Network Traffic Analyzer - Setup and Usage Guide

Overview

The Network Traffic Analyzer is a comprehensive distributed security monitoring system that captures, analyzes, and detects suspicious network traffic in real-time. The system provides centralized logging, alerting, and visualization capabilities for network security monitoring.

System Requirements

Software Requirements

- Python 3.8 or higher
- PostgreSQL 12 or higher
- Root/Administrator privileges (for packet capture)
- Modern web browser (for dashboard)

Hardware Requirements

- Minimum 4GB RAM
- 50GB+ disk space (depending on traffic volume)
- Network interface access
- Multiple cores recommended for high-traffic environments

Installation

1. Install System Dependencies

Ubuntu/Debian:

```
sudo apt update
sudo apt install python3 python3-pip postgresql postgresql-contrib
sudo apt install libpcap-dev python3-dev  # For Scapy
```

CentOS/RHEL:

```
sudo yum install python3 python3-pip postgresql postgresql-server
sudo yum install libpcap-devel python3-devel
```

Windows:

- Install Python 3.8+ from python.org
- Install PostgreSQL from postgresql.org
- Install Npcap from nmap.org/npcap (for packet capture)

2. Setup PostgreSQL Database

```
# Start PostgreSQL service
sudo systemctl start postgresql
sudo systemctl enable postgresql

# Create database and user
sudo -u postgres psql

CREATE DATABASE network_analyzer;
CREATE USER analyzer_user WITH PASSWORD 'secure_password';
GRANT ALL PRIVILEGES ON DATABASE network_analyzer TO analyzer_user;
\q
```

3. Initialize Database Schema

```
# Connect to database
psql -h localhost -U analyzer_user -d network_analyzer

# Run the SQL from database_setup.sql
\i database_setup.sql
\q
```

4. Install Python Dependencies

```
# Create virtual environment (recommended)
python3 -m venv network_analyzer_env
source network_analyzer_env/bin/activate  # Linux/Mac
# or
network_analyzer_env\Scripts\activate  # Windows

# Install dependencies
pip install -r requirements.txt
```

Configuration

Database Configuration

Update database connection settings in your startup commands or create a configuration file:

```
export DB_HOST=localhost
export DB_PORT=5432
export DB_NAME=network_analyzer
export DB_USER=analyzer_user
export DB_PASSWORD=secure_password
```

Network Interface

Find available network interfaces:

```
# Linux
ip link show
# or
sudo python3 -c "from scapy.all import get_if_list; print(get_if_list())"
```

Usage

1. Complete System (Recommended)

Start both packet capture and API server:

```
sudo python3 main.py --enable-capture --enable-api \
  --db-password secure_password \
  --interface eth0 \
  --node-id main-node \
  --enable-cleanup
```

2. Distributed Setup

Central API Server:

```
python3 main.py --enable-api \
  --api-host 0.0.0.0 \
  --api-port 5000 \
  --db-password secure_password \
  --enable-cleanup
```

Remote Capture Nodes:

```
sudo python3 main.py --enable-capture \
  --interface eth0 \
  --node-id capture-node-1 \
  --db-host 192.168.1.100 \
  --db-password secure_password
```

3. Individual Components

Packet Capture Only:

```
sudo python3 packet_capture.py \
  --interface eth0 \
  --node-id test-node \
  --db-password secure_password
```

API Server Only:

```
python3 api_server.py \
  --host 0.0.0.0 \
  --port 5000 \
  --db-password secure_password
```

Dashboard Access

Once the API server is running, access the web dashboard:

- Open browser to `http://localhost:5000` (or your server IP)
- The dashboard provides real-time monitoring of:
  - Network traffic statistics
  - Security alerts
  - Top source IPs
  - Protocol distribution
  - Recent packets

API Endpoints

Packets

- `GET /api/packets` - Retrieve packets with filters
- `POST /api/packets/search` - Advanced packet search

Alerts

- `GET /api/alerts` - Retrieve security alerts
- `POST /api/alerts/{id}/resolve` - Mark alert as resolved

Statistics

- `GET /api/stats` - Traffic and security statistics
- `GET /api/nodes` - Active capture nodes information

System

- `GET /api/health` - System health check

Example API Usage

```
# Get recent packets
curl "http://localhost:5000/api/packets?limit=10"

# Get alerts from specific IP
curl "http://localhost:5000/api/alerts?source_ip=192.168.1.100"

# Get traffic statistics for last 6 hours
curl "http://localhost:5000/api/stats?hours=6"

# Advanced packet search
curl -X POST http://localhost:5000/api/packets/search \
  -H "Content-Type: application/json" \
  -d '{"ip_range": "192.168.1.0/24", "protocols": ["TCP", "UDP"]}'
```

Attack Detection

The system automatically detects and alerts on:

Port Scanning

- **Threshold:** 10+ unique ports from single IP within 60 seconds
- **Severity:** Medium (3)

SYN Flood Attacks

- **Threshold:** 100+ SYN packets from single IP within 60 seconds
- **Severity:** High (4)

Brute Force Attacks

- **Threshold:** 20+ connection attempts to same port within 60 seconds
- **Severity:** Medium (3)
- **Monitored Ports:** SSH (22), Telnet (23), FTP (21), SMTP (25), POP3 (110), IMAP (143), RDP (3389)

ARP Spoofing

- **Detection:** Inconsistent IP-to-MAC address mappings
- **Severity:** Low (2)

Performance Tuning

High Traffic Environments

```
# Increase database connections
--db-max-conn 50

# Use packet filtering to reduce load
--packet-filter "not port 53 and not port 123"

# Adjust cleanup intervals
--cleanup-interval 12 --data-retention-days 7
```

Memory Optimization

- Monitor system memory usage
- Adjust PostgreSQL shared_buffers and work_mem
- Consider packet sampling for very high traffic

Storage Management

```
# Monitor database size
du -sh /var/lib/postgresql/

# Manual cleanup of old data
python3 -c "
from database import DatabaseManager
db = DatabaseManager(password='your_password')
db.cleanup_old_data(days_to_keep=7)
"
```

Security Considerations

Network Access

- Run API server on internal network only
- Use reverse proxy (nginx/Apache) for external access
- Enable HTTPS in production

Database Security

- Use strong passwords

- Enable SSL connections
- Restrict database access by IP
- Regular backups

System Permissions

- Packet capture requires root privileges
- Consider using capabilities instead of full root
- Run API server as non-privileged user

Troubleshooting

Common Issues

Permission Denied (Packet Capture):

```
# Run with sudo or set capabilities
sudo setcap cap_net_raw,cap_net_admin=eip /usr/bin/python3
```

Database Connection Failed:

- Verify PostgreSQL is running
- Check connection parameters
- Ensure database and user exist
- Check PostgreSQL pg_hba.conf for authentication

No Packets Captured:

- Verify network interface name
- Check interface is up and has traffic
- Test with tcpdump first
- Verify BPF filter syntax

High Memory Usage:

- Reduce packet retention period
- Implement packet sampling
- Increase cleanup frequency
- Monitor database query performance

Debug Mode

```
# Enable verbose logging
python3 main.py --enable-capture --enable-api --log-level DEBUG

# Check API connectivity
curl http://localhost:5000/api/health
```

Log Files

- System logs: `network_analyzer.log`
- PostgreSQL logs: `/var/log/postgresql/`
- API access logs: Flask console output

Monitoring and Maintenance

Regular Tasks

1. Monitor disk space usage
2. Review security alerts
3. Update attack detection thresholds
4. Backup database regularly
5. Update software dependencies

Health Checks

```
# System status
curl http://localhost:5000/api/health

# Database connectivity
psql -h localhost -U analyzer_user -d network_analyzer -c "SELECT COUNT(*) FROM packets;"

# Recent activity
curl "http://localhost:5000/api/stats?hours=1"
```

Integration

SIEM Integration

The REST API allows integration with SIEM systems:

- Export alerts in JSON format
- Real-time alert streaming
- Custom alert webhooks (can be implemented)

Custom Dashboards

- Grafana integration via API
- Custom visualization tools
- Mobile applications

Automated Response

- Script-based alert handling
- Firewall rule automation
- Incident response workflows

Support and Development

Contributing

- Follow Python PEP 8 style guidelines
- Add tests for new features
- Update documentation

Extending the System

- Add new attack detection algorithms
- Implement additional protocols
- Create custom visualization components
- Add machine learning capabilities

For questions and issues, refer to the system logs and API documentation.