Network Intrusion Detection System (NIDS) Documentation

Table of Contents

System Requirements

Hardware Requirements

- **Minimum**: 2GB RAM, Dual-core processor
- **Recommended**: 4GB+ RAM, Quad-core processor
- **Network**: Access to network interface (Wi-Fi or Ethernet)

Software Requirements

- **Operating System**: Windows 10/11, Linux, macOS
- **Python**: Version 3.7 or higher
- **Administrator/Root privileges** for packet capture

Python Dependencies

```bash
pip install scapy pandas matplotlib seaborn numpy plotly
# Optional for AI features:
pip install openai
```

Installation Guide

Step 1: Install Python

Download and install Python from python.org if not already installed.

Step 2: Install Required Libraries

Windows

```cmd
# Open Command Prompt as Administrator
pip install scapy pandas matplotlib seaborn numpy plotly

# For Windows, you may also need:
pip install pywin32
```

Linux/macOS

```bash
# Open Terminal
sudo pip3 install scapy pandas matplotlib seaborn numpy plotly

# On Linux, you may need:
sudo apt-get install python3-tk tcpdump
```

Step 3: Verify Installation

```python
python -c "import scapy.all; print('Scapy installed successfully')"
```

Step 4: Download the NIDS Script

Save the main Python script as `nids.py` in your working directory.

Configuration

Basic Configuration

Edit the following variables in the script:

```python
# In main() function:
interface = None  # Auto-detect, or specify: 'Wi-Fi', 'eth0', 'en0'
duration = 60     # Capture duration in seconds

# Detection thresholds (in NetworkIDS.__init__):
self.thresholds = {
    'syn_flood_threshold': 100,       # SYN packets threshold
    'port_scan_threshold': 20,        # Different ports threshold
    'packet_rate_threshold': 1000,    # Packets/second threshold
    'failed_connection_threshold': 10, # Failed connections threshold
    'unusual_port_threshold': 5       # Unusual port connections
}
```

Finding Your Network Interface

Windows

```cmd
# List network interfaces
ipconfig /all
# Look for "Wireless LAN adapter Wi-Fi" or "Ethernet adapter"
```

For Windows with Python:

```python
from scapy.all import get_if_list, get_if_hwaddr
for iface in get_if_list():
    print(f"{iface}: {get_if_hwaddr(iface)}")
```

Common Windows interface names:

- `"Ethernet"` - Wired connection
- `"Wi-Fi"` - Wireless connection
- `"Ethernet 2"`, `"Ethernet 3"` - Multiple adapters
- Use the exact name shown in ipconfig

Linux

```bash
# List network interfaces
ip link show
# or
ifconfig -a
# Common: eth0, wlan0, ens33
```

macOS

```bash
# List network interfaces
ifconfig
# Common: en0 (Wi-Fi), en1 (Ethernet)
```

Usage Instructions

Basic Usage

1. Run with Default Settings

```bash
# Windows (Run as Administrator)
python nids.py

# Linux/macOS (Run as root)
sudo python3 nids.py
```

2. Specify Network Interface

```python
# Modify in script:
interface = "Wi-Fi"  # Windows
interface = "wlan0"  # Linux
interface = "en0"    # macOS
```

3. Analyze PCAP File (Offline Analysis)

```python
```

```
# Modify in script:
nids = NetworkIDS(pcap_file="capture.pcap")
```

Real-time Monitoring

The system displays real-time statistics:

```
Packets: 1234 | Alerts: 5 | TCP: 890 | UDP: 344
```

Generated Output Files

After running, the system generates:

1. **nids.log** - Detailed system logs
2. **security_report.md** - Markdown security report
3. **network_analysis.html** - Interactive traffic dashboard
4. **interaction_heatmap.html** - Network interaction visualization

---

Testing Guide

Test Environment Setup

1. Create a Safe Testing Environment

Option A: Virtual Network

```
bash

# Using VirtualBox or VMware
# Create 2-3 VMs on an isolated network
# Run NIDS on host or one VM
```

Option B: Docker Network

```
bash
```

```
# Create isolated Docker network
docker network create --driver bridge testnet

# Run containers for testing
docker run -d --name web --network testnet nginx
docker run -d --name client --network testnet alpine sleep 3600
```

Test Scenarios

Test 1: Port Scanning Detection

bash

```
# From another machine or VM
# Using nmap (install if needed)
nmap -sS -p 1-1000 TARGET_IP

# Expected Alert: PORT_SCAN
```

Test 2: SYN Flood Detection

python

```
# Python script to generate SYN flood (TEST ONLY ON YOUR OWN NETWORK)
from scapy.all import *
import random

target_ip = "192.168.1.100"  # Your test target
for i in range(200):
    sport = random.randint(1024, 65535)
    packet = IP(dst=target_ip)/TCP(sport=sport, dport=80, flags="S")
    send(packet, verbose=0)

# Expected Alert: SYN_FLOOD
```

Test 3: Suspicious Port Connection

bash

```
# Connect to a suspicious port
telnet TARGET_IP 23  # Telnet
telnet TARGET_IP 445 # SMB

# Expected Alert: SUSPICIOUS_PORT
```

Test 4: High Packet Rate

bash

```
# Using hping3 (Linux)
sudo hping3 -i u10 -S -p 80 TARGET_IP

# Or using Python
from scapy.all import *
for i in range(2000):
    send(IP(dst="TARGET_IP")/ICMP(), verbose=0)

# Expected Alert: HIGH_PACKET_RATE
```

Test 5: Large Data Transfer

bash

```
# Generate large file transfer
# Server side:
nc -l 12345 > /dev/null

# Client side:
dd if=/dev/zero bs=1M count=100 | nc TARGET_IP 12345

# Expected Alert: Large_Data_Transfer (if custom rule enabled)
```

Automated Testing Script

python

```python
#!/usr/bin/env python3
"""
NIDS Testing Script - Run in a separate terminal
"""
import time
import subprocess
from scapy.all import *

def test_port_scan(target):
    print("Testing port scan detection...")
    for port in range(20, 50):
        send(IP(dst=target)/TCP(dport=port, flags="S"), verbose=0)
        time.sleep(0.1)

def test_syn_flood(target):
    print("Testing SYN flood detection...")
    for _ in range(150):
        send(IP(dst=target)/TCP(dport=80, flags="S"), verbose=0)

def test_suspicious_ports(target):
    print("Testing suspicious port detection...")
    suspicious = [23, 445, 3389, 5900]
    for port in suspicious:
        send(IP(dst=target)/TCP(dport=port, flags="S"), verbose=0)
        time.sleep(1)

def run_tests(target_ip):
    print(f"Starting NIDS tests against {target_ip}")
    print("WARNING: Only run on your own network!")

    test_port_scan(target_ip)
    time.sleep(5)

    test_syn_flood(target_ip)
    time.sleep(5)

    test_suspicious_ports(target_ip)

    print("Tests complete!")

if __name__ == "__main__":
    # CHANGE THIS to your test target
```

```
TARGET = "192.168.1.100"
run_tests(TARGET)
```

Understanding Alerts

Alert Types

Alert Type	Description	Severity	Indicators
SYN_FLOOD	Potential SYN flood attack	HIGH	Many SYN packets, few SYN-ACK responses
PORT_SCAN	Port scanning activity	MEDIUM	Multiple ports accessed from single IP
SUSPICIOUS_PORT	Connection to known malicious port	HIGH	Access to ports like 23, 445, 3389
HIGH_PACKET_RATE	Abnormal packet rate	MEDIUM	>1000 packets/second from single source

Reading the Security Report

```
markdown

# Example Alert in Report

### SYN_FLOOD - HIGH
**Time**: 14:23:45
**Description**: Possible SYN flood from 192.168.1.105
**Source IP**: 192.168.1.105
```

Alert Response Guidelines

1. **SYN_FLOOD**
   - Block source IP temporarily
   - Implement SYN cookies
   - Check for DDoS attack
2. **PORT_SCAN**
   - Monitor for follow-up attacks
   - Block source if persistent
   - Review firewall rules
3. **SUSPICIOUS_PORT**

- Immediately investigate connection
- Check for malware/backdoors
- Review system logs

4. **HIGH_PACKET_RATE**
   - Implement rate limiting
   - Check for data exfiltration
   - Monitor bandwidth usage

Customization

Adding Custom Detection Rules

```python
# Add to the script after rule_engine initialization
rule_engine.add_custom_rule(
    name='HTTP_Attack',
    description='Detect potential HTTP attacks',
    condition=lambda p: p.get('dst_port') == 80 and p.get('payload_size', 0) > 5000,
    threshold=5,
    window=60
)

rule_engine.add_custom_rule(
    name='ICMP_Flood',
    description='Detect ICMP flood',
    condition=lambda p: p.get('protocol') == 'ICMP',
    threshold=100,
    window=10
)
```

Modifying Suspicious Ports

```python
```

```
# In NetworkIDS.__init__, modify the suspicious_ports dictionary
self.suspicious_ports = {
    23: 'Telnet',
    135: 'RPC',
    # Add your ports:
    8080: 'HTTP Proxy',
    27017: 'MongoDB',
    6379: 'Redis'
}
```

Customizing Visualizations

```
python

# Add after the main visualize_traffic method
def create_custom_visualization(self):
    df = self.get_dataframe()

    # Time-based protocol distribution
    fig = px.area(df, x='timestamp', color='protocol',
                  title='Protocol Distribution Over Time')
    fig.write_html("protocol_timeline.html")

    # Top talkers pie chart
    top_ips = df['src_ip'].value_counts().head(10)
    fig = px.pie(values=top_ips.values, names=top_ips.index,
                 title='Top 10 Source IPs')
    fig.write_html("top_talkers.html")
```

Troubleshooting

Common Issues and Solutions

1. Permission Denied Error

```
Error: Operation not permitted
```

Solution: Run with administrator/root privileges

- Windows: Right-click → Run as Administrator
- Linux/macOS: Use `sudo`

2. No Packets Captured

Possible Causes:

- Wrong interface specified
- Firewall blocking
- No network activity

Solution:

```python
# List available interfaces
from scapy.all import get_if_list
print(get_if_list())
```

3. ImportError: No module named 'scapy'

Solution:

```bash
pip install --upgrade scapy
# or
pip3 install scapy
```

4. High CPU Usage

Solution:

- Reduce packet processing
- Implement filtering:

```python
# Only capture specific traffic
sniff(filter="tcp port 80 or tcp port 443", ...)
```

5. Memory Issues with Large Captures

Solution:

- Limit packet storage:

```python
self.packets = self.packets[-10000:]  # Keep only last 10k packets
```

Performance Optimization

For High-Traffic Networks

```python
# Add BPF filter to reduce load
sniff(filter="tcp or udp", ...)  # Skip other protocols

# Process packets in batches
def process_batch(self):
    batch = []
    while not self.packet_queue.empty():
        batch.append(self.packet_queue.get())
        if len(batch) >= 100:
            break
    # Process batch
```

For Long-Duration Monitoring

```python
# Implement data rotation
def rotate_data(self):
    if len(self.packets) > 100000:
        # Archive old data
        df = pd.DataFrame(self.packets[:50000])
        df.to_csv(f"archive_{datetime.now():%Y%m%d_%H%M%S}.csv")
        self.packets = self.packets[50000:]
```

Security Considerations

When Testing

1. **Only test on networks you own or have permission to test**
2. Use isolated test environments
3. Document all testing activities

4. Never test on production networks without approval

When Deploying

1. Secure the NIDS system itself
2. Encrypt stored data and reports
3. Implement access controls
4. Regular updates and patches
5. Monitor NIDS system resources

Legal Compliance

- Ensure compliance with local network monitoring laws
- Obtain necessary permissions
- Respect privacy regulations (GDPR, etc.)
- Maintain audit logs

Advanced Features

AI Integration (Optional)

To enable AI-powered alert summarization:

1. Get OpenAI API key from platform.openai.com
2. Install OpenAI library: `pip install openai`
3. Use in script:

```python
# After generating alerts
summary = nids.ai_summarize_alerts(api_key="your-api-key-here")
print(summary)
```

Email Notifications

```python
```

```
import smtplib
from email.mime.text import MIMEText

def send_alert_email(alert):
    msg = MIMEText(f"Alert: {alert['description']}")
    msg['Subject'] = f'NIDS Alert: {alert["type"]}'
    msg['From'] = 'nids@example.com'
    msg['To'] = 'admin@example.com'

    # Send email (configure SMTP settings)
    # smtp.send_message(msg)
```

Integration with SIEM

Export data for SIEM integration:

python

```
def export_to_siem(self):
    # Export in CEF format
    cef_logs = []
    for alert in self.alerts:
        cef = f"CEF:0|NIDS|NetworkIDS|1.0|{alert['type']}|{alert['description']}|{alert['severity']}|"
        cef_logs.append(cef)

    with open('nids_cef.log', 'w') as f:
        f.write('\n'.join(cef_logs))
```

Support and Resources

Documentation

- Scapy Documentation
- Pandas Documentation
- Plotly Documentation

Network Security Resources

- SANS Internet Storm Center
- OWASP Top 10
- CVE Database

- Snort Rules

Getting Help

1. Check the troubleshooting section
2. Review system logs (`nids.log`)
3. Verify network permissions
4. Test with simple packet capture first

Last Updated: 2024 Version: 1.0