network security monitoring system. Key operations include:

* **Configuration Management:** Endpoints for retrieving and updating configuration settings, enabling dynamic adjustments to system parameters.

@app.route('/api/config', methods=['GET'])

@requires\_auth

def get\_config():

    """API endpoint to get current configuration"""

    return jsonify(load\_config())

* **Log Retrieval:** Endpoints for accessing security logs, supporting both real-time monitoring and historical analysis.

@app.route('/logs')

@requires\_auth

def logs():

    """Render the logs page"""

    return render\_template('logs.html')

@app.route('/api/logs/today')

@requires\_auth

def get\_today\_logs():

    """API endpoint to get all logs from today"""

    try:

        log\_file = log\_reader.get\_current\_log\_file()

        if os.path.exists(log\_file):

            with open(log\_file, 'r') as f:

                logs = f.readlines()

                return jsonify({

                    'logs': [log.strip() for log in logs],

                    'count': len(logs)

                })

        return jsonify({'logs': [], 'count': 0})

    except Exception as e:

        return jsonify({'error': str(e)}), 500

@app.route('/api/logs/download')

@requires\_auth

def download\_logs():

    """Download today's logs as a text file"""

    try:

        log\_file = log\_reader.get\_current\_log\_file()

        if os.path.exists(log\_file):

            return send\_file(

                log\_file,

                mimetype='text/plain',

                as\_attachment=True,

                download\_name=f'security\_logs\_{datetime.now().strftime("%Y-%m-%d")}.txt'

            )

        return jsonify({'error': 'No logs found for today'}), 404

    except Exception as e:

        return jsonify({'error': str(e)}), 500

@app.route('/api/logs/dates')

@requires\_auth

def get\_available\_log\_dates():

    """Get list of dates that have log files"""

    try:

        available\_dates = []

        for file in os.listdir(log\_reader.log\_dir):

            if file.endswith('.log'):

                date\_str = file[:-4]  # Remove .log extension

                try:

                    # Validate it's a proper date

                    datetime.strptime(date\_str, '%Y-%m-%d')

                    available\_dates.append(date\_str)

                except ValueError:

                    continue

        return jsonify({

            'dates': sorted(available\_dates, reverse=True),

            'current': datetime.now().strftime('%Y-%m-%d')

        })

    except Exception as e:

        return jsonify({'error': str(e)}), 500

@app.route('/api/logs/<date>')

@requires\_auth

def get\_logs\_for\_date(date):

    """Get logs for a specific date"""

    try:

        # Validate date format

        try:

            datetime.strptime(date, '%Y-%m-%d')

        except ValueError:

            return jsonify({'error': 'Invalid date format'}), 400

        log\_file = os.path.join(log\_reader.log\_dir, f"{date}.log")

        if not os.path.exists(log\_file):

            return jsonify({'error': 'No logs found for this date'}), 404

        with open(log\_file, 'r') as f:

            logs = f.readlines()

            return jsonify({

                'logs': [log.strip() for log in logs],

                'count': len(logs)

            })

    except Exception as e:

        return jsonify({'error': str(e)}), 500

* **Alert Management:** Endpoints for querying and managing security alerts, allowing for efficient incident response and reporting.

**Authentication and Security:** The API is secured using token-based authentication, ensuring that only authorized users can access sensitive operations and data. This mechanism helps protect the system from unauthorized access and potential security breaches.

**Integration with Web Interface:** The API is seamlessly integrated with the system's web interface, providing a responsive and user-friendly experience. This integration allows users to interact with the system through a browser, accessing real-time data and performing management tasks without the need for specialized software.

@app.route('/')

@requires\_auth

def index():

    """Render the main page"""

    return render\_template('index.html')

@app.route('/events')

@requires\_auth

def get\_events():

    """Event stream for real-time updates"""

    def generate():

        # Set initial position to current file size when client connects

        log\_file = log\_reader.get\_current\_log\_file()

        if os.path.exists(log\_file):

            log\_reader.last\_position = os.path.getsize(log\_file)

        # Do the same for JSON log reader

        json\_file = json\_log\_reader.get\_current\_log\_file()

        if os.path.exists(json\_file):

            json\_log\_reader.current\_position = os.path.getsize(json\_file)

        # Send initial connection message

        yield f"data: {json.dumps({'type': 'system', 'message': 'Connected to event stream'})}\n\n"

        while True:

            # Get regular logs

            new\_logs = log\_reader.read\_new\_logs()

            # Get JSON logs

            new\_json\_logs = json\_log\_reader.read\_new\_logs()

            # Send regular logs

            for log in new\_logs:

                yield f"data: {json.dumps({'type': 'log', 'message': log.strip()})}\n\n"

            # Send JSON logs with type identifier

            for log in new\_json\_logs:

                yield f"data: {json.dumps({'type': 'json', 'message': log})}\n\n"

            # Keep-alive

            if not new\_logs and not new\_json\_logs:

                yield ": keep-alive\n\n"

            time.sleep(1)

    return Response(generate(), mimetype='text/event-stream')

**Scalability and Extensibility:** The API is designed to be scalable and extensible, supporting future enhancements and additional functionalities. This design ensures that the system can evolve to meet changing security requirements and technological advancements.

The API implementation is a critical component of the network security monitoring system, providing the necessary infrastructure for efficient communication and control. By leveraging modern web technologies and security practices, the API ensures robust and reliable operation in diverse network environments.

# Conclusion

The Advanced Network Security Monitoring System developed in this project provides a comprehensive solution for real-time threat detection and response. By integrating signature-based and anomaly-based detection methods, the system effectively identifies and mitigates potential security threats. The modular architecture, combined with a user-friendly web interface, ensures that the system is both scalable and adaptable to various network environments.

Key features such as configurable alert levels, detailed logging, and dynamic configuration management enhance the system's usability and effectiveness. The implementation of a RESTful API further facilitates seamless interaction between system components, enabling efficient data exchange and control operations.

Overall, this project demonstrates the potential of leveraging modern technologies to enhance network security. Future improvements could focus on expanding detection capabilities, improving performance, and integrating additional security features to address emerging threats.