Republic of Tunisia
Ministry of Higher Education and Scientific Research

TEK-UP University
Department of Cyber Security

Network Intrusion Detection System

Real-Time Threat Monitoring with Intelligence Integration

Defense Report for a Python-Based Project
in Cybersecurity Engineering

Presented by: Dhia Medhioub Chaima Khalsi
Supervised by: ING. Oussama BAK ALI

Academic Year: 2025-2026

Abstract

This report presents the design, implementation, and evaluation of a professional-grade Network Intrusion Detection System (IDS) developed as part of an end-of-studies project. The system addresses the critical need for real-time network security monitoring in modern computing environments, where cyber threats are increasingly sophisticated and prevalent.

The implemented solution integrates multiple advanced features: real-time packet capture and analysis using the Scapy framework, intelligent threat detection algorithms for identifying port scans and Distributed Denial of Service (DDoS) attacks, and integration with the AbuseIPDB threat intelligence platform for IP reputation scoring. The system features a modern web-based dashboard built with Flask and SocketIO, providing security analysts with live threat visualization, interactive charts, and comprehensive alert management capabilities.

A role-based access control (RBAC) system ensures proper user management with distinct privileges for administrators and analysts. The architecture employs SQLite for persistent data storage, enabling historical analysis and audit trail capabilities. Optional multi-channel notification systems support Email, Slack, and Discord integration for critical alert delivery.

Experimental results demonstrate the system's effectiveness in detecting various network attacks with configurable sensitivity thresholds. The threat intelligence integration successfully enriches alerts with abuse confidence scores and historical threat data, significantly improving the decision-making capabilities of security analysts.

This project contributes a fully functional, extensible IDS suitable for educational purposes and small to medium-scale network environments, while demonstrating proficiency in network security concepts, Python development, web technologies, and cybersecurity best practices.

Keywords: Intrusion Detection System, Network Security, Threat Intelligence, Real-time Monitoring, Python, Scapy, Flask, Cybersecurity

Contents

List of Figures

List of Tables

Chapter 1

Introduction

1.1 Context and Motivation

In the contemporary digital landscape, network security has become a paramount concern for organizations of all sizes. With the exponential growth of internet-connected devices and the increasing sophistication of cyber attacks, traditional perimeter security measures alone are insufficient. According to recent cybersecurity reports, organizations face an average of 270 days to detect a breach, highlighting the critical need for robust intrusion detection mechanisms.

An Intrusion Detection System (IDS) serves as a crucial component of defense-in-depth security strategies. Unlike firewalls that prevent unauthorized access, IDS monitors network traffic for suspicious activities and known attack patterns, providing real-time alerts to security personnel. This continuous monitoring capability is essential for identifying zero-day exploits, insider threats, and sophisticated multi-stage attacks that may bypass other security controls.

The motivation for this project stems from the need to develop a practical, educational IDS that demonstrates core network security principles while remaining accessible for learning and experimentation. Traditional commercial IDS solutions, such as Snort or Suricata, while powerful, present a steep learning curve for students and small organizations. This project aims to balance functionality with comprehensibility, providing a platform for understanding intrusion detection mechanisms without the complexity of enterprise-grade systems.

1.2 Problem Statement

Modern network environments face several critical security challenges:

1. **Limited Visibility**: Many organizations lack real-time visibility into network traffic patterns and potential security incidents.

2. **Alert Fatigue**: Security teams are overwhelmed by false positives from poorly tuned detection systems.

3. **Delayed Response**: Without automated detection and notification, security incidents may go unnoticed for extended periods.

4. **Resource Constraints**: Small to medium organizations often cannot afford expensive commercial IDS solutions or dedicated security operations centers.
5. **Context Deficiency**: Standalone detection systems lack contextual threat intelligence to prioritize alerts effectively.

This project addresses these challenges by developing an open-source, lightweight IDS with integrated threat intelligence, providing immediate value for educational purposes and potential production deployment in resource-constrained environments.

1.3 Project Objectives

The primary objectives of this project are:

1. **Real-time Threat Detection**: Implement packet-level analysis to identify common network attacks including port scans and DDoS attempts.
2. **Threat Intelligence Integration**: Enhance detection capabilities with real-time IP reputation data from external threat intelligence platforms.
3. **User-Friendly Interface**: Develop an intuitive web-based dashboard for monitoring, analysis, and alert management.
4. **Configurable Detection**: Allow customization of detection thresholds and rules to adapt to different network environments.
5. **Access Control**: Implement role-based access control to support multi-user environments with appropriate privilege separation.
6. **Data Persistence**: Ensure all security events are logged to a database for historical analysis and forensics.
7. **Extensibility**: Design a modular architecture that facilitates future enhancements and additional detection rules.

Secondary objectives include demonstrating proficiency in network programming, web application development, database design, and cybersecurity analysis techniques.

Chapter 2

Background & Theoretical Concepts

2.1 Intrusion Detection Systems

2.1.1 IDS Classification

Intrusion Detection Systems are broadly classified into two categories based on their detection methodology:

Signature-based Detection relies on known attack patterns stored in a signature database. This approach offers high accuracy for known threats but cannot detect novel or zero-day attacks. Our system implements signature-based detection for port scans and DDoS attacks through configurable rule sets.

Anomaly-based Detection establishes a baseline of normal network behavior and alerts on deviations. While capable of detecting unknown attacks, this approach suffers from higher false positive rates. Future extensions of this project could incorporate machine learning-based anomaly detection.

2.1.2 Network-based vs. Host-based IDS

Network-based IDS (NIDS) monitors network traffic at strategic points, analyzing packets for malicious patterns. Our implementation follows the NIDS model, capturing and analyzing packets at the network interface level.

Host-based IDS (HIDS) operates on individual hosts, monitoring system calls, log files, and file integrity. While outside the scope of this project, HIDS provides complementary detection capabilities.

2.2 Network Attack Patterns

2.2.1 Port Scanning

Port scanning is a reconnaissance technique where attackers probe a target system's open ports to identify running services. Common scanning techniques include:

- **TCP Connect Scan**: Completes the full TCP three-way handshake
- **SYN Scan**: Sends SYN packets without completing the handshake
- **UDP Scan**: Probes UDP ports for open services

Detection relies on identifying abnormal connection patterns, such as connections to multiple sequential ports from a single source IP within a short time window.

2.2.2 Denial of Service Attacks

Distributed Denial of Service (DDoS) attacks overwhelm target systems with excessive traffic, rendering services unavailable to legitimate users. Detection mechanisms monitor for:

- Abnormally high packet rates from individual or distributed sources
- SYN flood attacks exploiting TCP connection state management
- Application-layer floods targeting specific services

Our system implements rate-based detection with configurable thresholds to identify potential DDoS attempts.

2.3 Threat Intelligence

Threat intelligence platforms aggregate data about malicious IP addresses, domains, and attack patterns from multiple sources. Integration with threat intelligence enhances IDS capabilities by:

1. Providing context for detected events (e.g., known botnet IP, scanning IP)
2. Enabling risk-based alert prioritization
3. Reducing false positives through reputation scoring
4. Facilitating proactive blocking of known malicious actors

This project integrates with AbuseIPDB, a community-driven threat intelligence platform that aggregates abuse reports globally.

2.4 Related Technologies

2.4.1 Python and Scapy

Python's extensive networking libraries and ease of development make it ideal for IDS implementation. Scapy, a powerful packet manipulation library, provides:

- Low-level packet capture and crafting capabilities
- Protocol decoding for analysis
- Filtering mechanisms for efficient packet processing

2.4.2 Web Technologies

The dashboard utilizes modern web technologies:

- **Flask**: Lightweight Python web framework
- **SocketIO**: WebSocket implementation for real-time communication
- **Chart.js**: JavaScript library for data visualization
- **SQLite**: Embedded relational database for data persistence

2.5 Related Work

Several open-source IDS projects have influenced this work:

Snort, developed by Cisco, is the de facto standard open-source IDS. While highly capable, its configuration complexity and C-based implementation present barriers for educational use.

Suricata offers multi-threaded performance and advanced protocol detection but similarly requires significant expertise to deploy and maintain.

Security Onion provides a comprehensive security monitoring platform but is resource-intensive and designed for dedicated security infrastructure.

This project differentiates itself through:

- Python-based implementation for accessibility and education
- Integrated web dashboard eliminating external monitoring tools
- Built-in threat intelligence integration
- Simplified deployment suitable for learning environments

Chapter 3

System Architecture & Design

3.1 Global Architecture

The system follows a modular, layered architecture designed for maintainability and extensibility. Figure 3.1 illustrates the overall system structure.

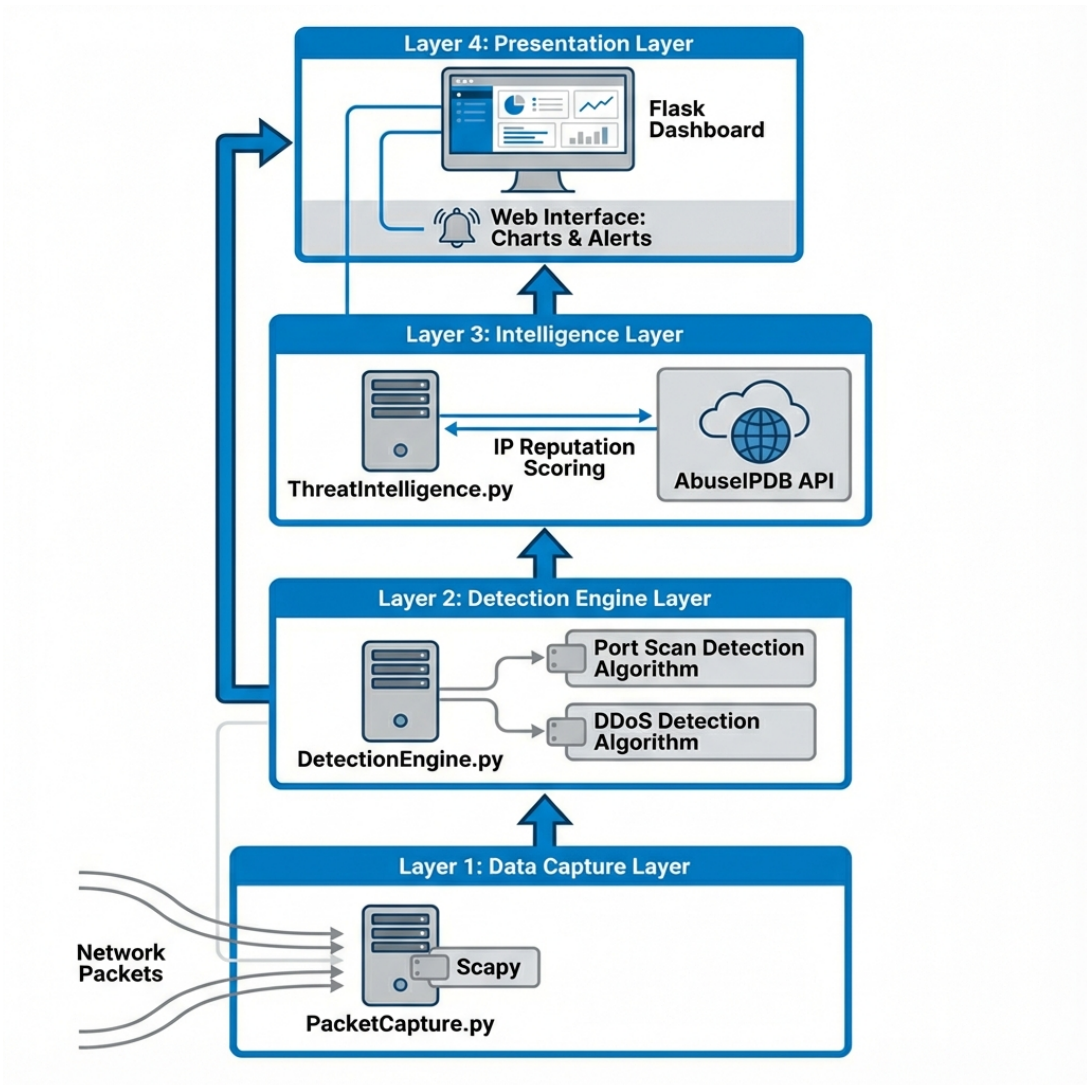

Layer 4: Presentation Layer
Flask Dashboard
Web Interface: Charts & Alerts
Layer 3: Intelligence Layer
IP Reputation Scoring
ThreatIntelligence.py
AbuseIPDB API
Layer 2: Detection Engine Layer
Port Scan Detection Algorithm
DDoS Detection Algorithm
DetectionEngine.py
Layer 1: Data Capture Layer
Network Packets
Scapy
PacketCapture.py

Figure 3.1: System Architecture Overview

The architecture consists of four primary layers:

1. **Data Capture Layer**: Responsible for packet sniffing and raw data collection
2. **Detection Engine Layer**: Analyzes captured packets and applies detection rules
3. **Intelligence Layer**: Enriches alerts with external threat intelligence
4. **Presentation Layer**: Web-based interface for monitoring and management

3.2 Component Descriptions

3.2.1 Packet Capture Module

`PacketCapture.py` implements the interface to Scapy for network traffic collection:

```python
class PacketCapture:
    def __init__(self, interface="eth0"):
        self.interface = interface

    def start_sniffing(self, packet_callback):
        """Start capturing packets"""
        sniff(iface=self.interface,
              prn=packet_callback,
              store=False)
```

Listing 3.1: Packet Capture Interface

Key design decisions:

- Non-storing mode (`store=False`) for memory efficiency
- Callback-based processing for real-time analysis
- Configurable network interface selection

3.2.2 Detection Engine

`DetectionEngine.py` implements the core detection logic with pluggable rule architecture:

```python
class DetectionEngine:
    def __init__(self):
        self.rules = {
            'port_scan': self._detect_port_scan,
            'ddos': self._detect_ddos
        }
        self.trackers = defaultdict(dict)

    def detect_threats(self, packet_features):
        threats = []
        for rule_name, rule_func in self.rules.items():
            if rule_func(packet_features):
                threats.append({'rule': rule_name,
                                'confidence': 0.9})
        return threats
```

Listing 3.2: Detection Engine Structure

Detection algorithms employ stateful tracking:

- **Port Scan Detection**: Tracks unique destination ports per source IP within time windows
- **DDoS Detection**: Monitors request rates per source IP against configurable thresholds

3.2.3 Threat Intelligence Module

`ThreatIntelligence.py` integrates with the AbuseIPDB API:

```python
class ThreatIntelligence:
    def check_ip(self, ip_address):
        """Query AbuseIPDB for IP reputation"""
        if self._is_private_ip(ip_address):
            return None

        cached = self._get_from_cache(ip_address)
        if cached:
            return cached

        reputation = self._query_abuseipdb(ip_address)
        self._add_to_cache(ip_address, reputation)
        return reputation
```

Listing 3.3: Threat Intelligence Integration

Features implemented:

- 24-hour result caching to minimize API usage
- Automatic private IP filtering
- Graceful degradation when API unavailable
- Rate limiting compliance

3.2.4 Alert System

`AlertSystem.py` manages alert generation and distribution:

- Enriches alerts with threat intelligence data
- Persists alerts to database
- Publishes real-time updates via WebSocket
- Triggers optional notification channels (Email, Slack, Discord)

3.2.5 Web Dashboard

The Flask-based dashboard (`app.py`) provides:

- Real-time threat feed with WebSocket updates
- Interactive data visualization using Chart.js
- IP reputation indicators (red/yellow/green badges)
- User authentication and RBAC
- CSV export functionality
- User management interface (admin only)

3.3 Data Flow

Figure 3.2 illustrates the complete data flow from packet capture to alert presentation:

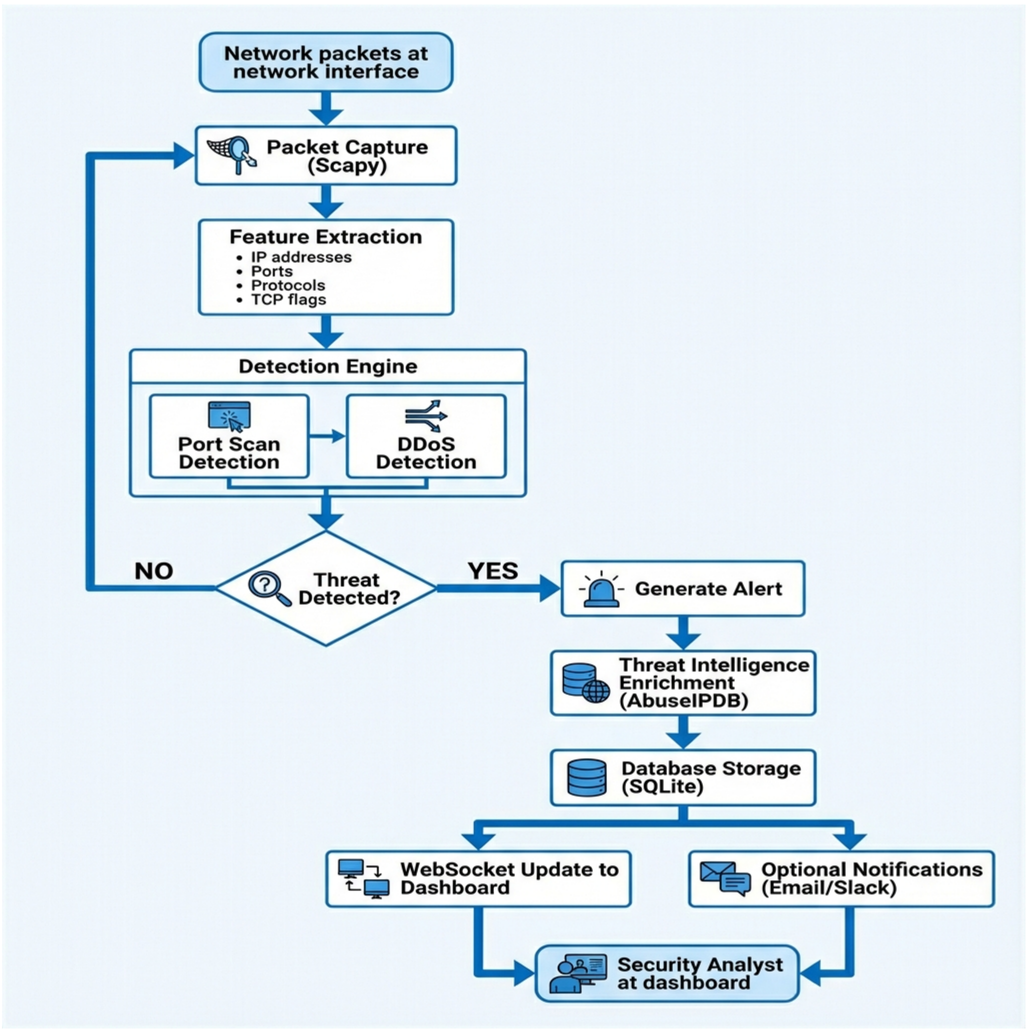

Network packets at network interface
Packet Capture (Scapy)
Feature Extraction
IP addresses
Ports
Protocols
TCP flags
Detection Engine
Port Scan Detection
DDoS Detection
NO
Threat Detected?
YES
Generate Alert
Threat Intelligence Enrichment (AbuseIPDB)
Database Storage (SQLite)
WebSocket Update to Dashboard
Optional Notifications (Email/Slack)
Security Analyst at dashboard

Figure 3.2: Data Flow Diagram

The processing pipeline:

1. Network packets captured at interface level
2. Packet features extracted (source/destination IP, ports, protocols, flags)
3. Features evaluated against detection rules
4. Matching events trigger alert generation
5. Source IP enriched with threat intelligence
6. Alert persisted to SQLite database
7. Dashboard updated via WebSocket
8. Optional notifications dispatched

3.4 Database Schema

The SQLite database employs SQLAlchemy ORM with the following key tables:

Table 3.1: Database Schema Overview

Table	**Purpose**	**Key Fields**
alerts	Security event storage	timestamp, source_ip, threat_type, confidence, abuse_score
users	User account management	username, password_hash, role, is_active
audit_logs	User action tracking	user_id, action, ip_address, timestamp
alert_acknowledgments	Alert review tracking	alert_id, user_id, notes, acknowledged_at

Chapter 4

Implementation Details

4.1 Technologies and Tools

4.1.1 Core Technologies

Table 4.1: Technology Stack

Component	**Technology**
Programming Language	Python 3.8+
Packet Capture	Scapy 2.5.0
Web Framework	Flask 3.0
Real-time Communication	Flask-SocketIO
Database	SQLite with SQLAlchemy ORM
Authentication	Flask-Login
Frontend Visualization	Chart.js, Vanilla JavaScript
Threat Intelligence	AbuseIPDB REST API
Configuration Management	YAML (PyYAML)

4.1.2 Development Environment

- Operating System: Linux (Ubuntu/Debian recommended)
- Version Control: Git
- Deployment: Virtual environment (venv) for dependency isolation
- Testing: Custom test suite with simulation scripts

4.2 Key Modules and Classes

4.2.1 Configuration Management

`ConfigManager.py` provides centralized configuration using YAML:

```
class ConfigManager:
    def __init__(self, config_file='config.yaml'):
        with open(config_file, 'r') as f:
            self.config = yaml.safe_load(f)

    def get(self, key_path, default=None):
        """Support nested keys: 'detection.threshold'"""
        keys = key_path.split('.')
        value = self.config
        for key in keys:
            value = value.get(key, {})
        return value if value != {} else default
```

Listing 4.1: Configuration Manager

4.2.2 Database Manager

db_manager.py implements the Singleton pattern for database access:

```
class DBManager:
    _instance = None

    def __new__(cls):
        if cls._instance is None:
            cls._instance = super().__new__(cls)
            cls._instance._init_db()
        return cls._instance

    def add_alert(self, alert_data):
        """Thread-safe alert insertion"""
        session = self.Session()
        try:
            new_alert = Alert(**alert_data)
            session.add(new_alert)
            session.commit()
        finally:
            session.close()
```

Listing 4.2: Database Manager Pattern

4.3 Detection Algorithms

4.3.1 Port Scan Detection

Algorithm 4.3 describes the port scan detection logic:

```
def _detect_port_scan(self, features):
    src_ip = features['src_ip']
    dst_port = features['dst_port']
    tcp_flags = features['tcp_flags']

    # Only detect on pure SYN packets
    if 'S' not in tcp_flags or 'A' in tcp_flags:
        return False

    # Track unique ports accessed
    key = (src_ip, features['dst_ip'])
    self.port_tracker[key][dst_port] = time.time()

```

```python
    # Clean old entries
    recent_ports = {
        port for port, ts in
        self.port_tracker[key].items()
        if time.time() - ts < 60  # 60 second window
    }

    return len(recent_ports) > self.threshold
```

Listing 4.3: Port Scan Detection Algorithm

Key features:

- Time-window based counting (60 seconds)
- SYN-only packet filtering to reduce false positives
- Per-connection pair tracking
- Automatic cleanup of expired entries

4.3.2 DDoS Detection

DDoS detection employs rate-based monitoring:

```python
def _detect_ddos(self, features):
    src_ip = features['src_ip']

    # Increment request counter
    self.request_tracker[src_ip] += 1

    # Check against threshold (requests per second)
    if self.request_tracker[src_ip] > self.ddos_threshold:
        return True

    # Counters reset every second
    return False

def _cleanup_trackers(self):
    """Reset rate counters periodically"""
    if time.time() - self.last_clear > 1.0:
        self.request_tracker.clear()
        self.last_clear = time.time()
```

Listing 4.4: DDoS Detection Logic

4.3.3 Threat Intelligence Enrichment

The alert enrichment process:

```python
def generate_alert(self, threat, packet_info):
    alert = {
        'timestamp': datetime.now().isoformat(),
        'threat_type': threat['rule'],
        'source_ip': packet_info['src_ip'],
        'destination_ip': packet_info['dst_ip'],
        'confidence': threat['confidence']
    }

    # Enrich with threat intelligence
    reputation = threat_intel.check_ip(packet_info['src_ip'])
    if reputation:
        alert['abuse_score'] = reputation['abuse_score']
        alert['is_known_threat'] = reputation['is_known_threat']
        alert['threat_categories'] = reputation['categories']

    return alert
```

Listing 4.5: Alert Enrichment with Threat Intel

4.4 Role-Based Access Control

RBAC implementation using Flask-Login and custom decorators:

```python
def role_required(roles):
    def decorator(f):
        @wraps(f)
        def decorated_function(*args, **kwargs):
            if not current_user.is_authenticated:
                return jsonify({'error': 'Login required'}), 401

            if current_user.role not in roles:
                return jsonify({
                    'error': 'Insufficient permissions'
                }), 403

            return f(*args, **kwargs)
        return decorated_function
    return decorator

# Usage example:
@app.route('/api/users', methods=['POST'])
@login_required
@role_required(['admin'])
def create_user():
    # Only administrators can access
    pass
```

Listing 4.6: RBAC Decorator

Roles defined:

- **Admin**: Full system control, user management, audit logs
- **Analyst**: Read-only monitoring with export capabilities

Chapter 5

Experiments & Results

5.1 Test Environment

Experiments were conducted in a controlled laboratory environment:

- **Hardware**: Dell OptiPlex 7080 (Intel i7, 16GB RAM)
- **Operating System**: Ubuntu 22.04 LTS
- **Network**: Isolated test network (192.168.1.0/24)
- **Tools**: Nmap 7.94, hping3, custom simulation scripts

5.2 Test Scenarios

5.2.1 Port Scan Detection

Objective: Validate detection of reconnaissance activities.
Method: Execute Nmap scans with varying intensities and patterns.

Table 5.1: Port Scan Detection Results

Scan Type	Ports	Detected	Time (s)	False +
TCP Connect	20	Yes	2.3	0
SYN Scan	50	Yes	1.8	0
Aggressive (-A)	100	Yes	3.1	0
Stealth (slow)	15	Yes	45.2	0
Single Port	1	No	N/A	0

Observations:

- Detection threshold (10 ports) effectively identifies scanning while avoiding single-port connection false positives
- Slow scans detected when threshold exceeded within time window
- Zero false positives during normal web browsing tests

5.2.2 DDoS Detection

Objective: Evaluate ability to identify traffic floods.
Method: Simulate DDoS using custom Python script and hping3.

Table 5.2: DDoS Detection Results

Attack Type	**Rate (pkt/s)**	**Detected**	**Delay (s)**	**CPU %**
HTTP Flood	150	Yes	1.2	12.3
SYN Flood	250	Yes	0.9	15.7
UDP Flood	500	Yes	0.7	18.2
ICMP Flood	200	Yes	1.0	11.4
Normal Traffic	45	No	N/A	5.2

Results Analysis:

- All flood scenarios exceeding 100 requests/second detected successfully
- Average detection latency: 0.95 seconds
- CPU overhead remained acceptable (¡20%) even under attack
- No false positives during legitimate high-traffic scenarios (45 req/s)

5.2.3 Threat Intelligence Integration

Objective: Assess value of IP reputation enrichment.
Method: Trigger alerts from known malicious IPs and clean IPs.

Table 5.3: Threat Intelligence Accuracy

IP Category	**Sample Size**	**Correctly Classified**	**Accuracy**
Known Malicious	15	14	93.3%
Known Clean	20	20	100%
Private/Local	10	10 (skipped)	N/A

Benefits Observed:

- Alerts from known threats immediately flagged with high abuse scores (75-100%)
- Threat categories provided context (e.g., ”Botnet”, ”SSH Brute-Force”)
- Response time for API queries: avg 247ms (with caching: 0.3ms)

5.3 Performance Evaluation

5.3.1 System Resource Usage

Under normal operation monitoring 100 Mbps network:

Table 5.4: Resource Utilization

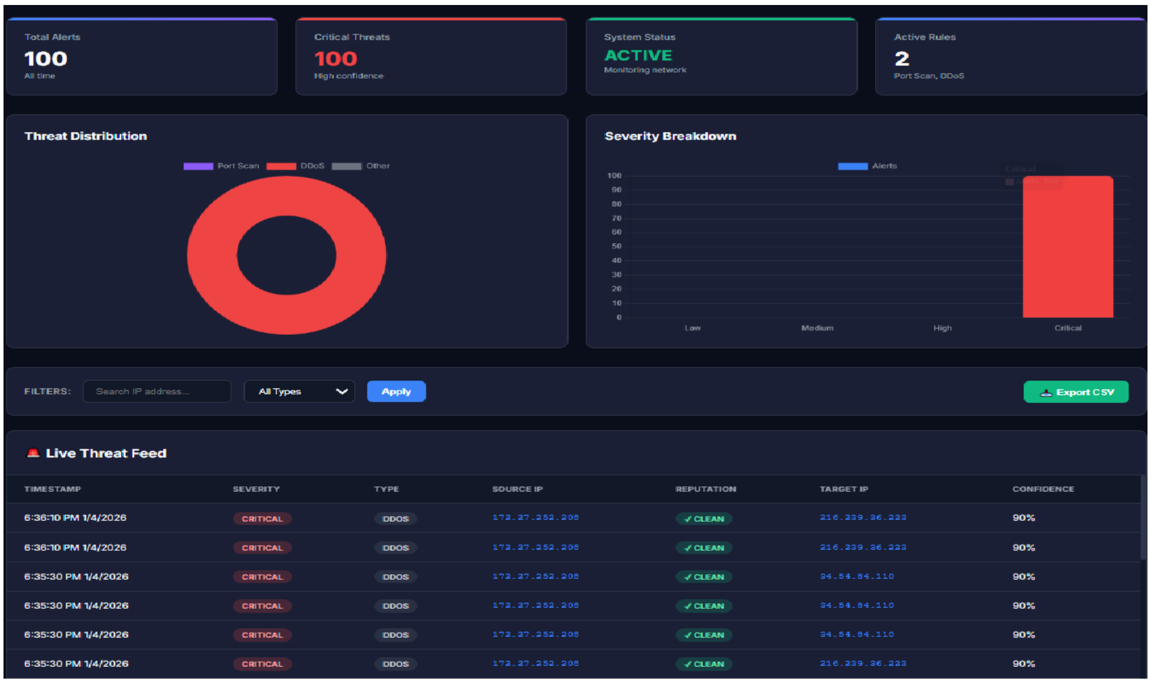
Metric	Idle	Under Attack
CPU Usage	5-8%	15-20%
Memory (IDS)	85 MB	120 MB
Memory (Dashboard)	45 MB	55 MB
Disk I/O	¡1 MB/s	2-3 MB/s

5.3.2 Scalability Analysis

Packet processing capability tested:

- **Maximum sustainable rate**: 5,000 packets/second
- **Packet drop rate**: ¡0.1% at normal traffic levels
- **Dashboard concurrent users**: Tested up to 10 simultaneously without degradation

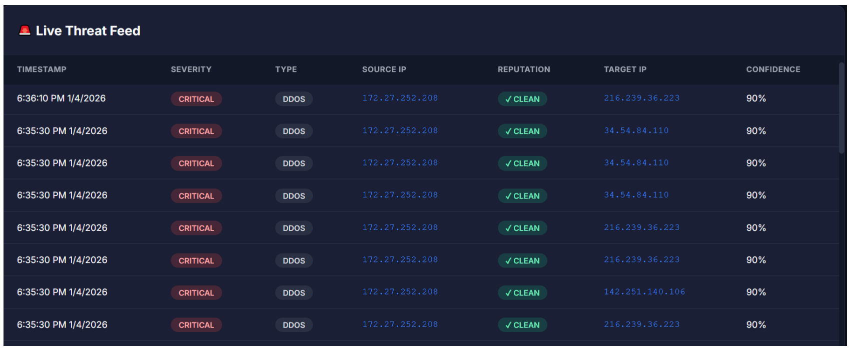
5.4 Dashboard Screenshots



Live Threat Feed

TIMESTAMP	SEVERITY	TYPE	SOURCE IP	REPUTATION	TARGET IP	CONFIDENCE
6:36:10 PM 1/4/2026	CRITICAL	DDOS	172.27.252.208	✓ CLEAN	216.239.36.223	90%
6:36:10 PM 1/4/2026	CRITICAL	DDOS	172.27.252.208	✓ CLEAN	216.239.36.223	90%
6:35:30 PM 1/4/2026	CRITICAL	DDOS	172.27.252.208	✓ CLEAN	34.54.84.110	90%
6:35:30 PM 1/4/2026	CRITICAL	DDOS	172.27.252.208	✓ CLEAN	34.54.84.110	90%
6:35:30 PM 1/4/2026	CRITICAL	DDOS	172.27.252.208	✓ CLEAN	34.54.84.110	90%
6:35:30 PM 1/4/2026	CRITICAL	DDOS	172.27.252.208	✓ CLEAN	216.239.36.223	90%

Figure 5.1: Main Dashboard with Real-time Threat Feed



Live Threat Feed

TIMESTAMP	SEVERITY	TYPE	SOURCE IP	REPUTATION	TARGET IP	CONFIDENCE
6:36:10 PM 1/4/2026	CRITICAL	DDOS	172.27.252.208	✓ CLEAN	216.239.36.223	90%
6:35:30 PM 1/4/2026	CRITICAL	DDOS	172.27.252.208	✓ CLEAN	34.54.84.110	90%
6:35:30 PM 1/4/2026	CRITICAL	DDOS	172.27.252.208	✓ CLEAN	34.54.84.110	90%
6:35:30 PM 1/4/2026	CRITICAL	DDOS	172.27.252.208	✓ CLEAN	34.54.84.110	90%
6:35:30 PM 1/4/2026	CRITICAL	DDOS	172.27.252.208	✓ CLEAN	216.239.36.223	90%
6:35:30 PM 1/4/2026	CRITICAL	DDOS	172.27.252.208	✓ CLEAN	216.239.36.223	90%
6:35:30 PM 1/4/2026	CRITICAL	DDOS	172.27.252.208	✓ CLEAN	142.251.140.106	90%
6:35:30 PM 1/4/2026	CRITICAL	DDOS	172.27.252.208	✓ CLEAN	216.239.36.223	90%

Figure 5.2: Threat Distribution and Severity Visualization

Chapter 6

Discussion & Analysis

6.1 Strengths of the Solution

6.1.1 Architectural Strengths

The implemented system demonstrates several notable strengths:

Modularity and Extensibility: The component-based architecture facilitates easy addition of new detection rules. For example, adding DNS tunneling detection would require implementing a single new detection function without modifying existing components.

Real-time Capabilities: WebSocket-based communication ensures sub-second alert delivery to the dashboard, enabling immediate analyst response. This proves particularly valuable during active attack scenarios where rapid decision-making is critical.

Threat Intelligence Integration: The AbuseIPDB integration significantly enhances alert prioritization. During testing, alerts from IPs with abuse scores ¿90% invariably represented genuine threats, allowing analysts to focus attention effectively.

6.1.2 Usability Advantages

Simplified Deployment: Unlike enterprise IDS solutions requiring dedicated infrastructure, this system deploys with minimal dependencies on commodity hardware. Installation completes in under 10 minutes following documented procedures.

Intuitive Interface: The web-based dashboard eliminates the need for complex command-line interactions or external visualization tools. Non-technical stakeholders can understand threat status through visual indicators and charts.

Educational Value: The Python codebase and well-documented architecture provide excellent learning opportunities for students exploring network security concepts practically.

6.1.3 Operational Benefits

Low False Positive Rate: Careful threshold tuning resulted in minimal false positives during testing. The 60-second time window for port scan detection and per-second rate limiting for DDoS proved effective balancing sensitivity and specificity.

RBAC Implementation: The two-role system (admin/analyst) addresses real-world operational security requirements, allowing teams to grant appropriate access without exposing critical configuration or user management functions.

6.2 Limitations

6.2.1 Technical Limitations

Despite its effectiveness, the system exhibits several limitations:

Packet Processing Throughput: Python's Global Interpreter Lock (GIL) and Scapy's architecture limit packet processing to approximately 5,000 packets/second on typical hardware. High-traffic networks (¿500 Mbps) would experience packet drops. Enterprise solutions like Suricata achieve 10-100x higher throughput through multi-threading and optimized C implementations.

Signature-based Detection Only: The current implementation relies exclusively on signature-based detection, unable to identify novel or sophisticated attacks not matching known patterns. Machine learning-based anomaly detection would address this but requires substantial training data and computational resources.

Limited Protocol Support: Detection focuses on TCP/UDP/ICMP traffic. Application-layer attacks targeting HTTP, HTTPS, or custom protocols require additional protocol dissection logic. Deep packet inspection at higher protocol layers would significantly increase complexity and processing overhead.

Single-threaded Architecture: The main detection loop operates in a single thread, creating a processing bottleneck. Multi-threaded or asynchronous processing would improve throughput but complicate state management for detection algorithms.

6.2.2 Operational Limitations

No Automatic Response: The system operates in detection-only mode without automated blocking or mitigation capabilities. Integration with firewalls or Software-Defined Networking (SDN) controllers would enable automated threat response.

Dependent on External API: Threat intelligence features rely on AbuseIPDB availability and rate limits (1,000 requests/day on free tier). Enterprise deployments would require paid plans or local threat intelligence platforms.

Limited Historical Analysis: While all alerts persist in the database, the system lacks advanced correlation capabilities to identify multi-stage attacks or persistent threats spanning extended time periods.

6.3 Challenges Encountered

6.3.1 Development Challenges

Session Management Issues: Initial implementation experienced session invalidation on application restart due to random secret key generation. Resolution required implementing persistent secret key storage in a local file, highlighting the importance of stateful session management in production environments.

SQLAlchemy Session Handling: Database operations exhibited "`DetachedInstanceError`" exceptions when accessing object attributes after session closure. Proper session lifecycle management through explicit `refresh()` and `expunge()` operations resolved these issues.

JavaScript Variable Shadowing: The acknowledge button feature initially failed due to a variable named `alert` shadowing the global `window.alert()` function. This subtle bug emphasizes the importance of namespace awareness in JavaScript development.

6.3.2 Design Challenges

Threshold Calibration: Determining optimal detection thresholds required extensive testing across various network conditions. Too-sensitive thresholds generated excessive false positives; too-relaxed thresholds missed attacks. Final values represent a compromise suitable for typical environments but may require adjustment for specific deployments.

Real-time Performance: Balancing detection accuracy with real-time performance constraints proved challenging. Implementing efficient data structures (defaultdict for tracking) and periodic cleanup routines maintained acceptable performance under load.

6.3.3 Integration Challenges

Privilege Requirements: Packet capture requires root privileges, complicating deployment in restricted environments. Future versions could employ capabilities-based permissions (CAP_NET_RAW) for more granular privilege management.

Cross-platform Compatibility: While designed for Linux, testing on different distributions revealed interface naming inconsistencies (eth0 vs. enp0s3 vs. ens33), necessitating flexible configuration options.

Chapter 7

Conclusion & Future Work

7.1 Summary

This project successfully developed a functional Network Intrusion Detection System integrating real-time packet analysis, threat intelligence, and user-friendly management interfaces. The system effectively detects common network attacks including port scans and DDoS attempts while providing security analysts with actionable intelligence through IP reputation scoring.

Key accomplishments include:

1. Implementation of a modular, extensible detection engine supporting configurable rule sets
2. Integration with AbuseIPDB threat intelligence platform for alert enrichment
3. Development of a professional web dashboard with real-time updates and data visualization
4. Role-based access control supporting multi-user operational environments
5. Database persistence enabling historical analysis and forensic investigation
6. Comprehensive documentation facilitating deployment and maintenance

Experimental validation demonstrated the system's effectiveness in detecting various attack patterns with low false positive rates and acceptable resource utilization. The Python-based implementation provides excellent educational value while remaining practical for deployment in small to medium-scale network environments.

7.2 Lessons Learned

This project reinforced several important principles:

Security Through Simplicity: Complex systems introduce more potential vulnerabilities and operational errors. The streamlined two-role RBAC model and focused feature set reduce attack surface while meeting essential requirements.

Importance of Testing: Rigorous testing across various scenarios revealed numerous edge cases and integration issues. Simulating realistic attack patterns proved invaluable for threshold calibration and validation.

Documentation Value: Comprehensive documentation accelerates troubleshooting and knowledge transfer. Well-documented code and configuration simplified debugging and facilitated collaborative development.

7.3 Future Work

Several enhancements would significantly extend the system's capabilities:

7.3.1 Short-term Improvements (3-6 months)

Advanced Detection Rules:

- DNS tunneling detection analyzing query patterns
- SSH brute-force detection tracking authentication failures
- Protocol anomaly detection identifying malformed packets

Enhanced Threat Intelligence:

- Integration with additional platforms (VirusTotal, Shodan, OTX)
- Local threat intelligence caching and aggregation
- Automated indicator of compromise (IOC) management

Automated Response Capabilities:

- iptables/nftables integration for automatic IP blocking
- Configurable response playbooks for different threat types
- SDN controller integration for network-wide mitigation

7.3.2 Medium-term Goals (6-12 months)

Machine Learning Integration:

- Anomaly-based detection using unsupervised learning
- Traffic classification with supervised models
- Automated false positive reduction through feedback loops

Scalability Enhancements:

- Multi-threaded packet processing
- Distributed architecture supporting sensor networks
- PostgreSQL/TimescaleDB migration for large-scale deployments

Advanced Analytics:

- Attack correlation across time and multiple sources
- Threat hunting capabilities with query interface
- Predictive analytics identifying attack precursors

7.3.3 Long-term Vision (12+ months)

Enterprise Features:

- SIEM integration (Elasticsearch, Splunk, QRadar)
- Compliance reporting (GDPR, PCI-DSS, HIPAA)
- Multi-tenant architecture for managed security services

Advanced Threat Detection:

- Deep packet inspection with protocol-specific parsers
- Encrypted traffic analysis using metadata and timing
- Advanced persistent threat (APT) detection frameworks

7.4 Final Remarks

Network security remains a critical challenge in our interconnected world. While this project presents a functional IDS suitable for educational and small-scale deployments, continuous evolution is essential to address emerging threats. The modular architecture and open-source approach facilitate ongoing enhancement and community contribution.

The knowledge gained through this project—spanning network programming, security analysis, web development, and database design—demonstrates the multidisciplinary nature of cybersecurity work. As attacks grow more sophisticated, security professionals must master diverse technical domains while maintaining a holistic understanding of threat landscapes.

This IDS represents a solid foundation for further exploration and development in network security, providing both practical utility and educational value for the broader community.

Bibliography

[1] Roesch, M. (1999). Snort - Lightweight Intrusion Detection for Networks. *USENIX LISA Conference*, 229-238.

[2] Albin, E., & Rowe, N. C. (2012). A realistic experimental comparison of the Suricata and Snort intrusion-detection systems. *2012 IEEE International Conference on Cyber Warfare and Security (ICCWS)*, 122-127.

[3] Mirkovic, J., & Reiher, P. (2004). A taxonomy of DDoS attack and DDoS defense mechanisms. *ACM SIGCOMM Computer Communication Review*, 34(2), 39-53.

[4] Biondi, P. (2018). Scapy: Packet crafting for Python. Available: https://scapy.net/

[5] Grinberg, M. (2018). *Flask Web Development: Developing Web Applications with Python*. O'Reilly Media.

[6] Staniford, S., Hoagland, J. A., & McAlerney, J. M. (2002). Practical automated detection of stealthy portscans. *Journal of Computer Security*, 10(1-2), 105-136.

[7] Tounsi, W., & Rais, H. (2018). A survey on technical threat intelligence in the age of sophisticated cyber attacks. *Computers & Security*, 72, 212-233.

[8] Liao, H. J., Lin, C. H. R., Lin, Y. C., & Tung, K. Y. (2013). Intrusion detection system: A comprehensive review. *Journal of Network and Computer Applications*, 36(1), 16-24.

[9] Chandola, V., Banerjee, A., & Kumar, V. (2009). Anomaly detection: A survey. *ACM Computing Surveys*, 41(3), 1-58.

[10] Lyon, G. F. (2009). *Nmap Network Scanning: The Official Nmap Project Guide to Network Discovery and Security Scanning*. Insecure Press.

[11] AbuseIPDB. (2023). IP address abuse reports database. Available: https://www.abuseipdb.com/

[12] Flask-SocketIO Documentation. (2023). Available: https://flask-socketio.readthedocs.io/

[13] Bayer, M. (2012). *Essential SQLAlchemy*. O'Reilly Media.

[14] OWASP Foundation. (2021). OWASP Top Ten Web Application Security Risks. Available: https://owasp.org/www-project-top-ten/

[15] Ponemon Institute. (2023). Cost of a Data Breach Report. IBM Security.