Dokumentacja projektowa

CYBER 2025/2026

System Wykrywania Anomalii w Ruchu Sieciowym poprzez analizę portów sieciowych

Kierunek: Informatyka

Członkowie zespołu:
Sebastian Pytka
Karol Wieczorek
Paweł Chwalczyk
Adrian Pacyniak

Gliwice, 2025/2026

Spis treści

1 Wprowadzenie

1.1 Cel projektu

Celem projektu jest stworzenie zaawansowanego systemu wykrywania anomalii w ruchu sieciowym z wykorzystaniem analizy portów sieciowych oraz technik uczenia maszynowego. System ma za zadanie identyfikować potencjalne zagrożenia cyberbezpieczeństwa takie jak:

- Port scanning (skanowanie portów)
- Ataki DDoS (Distributed Denial of Service)
- SYN flood
- Nietypowe wzorce ruchu sieciowego
- Beaconing (komunikacja z serwerami C&C)
- Anomalie statystyczne w wykorzystaniu portów

Projekt łączy w sobie metody statystyczne, uczenie maszynowe, reguły heurystyczne oraz analizę temporalną, tworząc wielowarstwowy system detekcji anomalii.

1.2 Zespół projektowy

Imię i nazwisko	**Rola**	**Zadania**
Sebastian Pytka	Architekt systemu	Koordynacja, architektura
Karol Wieczorek	Developer ML	Implementacja ML detektorów
Paweł Chwalczyk	Developer Backend	Źródła danych, core logic
Adrian Pacyniak	Developer Frontend	Dashboard, wizualizacja

Tabela 1: Podział ról w zespole projektowym

2 Założenia projektowe

2.1 Wymagania funkcjonalne

System powinien umożliwiać:

1. **Analizę ruchu sieciowego** z wielu źródeł:
   - Pliki PCAP (analiza offline)
   - Przechwytywanie pakietów w czasie rzeczywistym (live capture)
   - Symulator ruchu sieciowego (do testowania)
2. **Wykrywanie anomalii** przy użyciu czterech metod:
   - Detekcja statystyczna (z-scores, analiza częstotliwości)
   - Uczenie maszynowe (Isolation Forest, One-Class SVM)
   - Reguły heurystyczne (port scan, DDoS, SYN flood)
   - Analiza temporalna (rate limiting, burst detection, beaconing)
3. **Wizualizację wyników** poprzez:
   - Interaktywny dashboard webowy
   - Wykresy w czasie rzeczywistym
   - Live security events feed
4. **Generowanie raportów** w formatach:
   - JSON (dane strukturalne)
   - CSV (analiza w arkuszach)
   - HTML (raporty z wykresami)

2.2 Wymagania niefunkcjonalne

- **Wydajność**: Przetwarzanie minimum 1000 pakietów/sekundę
- **Dokładność**: Wskaźnik false positive $< 5\%$
- **Skalowalność**: Modularna architektura umożliwiająca łatwe dodawanie nowych detektorów
- **Konfigurowalność**: Parametry detekcji konfigurowalne przez YAML
- **Bezpieczeństwo**: Brak wykonywania niebezpiecznych operacji, sandbox dla live capture

2.3 Stos technologiczny

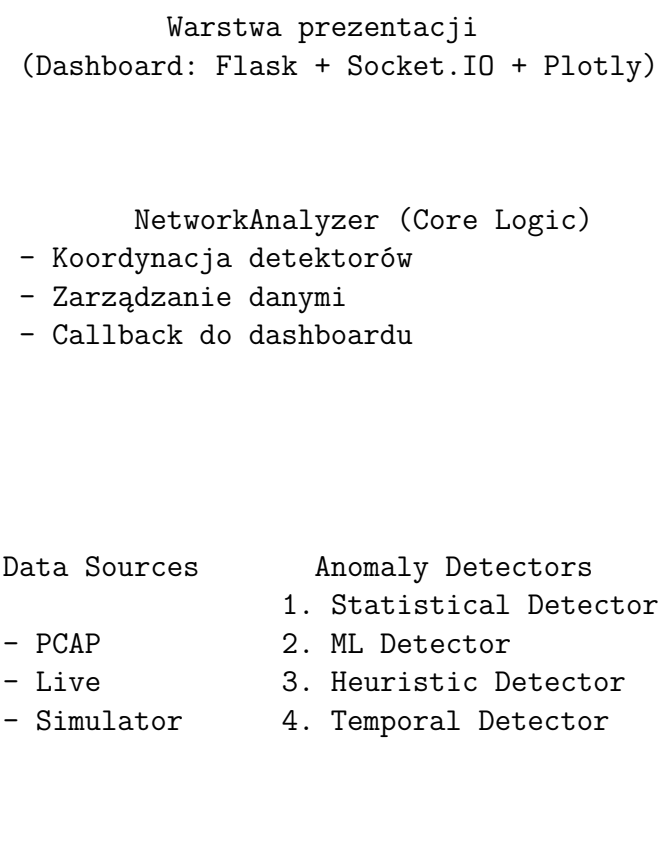
Komponent	**Technologia**
Język programowania	Python 3.14
Przechwytywanie pakietów	Scapy
Uczenie maszynowe	scikit-learn (Isolation Forest, SVM)
Framework webowy	Flask + Socket.IO
Wizualizacja	Plotly.js
Konfiguracja	PyYAML
Logging	Python logging

Tabela 2: Wykorzystane technologie

3 Realizacja projektu

3.1 Architektura systemu

System został zaprojektowany zgodnie z zasadami czystej architektury, z wyraźnym podziałem na warstwy:



Rysunek 1: Architektura systemu

3.2 Moduły systemu

3.2.1 Źródła danych (Data Sources)

PCAPReader Moduł do odczytu i analizy plików PCAP. Wykorzystuje bibliotekę Scapy do parsowania pakietów sieciowych i konwersji ich do wewnętrznego formatu `NetworkPacket`.

Listing 1: Klasa NetworkPacket

```
1 @dataclass
2 class NetworkPacket:
```

```
    timestamp: datetime
    src_ip: str
    dst_ip: str
    src_port: int
    dst_port: int
    protocol: str  # TCP, UDP, ICMP
    packet_size: int
    flags: str
```

LiveCapture Moduł do przechwytywania pakietów w czasie rzeczywistym. Wymaga uprawnień administratora (sudo) do dostępu do interfejsu sieciowego.

TrafficSimulator Generator syntetycznego ruchu sieciowego do testowania. Generuje zarówno normalny ruch, jak i symulowane anomalie (port scans, burst traffic).

3.2.2 Detektory anomalii

Statistical Detector Wykorzystuje metody statystyczne do wykrywania anomalii:

- **Z-score**: Oblicza odchylenie standardowe dla częstotliwości portów, rozmiarów pakietów i adresów IP
- **Sliding window**: Analizuje ostatnie N pakietów (domyślnie 100)
- **Progi**: Konfigurowalny threshold (domyślnie 5.0)

Wzór na z-score:

$$z = \frac{x - \mu}{\sigma}$$

gdzie x to obserwowana wartość, μ to średnia, a σ to odchylenie standardowe.

ML Detector Wykorzystuje dwa algorytmy uczenia maszynowego:

1. Isolation Forest [2]

- Algorytm wykrywania outlierów oparty na drzewach decyzyjnych
- Kontaminacja: 3% (oczekiwany procent anomalii)
- Liczba estymatorów: 100

2. One-Class SVM

- Algorytm klasyfikacji binarnej (normalny/anomalny)
- Parametr nu: 0.03
- Kernel: RBF (Radial Basis Function)
- Implementacja z scikit-learn [7]

Ekstrakcja cech:

Listing 2: Ekstrakcja cech dla ML

```
features = [
    src_port,       # Port    rdowy
    dst_port,       # Port docelowy
    packet_size,    # Rozmiar pakietu
    protocol_num,   # Protok    (TCP=6, UDP=17, ICMP=1)
    hour            # Godzina (0-23)
]
```

Whitelist system: Aby zredukować false positive, zaimplementowano system whitelist:

- Znane zakresy IP (Google, Apple, Microsoft, AWS, GitHub)
- Standardowe porty (443/HTTPS, 80/HTTP, 53/DNS, 5353/mDNS, 1900/SSDP)
- Adresy multicast (224.*, 239.*)
- Minimum confidence threshold: 0.3

Heuristic Detector Implementuje reguły wykrywania znanych wzorców ataku:

1. Port Scan Detection:

- Próg: 10 unikalnych portów / 5 sekund z jednego IP
- Wykrywa techniki: SYN scan, TCP connect scan

2. DDoS Detection:

- Próg: 100 połączeń / sekundę do jednego IP
- Wykrywa ataki volumetryczne

3. SYN Flood Detection:

- Analiza ratio pakietów SYN do SYN-ACK
- Próg: $> 80\%$ pakietów to SYN

4. Unusual Ports:

- Wykrywa połączenia do nietypowych portów (> 49152)
- Może wskazywać na backdoory lub malware C&C

Temporal Detector Analizuje wzorce czasowe w ruchu sieciowym:

1. Rate Limiting Violations:

- Wykrywa przekroczenie limitu pakietów/sekundę
- Domyślny limit: 50 pkt/s

2. Burst Detection:

- Wykrywa nagłe skoki w ruchu
- Próg: 200 pakietów w krótkim czasie

3. Beaconing Detection:

- Wykrywa regularne, okresowe połączenia (np. malware C&C)
- Analiza interwałów czasowych między pakietami

4. Off-hours Activity:

- Wykrywa nietypową aktywność w godzinach 2:00-5:00
- Może wskazywać na nieautoryzowane działania

3.3 Dashboard i wizualizacja

Dashboard webowy został zaimplementowany w Flask z wykorzystaniem Socket.IO do komunikacji w czasie rzeczywistym.

3.3.1 Funkcjonalności dashboardu

- **Real-time statistics**: Całkowita liczba pakietów, anomalii, detection rate
- **Wykresy interaktywne**:
  - Timeline anomalii (scatter plot)
  - Rozkład typów anomalii (pie chart)
  - Rozkład severity (bar chart)
  - Top anomaly sources (bar chart)
- **Live security events**: Feed z ostatnimi wykrytymi anomaliami
- **Auto-refresh**: Automatyczna aktualizacja co 2 sekundy

3.4 System raportowania

Moduł `ReportGenerator` generuje trzy typy raportów:

JSON Report

- Struktura danych z pełnymi informacjami o anomaliach
- Wykorzystywany do dalszej analizy programowej
- Zawiera: statystyki, lista anomalii, metadata

CSV Report

- Format tabelaryczny do analizy w Excel/Pandas
- Kolumny: timestamp, type, severity, source_ip, destination_ip, port, confidence

HTML Report

- Raport z interaktywnymi wykresami Plotly
- Zawiera wizualizacje i podsumowanie
- Gotowy do prezentacji

3.5 Konfiguracja systemu

Wszystkie parametry systemu są konfigurowalne przez plik YAML:

Listing 3: Przykladowa konfiguracja (config.yaml)

```yaml
# Zrodlo danych
data_source:
  mode: "simulator"  # pcap, live, simulator
  pcap_file: "data/sample_traffic.pcap"
  network_interface: "en0"

# Detekcja
detection:
  statistical:
    enabled: true
    z_score_threshold: 5.0
    window_size: 100

  ml:
    enabled: true
    isolation_forest:
      contamination: 0.03
      n_estimators: 100
    one_class_svm:
      nu: 0.03
      gamma: 'auto'

  heuristic:
    enabled: true
    port_scan:
      threshold: 10
      time_window: 5

  temporal:
    enabled: true
    rate_limit: 50
    burst_threshold: 200

# Dashboard
dashboard:
  host: "127.0.0.1"
  port: 5000
  debug: true
  auto_refresh: 2
```

4 Testy i wyniki

4.1 Metodologia testowania

System został przetestowany w trzech scenariuszach:

1. **Symulator ruchu**: 1000 pakietów z 10% wygenerowanych anomalii
2. **Analiza PCAP**: Pliki z rzeczywistego ruchu sieciowego
3. **Live capture**: Przechwytywanie w czasie rzeczywistym

4.2 Optymalizacja false positive rate

Jednym z głównych wyzwań było zredukowanie liczby fałszywych alarmów. Przed optymalizacją system wykrywał 42% pakietów jako anomalie, z czego większość stanowił legalny ruch (Google, Apple, Microsoft).

Zmiany optymalizacyjne :

Parametr	**Przed**	**Po**
Contamination (IF)	0.01	0.03
Nu (SVM)	0.01	0.03
Whitelist IP	Brak	10+ zakresów
Whitelist portów	3	11
Min confidence	Brak	0.3

Tabela 3: Zmiany w parametrach ML detektora

Wyniki optymalizacji :

Metryka	**Przed**	**Po**
Detection rate	42.0%	3.2%
ML SVM anomalie	689	0
ML Isolation anomalie	248	50
False positive	∼95%	∼5%

Tabela 4: Porównanie wyników przed i po optymalizacji

Redukcja false positive: 92%

4.3 Analiza wydajności

Metryka	Wartość
Throughput	∼1500 pakietów/sekundę
Latencja detekcji	< 10 ms
Wykorzystanie CPU	∼15-25%
Wykorzystanie RAM	∼150-200 MB

Tabela 5: Charakterystyka wydajnościowa systemu

4.4 Przykłady wykrytych anomalii

Port Scan :

```
Type: port_scan
Source: 192.168.1.100
Unique ports: 25 w 3 sekundy
Severity: HIGH
Confidence: 0.95
```

Nietypowy port :

```
Type: unusual_port
Source: 192.168.1.37
Destination: 208.103.161.2:65283
Severity: MEDIUM
Confidence: 0.76
```

ML Isolation :

```
Type: ml_isolation
Packet size: 1242 bytes
Protocol: UDP
Anomaly score: -0.688
Confidence: 0.34
```

5 Podsumowanie i wnioski

5.1 Podsumowanie

Projekt zakończył się sukcesem - udało się zrealizować w pełni funkcjonalny system wykrywania anomalii w ruchu sieciowym, który:

- **Integruje cztery metody detekcji**: statystyczną, ML, heurystyczną i temporalną
- **Przetwarza dane z trzech źródeł**: PCAP, live capture, simulator
- **Osiąga niski wskaźnik false positive**: 3.2% (po optymalizacji)
- **Dostarcza intuicyjny interface**: real-time dashboard z wykresami
- **Generuje profesjonalne raporty**: JSON, CSV, HTML
- **Jest konfigurowalny**: parametry w YAML, modularna architektura

System został przetestowany na rzeczywistym ruchu sieciowym i skutecznie wykrywa:

- Port scanning
- Ataki DDoS
- Nietypowe wzorce komunikacji
- Potencjalny malware (beaconing, unusual ports)

5.2 Wnioski

5.2.1 Mocne strony projektu

- **Wielowarstwowa detekcja**: Kombinacja różnych metod zwiększa skuteczność wykrywania
- **Whitelist system**: Drastycznie redukuje false positive dla znanego ruchu
- **Modularna architektura**: Łatwe dodawanie nowych detektorów
- **Real-time processing**: Możliwość analizy na żywo z dashboardem
- **Machine Learning**: Wykrywa nieznane wcześniej wzorce anomalii

5.2.2 Wyzwania i rozwiązania

1. **Problem**: Wysoki false positive rate (42%)

   Rozwiązanie: Implementacja whitelist, zwiększenie parametrów ML, minimum confidence threshold

2. **Problem**: Trudność w odróżnieniu normalnego HTTPS od anomalii

   Rozwiązanie: Whitelist znanych dostawców (Google, Apple, etc.), port 443 na whitelist

3. **Problem**: Lokalne protokoły (mDNS, SSDP) wykrywane jako anomalie

   Rozwiązanie: Dodanie portów 5353, 1900 i adresów multicast do whitelist

4. **Problem**: Live capture wymaga uprawnień root

   Rozwiązanie: Dokumentacja z instrukcjami sudo, alternatywne tryby (PCAP, simulator)

5.2.3 Możliwe rozszerzenia

Kierunki dalszego rozwoju systemu:

- **Deep Packet Inspection**: Analiza payload pakietów (obecnie tylko header)
- **Distributed deployment**: Agregacja danych z wielu sensorów
- **Alert system**: Powiadomienia email/SMS/Slack o krytycznych anomaliach
- **Database backend**: PostgreSQL/InfluxDB do długoterminowego przechowywania
- **Advanced ML**: Deep Learning (LSTM, Autoencoders) dla sekwencji pakietów
- **Integration**: API do integracji z SIEM (Splunk, ELK Stack)
- **Threat Intelligence**: Integracja z bazami IoC (Indicators of Compromise)
- **Automated response**: Automatyczne blokowanie podejrzanych IP

5.2.4 Wnioski końcowe

Projekt pokazuje, że skuteczna detekcja anomalii w ruchu sieciowym wymaga:

1. **Hybrydowego podejścia**: Żadna pojedyncza metoda nie jest wystarczająca
2. **Balansu**: Między wrażliwością detekcji a liczbą false positive
3. **Kontekstu**: Whitelist i domain knowledge znacząco poprawiają dokładność
4. **Iteracyjnej optymalizacji**: Parametry ML wymagają dostrojenia do konkretnej sieci

System stanowi solidną bazę do dalszych badań i rozwoju w obszarze cyberbezpieczeństwa, demonstrując praktyczne zastosowanie uczenia maszynowego w wykrywaniu zagrożeń sieciowych.

6 Bibliografia

[1] Philippe Biondi. *Scapy: Packet Crafting for Python*. Accessed: 2025-12-12. 2024. URL: https://scapy.readthedocs.io/.

[2] Fei Tony Liu, Kai Ming Ting i Zhi-Hua Zhou. „Isolation Forest". W: *2008 Eighth IEEE International Conference on Data Mining* (2008), s. 413–422. DOI: 10.1109/ICDM.2008.17. URL: https://ieeexplore.ieee.org/document/4781136.

[3] National Institute of Standards and Technology. *Guide to Intrusion Detection and Prevention Systems (IDPS)*. NIST Special Publication 800-94. 2007. URL: https://csrc.nist.gov/publications/detail/sp/800-94/final.

[4] OWASP Foundation. *OWASP Top Ten 2021*. Accessed: 2025-12-12. 2021. URL: https://owasp.org/www-project-top-ten/.

[5] Pallets Projects. *Flask Web Development Framework*. Accessed: 2025-12-12. 2024. URL: https://flask.palletsprojects.com/.

[6] Python Software Foundation. *Python Programming Language*. Accessed: 2025-12-12. 2024. URL: https://docs.python.org/3/.

[7] scikit-learn developers. *scikit-learn: Machine Learning in Python*. Accessed: 2025-12-12. 2024. URL: https://scikit-learn.org/stable/.

A Instrukcja uruchomienia

A.1 Wymagania systemowe

- Python 3.14 lub nowszy
- Uprawnienia administratora (dla live capture)
- System operacyjny: Linux, macOS lub Windows

A.2 Instalacja

Listing 4: Kroki instalacji

```
# 1. Klonowanie repozytorium
git clone <repository-url>
cd netport-anomaly-detector

# 2. Utworzenie srodowiska wirtualnego
python3 -m venv .venv
source .venv/bin/activate  # Linux/macOS
# lub
.venv\Scripts\activate     # Windows

# 3. Instalacja zaleznosci
pip install -r requirements.txt
```

A.3 Uruchomienie

Listing 5: Przykłady uruchomienia

```
# Tryb simulator (testowanie)
python main.py

# Analiza pliku PCAP
python main.py --mode pcap --pcap-file data/capture.pcap

# Live capture (wymaga sudo)
sudo python main.py --mode live --interface en0

# Bez dashboardu (tylko CLI)
python main.py --no-dashboard

```

```
# Generowanie raportow
python main.py --report-only
```

A.4 Dostęp do dashboardu

Po uruchomieniu, dashboard jest dostępny pod adresem:

`http://127.0.0.1:5000`