Kryminalistyka Cyfrowa

Laboratorium Projekt 2
Budowa systemu analizy sieciowej

Wiktor Zawadzki
Adrian Zalewski

Spis treści

1. Wstęp

1.1. Motywacja

W dobie dynamicznego rozwoju technologii informacyjnych oraz stale rosnącej liczby zagrożeń cybernetycznych, zapewnienie bezpieczeństwa infrastruktury sieciowej staje się kluczowym wyzwaniem dla organizacji. Sieci komputerowe, stanowiące podstawę działania współczesnych systemów informatycznych, są szczególnie narażone na różnorodne formy ataków, takie jak próby nieautoryzowanego dostępu, ataki typu DDoS, infiltracje złośliwego oprogramowania czy wycieki danych. W związku z tym konieczne jest wdrażanie zaawansowanych narzędzi umożliwiających efektywne monitorowanie, analizę i reakcję na incydenty bezpieczeństwa w czasie rzeczywistym

1.2. Cel

Celem tego projektu zbudowanie systemu do analizy bezpieczeństwa ruchu sieciowego, który umożliwi wykrywanie i reagowanie na incydenty bezpieczeństwa w czasie rzeczywistym. Wymagania dotyczące rozwiązania opisano w następnym dziale.

1.3. Wymagania projektowe

ID	Kategoria	Opis wymagania	Wykonanie
A.1	Analiza flow	Wczytywanie plików PCAP przy użyciu NFStream.	Wczytywanie ruchu sieciowego w czasie rzeczywistym lub poprzez plik PCAP
A.2	Analiza Flow	Dla wczytanych przepływów wyświetlanie podsumowania statystyk flow, takich jak podsumowanie ilości przesłanych pakietów pomiędzy danymi hostami.	Generowanie statystyk flow i zapisywanie do raportu w formacie JSON
D.1	Detection as a Code	Implementacja reguł detekcyjnych w Pythonie	Implementacja detekcji ruchu do złośliwych domen oraz detekcji eksfiltracji danych
D.2	Detection as a Code	Wczytywanie reguł w formacie Sigma	Detekcja z wykorzystaniem reguły Sigma: wykrywanie podejrzanych poleceń powershell w odpowiedzi HTTP
ML.1	Machine Learning	Klasyfikacja flow na podstawie cech, takich jak czas trwania, liczba pakietów, protokół	Model uczący oparty o drzewa decyzyjne, wynik w postaci wizualizacji drzewa
ML.2	Machine Learning	Redukcja liczby fałszywych pozytywów (FPR) za pomocą oceny jakości modelu i tuningu hiperparametrów.	Obliczanie metryk FPR, TPR, czułość, dokładność. Wizualizacja macierzy pomyłek

Tabela 1: Wymagania funkcjonalne

ID	Kategoria	Opis wymagania	Wykonanie
ML.3	Machine Learning	Narzędzie powinno pozwalać na trenowanie zawartego w nim modelu ML za pomocą nowych danych dostarczonych przez użytkownika	Wczytywanie dodatkowych plików PCAP i ponowne trenowanie modelu
E.1	Enrichment	Pobieranie podstawowych informacji o IP/domenach, np. z geopy lub innych źródeł Threat Intelligence przy użyciu API.	Enrichment alertu o podejrzanym ruchu do złośliwych domen poprzez wykorzystanie API VirusTotal
V.1	Wizualizacja	Wykres liczby wykrytych zagrożeń w czasie (np. wykres słupkowy)	Diagram kołowy z wykrytych zagrożeń
V.2	Wizualizacja	Mapa geograficzna przedstawiająca lokalizację adresów IP wykrytych jako podejrzane.	Wizualizacja podejrzanych IP na mapie z wykorzystaniem biblioteki Folium

Tabela 2: Wymagania funkcjonalne

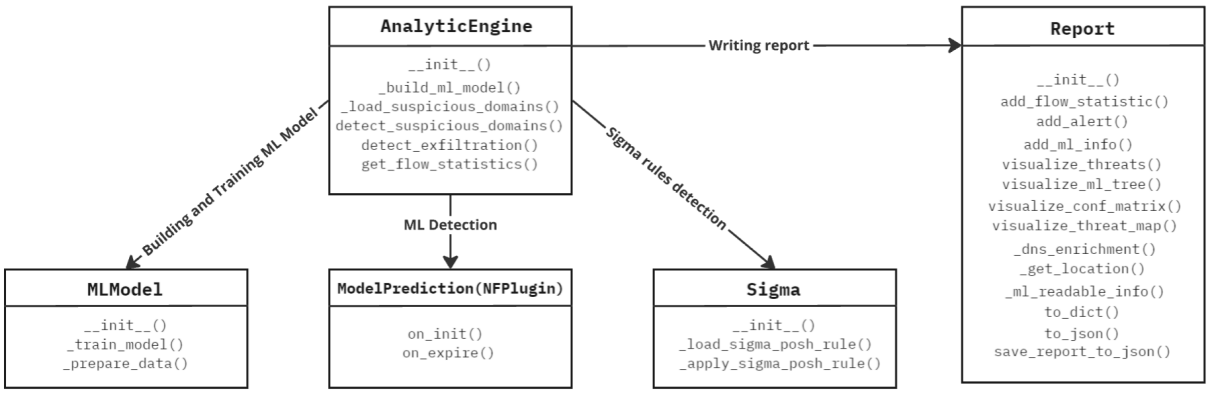
2. Architektura rozwiązania

Centralnym komponentem narzędzia jest klasa `AnalyticEngine` która odpowiada za przechwytywanie strumieni sieciowych i ich dalszą analizę, ocenę stanu bezpieczeństwa tych połączeń. W tej klasie zaimplementowane są metody odpowiedzialne za zebranie statystyk dot. poszczególnych przepływów oraz detekcje zagrożeń. Część funkcjonalności takie jak budowanie drzewa decyzyjnego czy detekcja na podstawie reguł Sigma oraz tworzenie raportu przeniesiono do innych klas.

Klasa `MLModel` odpowiada za budowę, trenowanie i potencjalne retrenowanie drzewca decyzyjnego, który będzie służył do klasyfikacji ruchu - czy jest złośliwy czy też nie. Stworzony model jest wykorzystywany w klasie `ModelPrediction` która dziedziczy po klasie `NFPlugin` i umożliwia tworzenie własnych pluginów do `NFStream`.

Klasa `Sigma` odpowiada za parsowanie reguł Sigma i ich aplikację na ruch sieciowy - tzn. wykonywanie detekcji na podstawie tych reguł.

Klasa `Report` odpowiada za generowanie końcowego raportu. Odpowiada również za tworzenie różnych wizualizacji w tym: wizualizacji zagrożeń na diagramie kołowym i na mapie, wizualizacji modelu drzewa decyzyjnego. Ponadto w tej klasie zdefiniowane są funkcje wzbogacające alerty.



Rysunek 1: Architektura narzędzia

```
python3 nids.py --help
Usage: nids.py [OPTIONS]

Options:
  -s, --source TEXT           Path to PCAP file or interface name e.g. eth0
                              [required]
  -nf, --normal-pcap TEXT     Path to clear PCAP file - useful for training ML
                              model
  -mf, --malicious-pcap TEXT  Path to malicious PCAP file - useful for
                              training ML model
  -o, --output TEXT           Path to output report file
  -ml, --ml-model             Show ML model visualization and save it to file
  -cm, --confusion-matrix     Show confusion matrix and save it to file
  -tpie, --threats-pie        Show threats pie chart and save it to file
  -tmap, --threats-map        Save threat map to file
  -p, --print-report          Print report
  -a, --all                   Show all visualizations
  -l, --live                  Live mode
  --help                      Show this message and exit.
```

Program 1: Interfejs został zaimplementowany przy pomocy biblioteki `click`

3. Analiza Przepływu Danych

3.1. Wymaganie A1

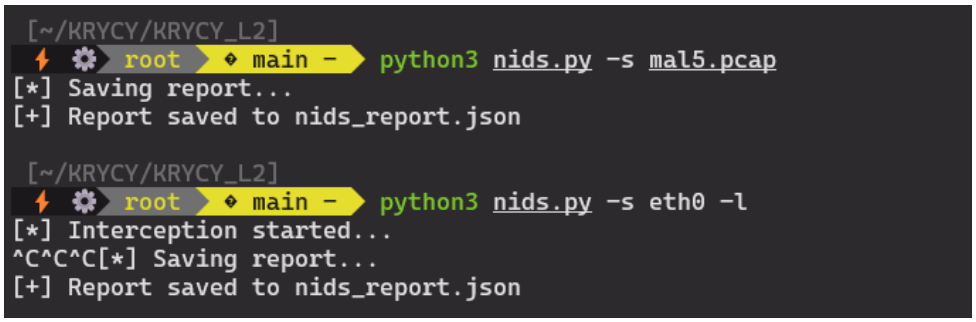
ID	Kategoria	Opis wymagania	Wykonanie
A.1	Analiza flow	Wczytywanie plików PCAP przy użyciu NFStream.	Wczytywanie PCAP i ruchu rzeczywistego

3.1.1. Działanie

Strumień sieciowy można wczytać na 2 sposoby:

- Poprzez plik PCAP - analiza statyczna ruchu sieciowego
- Bezpośrednie pobieranie strumieni z interfejsu

Źródło musi zostać podane po fladze `-s` lub `--source`. Ponadto jeśli uruchamiany jest live capture to należy uruchomić narzędzie z opcją `-l` lub `--live`.



Rysunek 2: **Wczytanie strumienia sieciowego.** Wymaganie A1

3.1.2. Implementacja

Do wczytania strumieni/pakietów wykorzystano biblioteki Scapy i NFStream. Do analizy samych strumieni wykorzystano NFStream gdyż od razu zwraca obiekt który zawiera wszystkie niezbędne do analizy dane, ponadto zwraca dane statystyczne. Z kolei Scapy idealnie nadaje się do manipulacji samymi pakietami, zapewnia wygodny interfejs programistyczny. To w jaki sposób Scapy przechwytuje pakiety zależy od tego czy została podana opcja `--live`.

```python
class AnalyticEngine:
    def __init__(self, input_stream: str, is_live: bool, ml_normal_stream, ml_malicious_stream) -> None:
        # Requirement A.1
        self.report = Report()
        self.model = self._build_ml_model("src/utils/normal_traffic.pcap", "src/utils/malicious_traffic.pcap")
        if ml_malicious_stream != "" and ml_normal_stream != "":
            self.model.retrain_model(ml_normal_stream, ml_malicious_stream)
        self.input_stream = NFStreamer(source=input_stream,
                                       idle_timeout=1,
                                       active_timeout=1,
                                       udps=ModelPrediction(my_model=self.model.tree_model),
                                       statistical_analysis=True
                                       )
        self.get_flow_statistics()
        self.detect_exfiltration()
        if is_live:
            self.scapy_packets = sniff()
        else:
            self.scapy_packets = rdpcap(input_stream)

        self.blacklist = self._load_suspicious_domains()
        self.detect_suspicious_domains(self.blacklist)
        self.sigma = Sigma(self.scapy_packets, self.report)
```

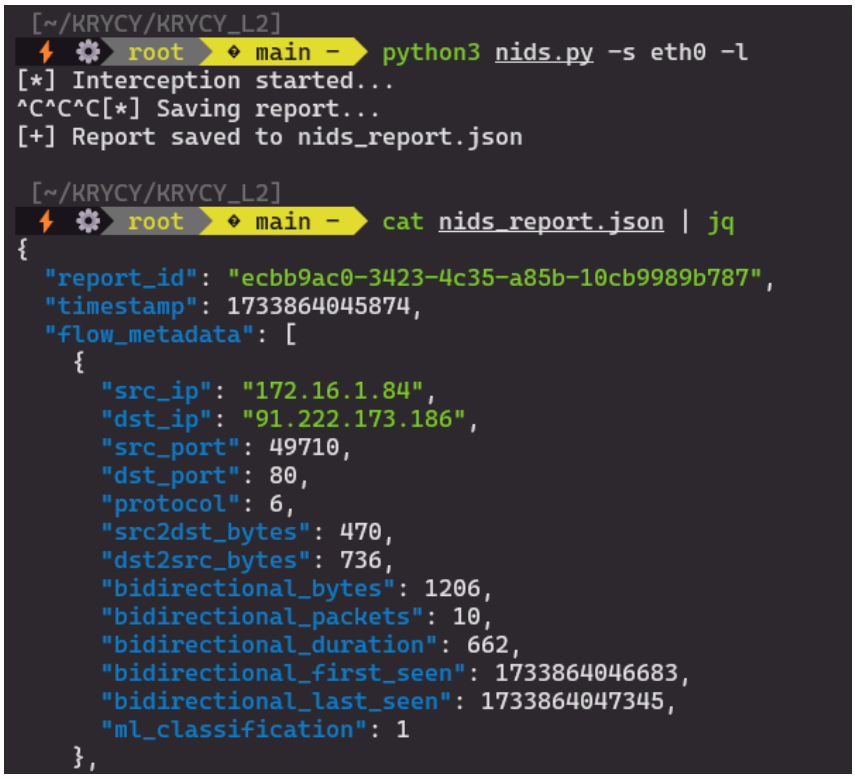
Rysunek 3: **Fragment kodu odpowiadający za wczytanie pliku PCAP**. Wymaganie A1

3.2. Wymaganie A2

ID	Kategoria	Opis wymagania	Wykonanie
A.2	Analiza Flow	Dla wczytanych przepływów wyświetlanie podsumowania statystyk flow, takich jak podsumowanie ilości przesłanych pakietów pomiędzy danymi hostami.	Generowanie statystyk flow i zapisywanie do raportu w formacie JSON

3.2.1. Działanie

Wygenerowany raport zawiera statystki dotyczące danego przepływu. Najważniejszymi logowanymi danymi są: adresy IP, porty, typ protokołu (TCP czy UDP), liczba przesłanych bajtów (do, z, w obie strony), znaczniki czasowe wskazujące początek konwersacji i jego koniec. Ponadto zawarta jest informacja o klasyfikacji, która określa czy dane flow jest złośliwe czy nie (patrz sekcja ML1).



Rysunek 4: **Fragment raportu z narzędzia**. Wymaganie A2

3.2.2. Implementacja

Za implementację tego wymagania odpowiada funkcja `get_flow_statistics` w klasie `AnalyticEngine`. Funkcja realizuje dodanie do raportu wyżej wspominanych danych

```python
def get_flow_statistics(self):
    # Requirement A.2
    for flow in self.input_stream:
        self.report.add_flow_statistic(
            flow.src_ip, flow.dst_ip, flow.src_port, flow.dst_port, flow.protocol,
            flow.src2dst_bytes, flow.dst2src_bytes, flow.bidirectional_bytes,
            flow.bidirectional_packets, flow.bidirectional_duration_ms,
            flow.bidirectional_first_seen_ms, flow.bidirectional_last_seen_ms, flow.udps.model_prediction
        )
```

Rysunek 5: **Funkcja implementująca zbieranie statystyk**. Wymaganie A2

4. Detection as a Code

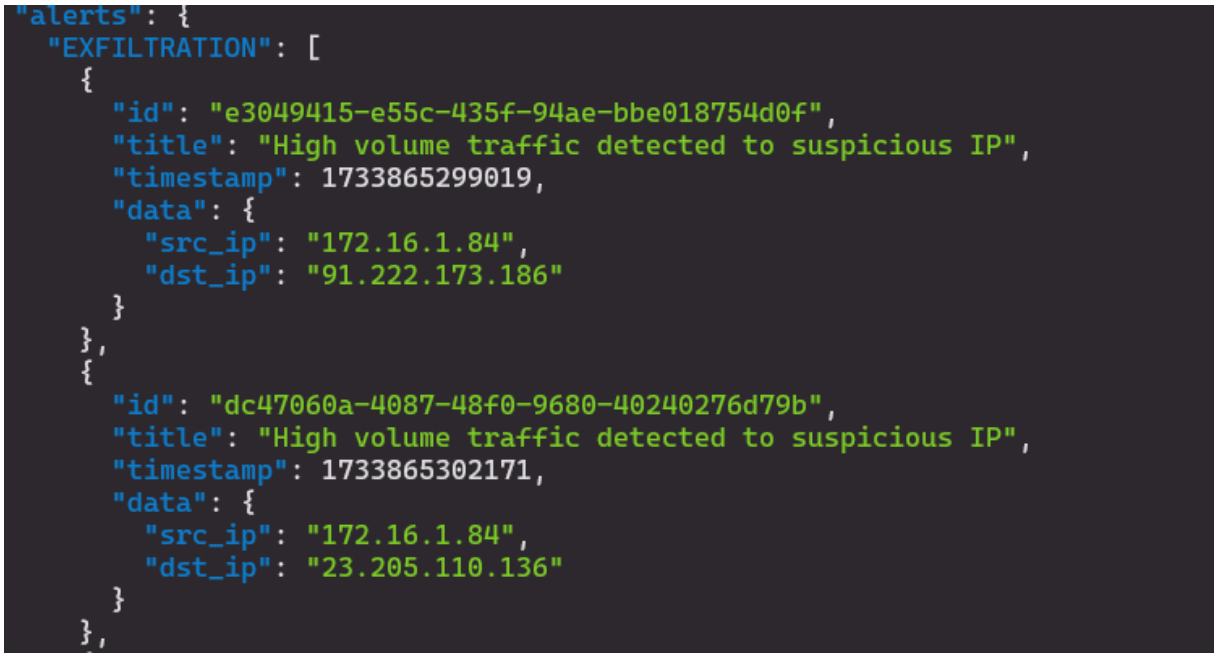
4.1. Wymaganie D1

ID	Kategoria	Opis wymagania	Wykonanie
D.1	Detection as a Code	Implementacja reguł detekcyjnych w Pythonie	Implementacja detekcji ruchu do złośliwych domen oraz detekcji eksfiltracji danych

Testy przeprowadzono przy pomocy narzędzia `tcpreplay` przy pomocy tego samego pliku PCAP.

4.1.1. Detekcja eksfiltracji

Zadaniem tej reguły jest wykrycie eksfiltracji danych do zewnętrznych odbiorców - detekcja ruchu o dużej objętości do podejrzanych adresów. Poniżej znajduje się przykładowy rezultat pozytywnego wykrycia takiego incydentu.

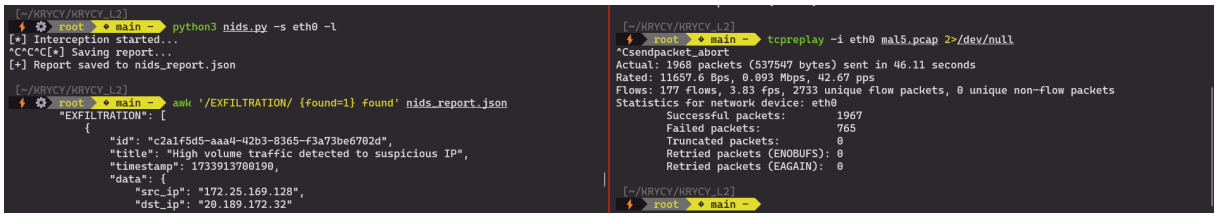


Rysunek 6: **Fragment raportu z narzędzia**. Wymaganie D1

Algorytm detekcyjny przy pomocy biblioteki NFStream sprawdza liczbę przesłanych bajtów z źródła do hosta docelowego. Jeśli przekroczony jest pewien próg, to do raportu dodawany jest alert, wskazujący na możliwą eksfiltrację danych z sieci.

```python
def detect_exfiltration(self, threshold=1000):
    reported_ips = []
    for flow in self.input_stream:
        if flow.src2dst_bytes > threshold:
            ip_pair = (flow.src_ip, flow.dst_ip)
            if ip_pair not in reported_ips:
                self.report.add_alert(
                    "EXFILTRATION",
                    "High volume traffic detected to suspicious IP",
                    int(flow.bidirectional_first_seen_ms),
                      {"src_ip": flow.src_ip, "dst_ip": flow.dst_ip}
                      )
                reported_ips.append(ip_pair)
                if flow.src_ip not in self.report._suspicious_ips or flow.dst_ip not in self.report._suspicious_ips:
                    self.report._suspicious_ips.append(flow.dst_ip)
```

Rysunek 7: **Implementacja detekcji**. Wymaganie D1



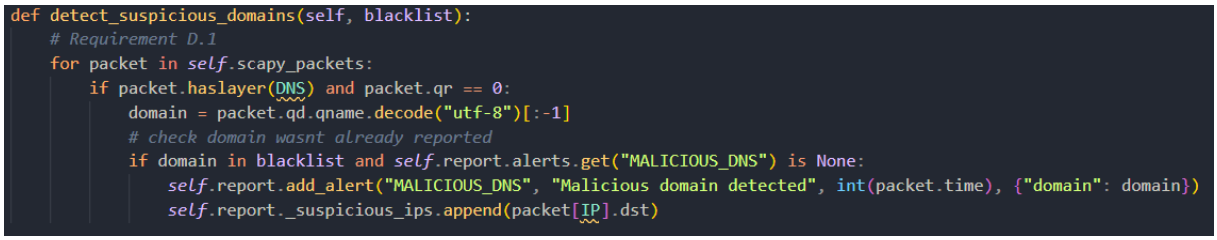
Rysunek 8: **Symulacja ruchu sieciowego przy pomocy `tcpreplay` i testowanie detekcji**

4.1.2. Detekcja złośliwych domen

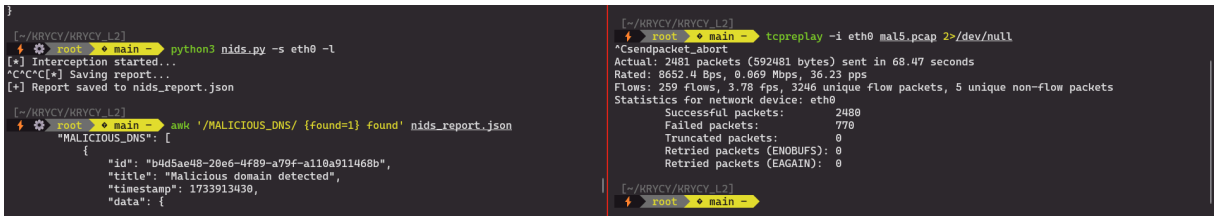
Zadaniem tej reguły jest detekcja domen uznawanych za złośliwe. Poniżej znajduje się przykładowy wpis w raporcie. Ten alert jest poddawany procesowi wzbogacania (*enrichment*), więcej na ten temat w dziale poświęconemu wzbogacaniu (Enrichment. Wymaganie E1).

Rysunek 9: **Fragment raportu z narzędzia**. Wymaganie D1

Algorytm detekcyjny wykorzystuje bibliotekę Scapy i sprawdza każdy pakiet pod kątem odwoływania się do złośliwych domen. To czy domena jest złośliwa czy nie określane jest na podstawie blacklisty. Na potrzeby samego eksperymentu stworzono krótką blacklistę składającą się tylko z 2 domen, w rzeczywistości należałoby zaimplementować znacznie większe listy (np. listy od CERT Polska). Blacklista tworzona jest w momencie tworzenia obiektu klasy `AnalyticEngine`, co można zauważyć na rysunku 2, za stworzenie blacklisty odpowiada metoda prywatna `_load_suspicious_domain`, gdzie domeny są ładowane z pliku.



Rysunek 10: **Funkcja odpowiadająca za detekcję złośliwych domen**. Wymaganie D1



Rysunek 11: **Symulacja ruchu sieciowego przy pomocy `tcpreplay` i testowanie detekcji.**

4.2. Wymaganie D2

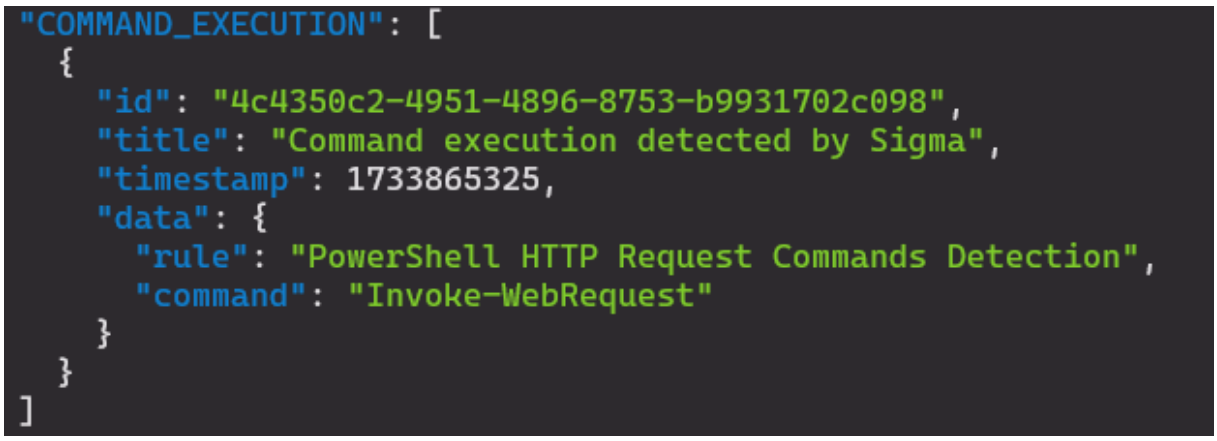
ID	Kategoria	Opis wymagania	Wykonanie
D.2	Detection as a Code	Wczytywanie reguł w formacie Sigma	Detekcja z wykorzystaniem reguły Sigma: wykrywanie podejrzanych poleceń powershell w odpowiedzi HTTP

4.2.1. Detekcja poleceń Powershell w HTTP

Zadaniem tej reguły jest detekcja poleceń Powershell (wykorzystywanych do interakcji z innymi hostami) w odpowiedziach HTTP pochodzących od zewnętrznych hostów.

```yaml
title: PowerShell HTTP Request Commands Detection
description: Detects PowerShell commands commonly used for HTTP operations like
Invoke-WebRequest, Invoke-RestMethod, WebClient
status: experimental
severity: medium

logsource:
  category: process_creation
  product: windows

detection:
  selection:
    CommandLine|contains:
      - 'Invoke-WebRequest'
      - 'iwr '
      - 'Invoke-RestMethod'
      - 'irm '
      - 'Net.WebClient'
      - 'DownloadString'
      - 'DownloadFile'
      - 'Invoke-Expression'
      - 'IEX'
      - 'wget '
      - 'curl '
      - 'WebRequest'

  condition: selection
```

Program 2: Reguła Sigma do detekcji poleceń Powershell



Rysunek 12: **Fragment raportu z narzędzia**. Wymaganie D2

Niezdecydowano się na wykorzystanie biblioteki `pysigma` z racji na bardzo słabą/nieczytelną dokumentację oraz brak praktycznych przykładów. Ponadto stwierdzono że biblioteka głównie odpowiada za parsowanie reguły, nie ma żadnych interfejsów programistycznych pozwalających na sprawną detekcję zagrożeń w strumieniu sieciowym. Zdecydowano się przeparsować bezpośrednio sam plik `yaml` oraz zaaplikować detekcję bezpośrednio na strumień. Na detekcję przy pomocy reguł `Sigma` odpowiada klasa o analogicznej nazwie.

src > sigma.py > Sigma > _apply_sigma_posh_rule

```python
import yaml
from scapy.all import *

class Sigma:
    def __init__(self, scapy_packets, report):
        self.scapy_packets = scapy_packets
        self.report = report
        self._apply_sigma_posh_rule()

    def _load_sigma_posh_rule(self, sigma_rules_path="src/sigma_rules/detect_powershell_http.yml"):
        # Requirement D.2
        with open(sigma_rules_path) as f:
            try:
                sigma_content = yaml.safe_load(f)
            except yaml.YAMLError as exc:
                print(exc)
        return sigma_content

    def _apply_sigma_posh_rule(self):
        findings = []
        rule = self._load_sigma_posh_rule()
        for packet in self.scapy_packets:
            if packet.haslayer(Raw) and TCP in packet:
                if packet[TCP].sport == 80 and len(packet[TCP].payload) < 500:
                    try:
                        raw_data = packet[Raw].load.decode("utf-8")
                    except:
                        continue
                    if rule.get("detection"):
                        for command in rule["detection"]["selection"]["CommandLine|contains"]:
                            if command in raw_data:
                                self.report.add_alert(
                                    "COMMAND_EXECUTION",
                                    "Command execution detected by Sigma",
                                    int(packet.time), {"rule": rule["title"], "command": command})
                                if packet[IP].src not in self.report._suspicious_ips:
                                    self.report._suspicious_ips.append(packet[IP].src)
                                break
```

Rysunek 13: **Klasa odpowiadająca za detekcję zagrożeń przy pomocy formatu Sigma**. Wymaganie D2

```
[~/KRYCY/KRYCY_L2]
root ⋄ main ~ python3 nids.py -s eth0 -l
[*] Interception started...
^C^C^C[*] Saving report...
[+] Report saved to nids_report.json

[~/KRYCY/KRYCY_L2]
root ⋄ main ~ awk '/COMMAND_EXECUTION/ {found=1} found' nids_report.json
    "COMMAND_EXECUTION": [
        {
            "id": "18535137-978a-46b0-9c1d-0232ba1231a5",
            "title": "Command execution detected by Sigma",
```

```
[~/KRYCY/KRYCY_L2]
root ⋄ main ~ tcpreplay -i eth0 mal5.pcap 2>/dev/null
^Csendpacket_abort
Actual: 2418 packets (585896 bytes) sent in 65.66 seconds
Rated: 8922.3 Bps, 0.071 Mbps, 36.82 pps
Flows: 247 flows, 3.76 fps, 3183 unique flow packets, 5 unique non-flow packets
Statistics for network device: eth0
        Successful packets:        2417
        Failed packets:            770
        Truncated packets:         0
        Retried packets (ENOBUFS): 0
        Retried packets (EAGAIN):  0
```

Rysunek 14: **Symulacja ruchu sieciowego przy pomocy `tcpreplay` i testowanie detekcji**

5. Machine Learning

5.1. Wymaganie ML1

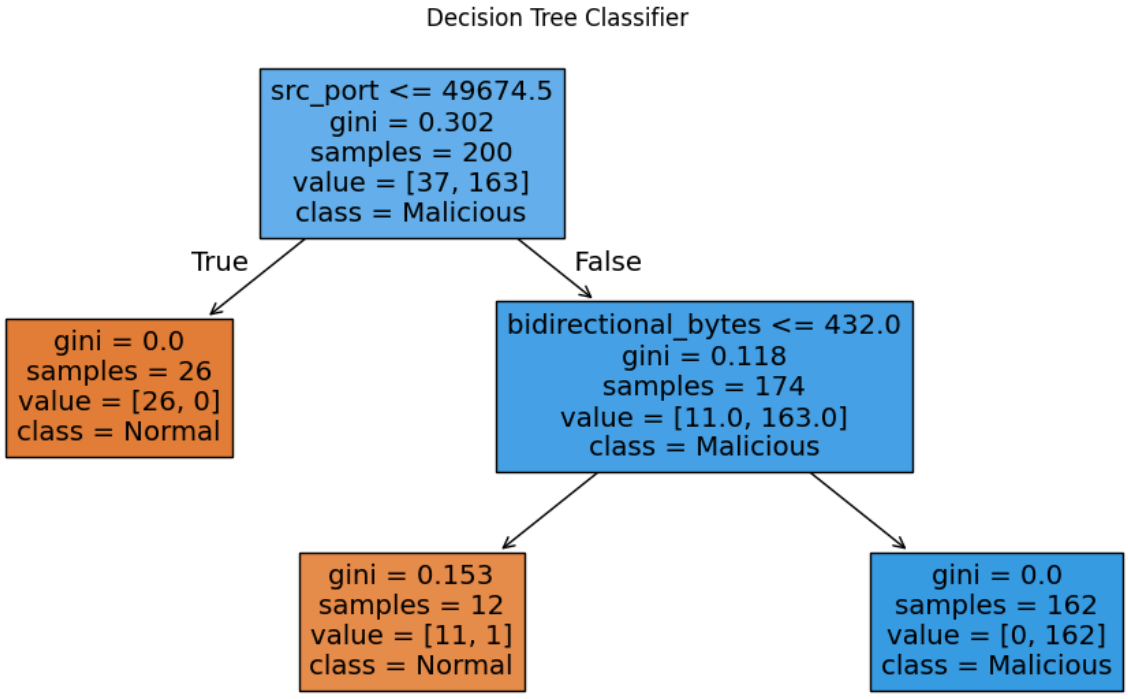
ID	Kategoria	Opis wymagania	Wykonanie
ML.1	Machine Learning	Klasyfikacja flow na podstawie cech, takich jak czas trwania, liczba pakietów, protokół	Model uczący oparty o drzewa decyzyjne, wynik w postaci wizualizacji drzewa

5.1.1. Wizualizacja drzewa decyzyjnego

Klasyfikację przepływów oparto na strukturze drzewa decyzyjnego. Klasyfikacja flow opiera się na poniższych cechach przepływów:

```
selected_columns = [
      'bidirectional_packets',
      'bidirectional_bytes',
      'src2dst_packets',
      'src2dst_bytes',
      'dst2src_packets',
      'dst2src_bytes',
      'bidirectional_duration_ms',
      'src2dst_duration_ms',
      'dst2src_duration_ms',
      'src_port',
      'dst_port',
      'protocol',
      'label'
]
```

Drzewo można wyświetlić przy pomocy opcji `-ml` lub `--ml-model` (+ zapis do pliku).



Rysunek 15: **Drzewo decyzyjne**. Wymaganie ML1

Wstępny model rozpoczyna proces uczenia się o domyślnie pliki PCAP znajdujące się w `/src/utils`. Użytkownik może załadować własne dodatkowe pliki o czym więcej w rozdziale ML3. Za budowanie i obsługę modelu uczenia maszynowego odpowiada klasa `MLModel` i jej 2 metody:

- `_prepare_data()` - odpowiadająca za przygotowanie danych - wstępna obróbka, wybranie właściwych cech
- `_train_model()` - odpowiadająca za trenowanie modelu

Rysunek 16: **Metody klasy `MLModel`**. Wymaganie ML1

Zbudowany model jest wykorzystywany w klasie `ModelPrediction`, która dziedziczy po klasie `NFPlugin` - umożliwia to tworzenie własnych pluginów, które mogą być później zaaplikowane do `NFStream`. Zatem będzie można na bieżąco klasyfikować ruch.

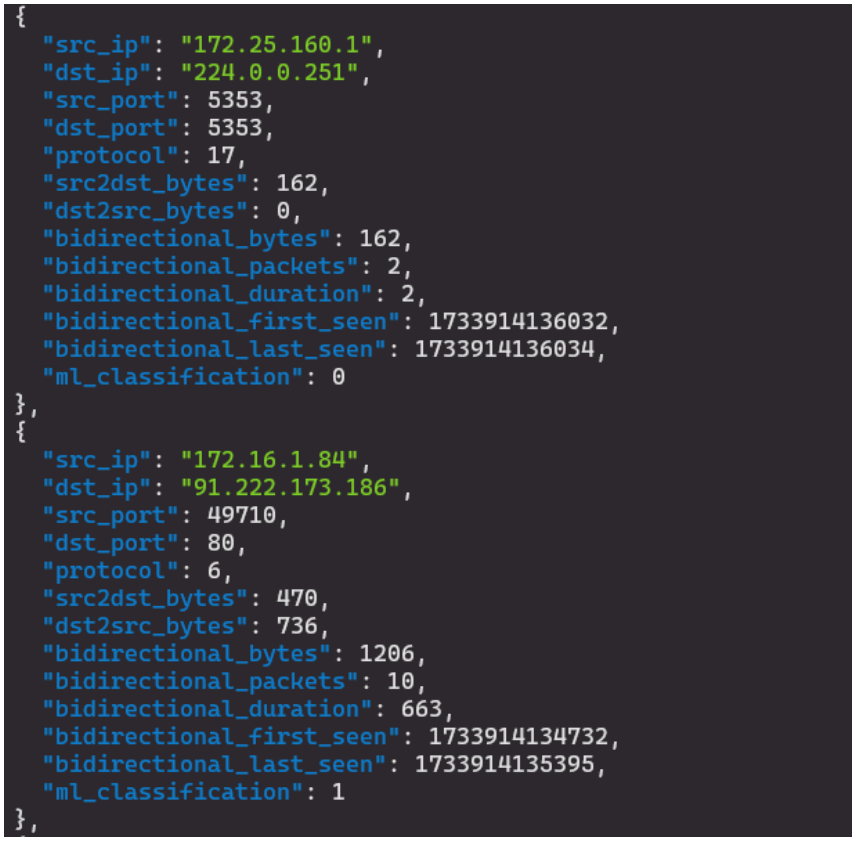
```
class ModelPrediction(NFPlugin):
  def on_init(self, packet, flow):
      flow.udps.ml_prediction = 0

  def on_expire(self, flow):
      numerical_features = [
          flow.bidirectional_packets,
          flow.bidirectional_bytes,
          flow.src2dst_packets,
          flow.src2dst_bytes,
          flow.dst2src_packets,
          flow.dst2src_bytes,
          flow.bidirectional_duration_ms,
          flow.src2dst_duration_ms,
          flow.dst2src_duration_ms,
          flow.src_port,
          flow.dst_port,
          flow.protocol
      ]

      to_predict = np.array(numerical_features).reshape(1, -1)
      flow.udps.model_prediction = self.my_model.predict(to_predict)[0]
```

Poniżej znajduje się wykorzystanie powyżej stworzonego pluginu:

```
from src.ml_model import MLModel, ModelPrediction
...
class AnalyticEngine:
...
def __init__(...):
  self.model = self._build_ml_model(ml_normal_stream, ml_malicious_stream)
  self.input_stream = NFStreamer(
    source=input_stream,
    idle_timeout=1,
    active_timeout=1,
    udps=ModelPrediction(self.model.tree_model),
    statistical_analysis=True
  )
  ...
```

Program 5: Fragment `AnalyticEngine` odpowiadający za podłączenie pluginu do strumienia

Stworzony plugin po podłączeniu do strumienia będzie klasyfikował ruch tak jak poniżej (parametr `ml_classification`). Klasyfikacja przebiegła poprawnie, wg. VirusTotala IP o adresie 224.0.0.251 nie jest złośliwe, natomiast 91.222.173.186 zostało oznaczone przez 8 skanerów jako zagrożenie.

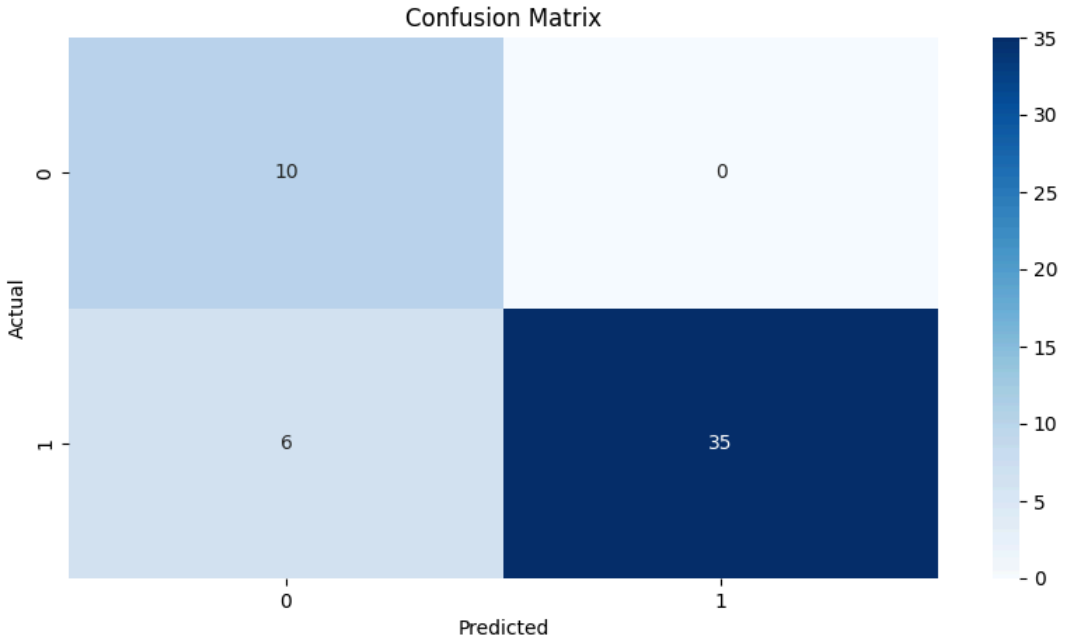


Rysunek 17: Klasyfikacji konkretnych przepływów

5.2. Wymaganie ML2

ID	Kategoria	Opis wymagania	Wykonanie
ML.2	Machine Learning	Redukcja liczby fałszywych pozytywów (FPR) za pomocą oceny jakości modelu i tuningu hiperparametrów.	Obliczanie metryk FPR, TPR, czułość, dokładność. Wizualizacja macierzy pomyłek

W celu lepszego dostosowywania parametrów modelu uczenia maszynowego i jego szczegółową analizę, narzędzie zwraca metryki takie jakie dokładność, czułość oraz precyzję. Ponadto istnieje możliwość wyświetlenia macierzy pomyłek przy pomocy opcji `-cm` lub `--confusion-matrix`, wówczas wizualizacja pokaże się na ekranie i zostanie również zapisana do pliku.



Rysunek 18: **Macierz pomyłek dla stworzonego modelu**. Wymaganie ML2.

```
[~/KRYCY/KRYCY_L2]
 root  main - python3 nids.py -s mal5.pcap --confusion-matrix --print-report | tail -n 8
    "ml_info": {
        "accuracy": 0.9607843137254902,
        "recall": 0.9512195121951219,
        "precision": 1.0
    }
}
[*] Saving report...
[+] Report saved to nids_report.json
```

Rysunek 19: **Metryki modelu uczenia maszynowego w raporcie**. Wymaganie ML2.

Metryki są zapisywane w momencie zakończenia procesu uczenia w funckji `_train_model`.

```
self.accuracy = accuracy_score(Y_test, predictions)
self.conf_matrix = confusion_matrix(Y_test, predictions)
self.recall = self.conf_matrix[1][1] / (self.conf_matrix[1][0] + self.conf_matrix[1][1])
self.precision = self.conf_matrix[1][1] / (self.conf_matrix[0][1] + self.conf_matrix[1][1])
self.tree_model = tree_model
self.X_train = X_train
```

Rysunek 20: **Zapisywanie metryk**. Wymaganie ML2.

5.3. Wymaganie ML3

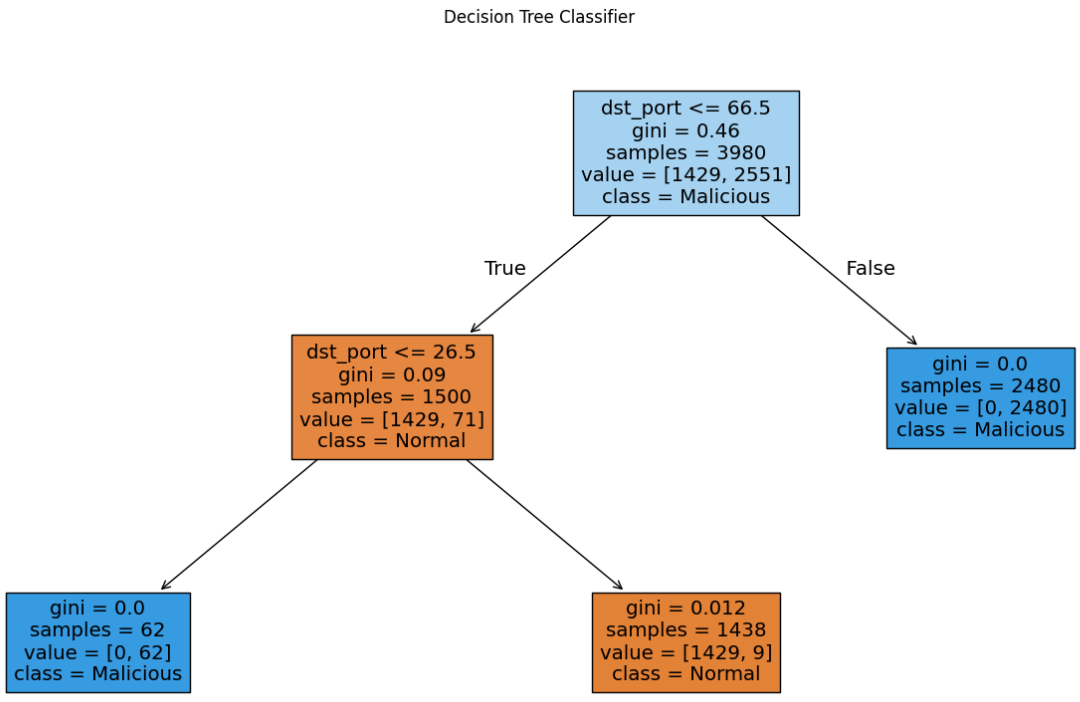
ID	Kategoria	Opis wymagania	Wykonanie
ML.3	Machine Learning	Narzędzie powinno pozwalać na trenowanie zawartego w nim modelu ML za pomocą nowych danych dostarczonych przez użytkownika	Wczytywanie dodatkowych plików PCAP i ponowne trenowanie modelu

Wczytanie dodatkowych danych odbywa się poprzez podanie dodatkowych plików PCAP przy uruchamianiu narzędzia z opcją `-nf` (`--normal-pcap`) oraz `-mf` (`--malicious-pcap`). Wraz z podaniem tych opcji narzędzia dokona ponownego trenowania modelu (stworzonego wcześniej na podstawie domyślnych w narzędziu plików), wraz z końcem trenowania możliwe jest od razu wykorzystanie modelu do analizy strumieni sieciowych.

```
[~/KRYCY/KRYCY_L2]
✗ root ◆ main – python3 nids.py -s mal5.pcap -nf add_norm.pcap -mf add_mal.pcap
-a
[*] Generating ML model visualization...
[+] ML model visualization saved to ml_tree.png
[*] Generating confusion matrix...
[+] Confusion matrix saved to confusion_matrix.png
[*] Generating threats pie chart...
[+] Threats pie chart saved to threats_pie.png
[*] Generating threat map...
[+] Threat map saved to threat_map.html
[*] Saving report...
[+] Report saved to nids_report.json
```

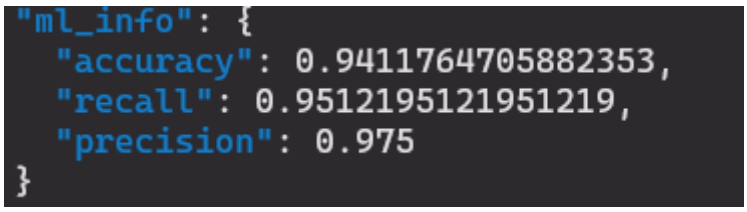
Rysunek 21: **Wczytanie dodatkowych danych.** Wymaganie ML3

Podstawowy model decyzyjny ulegał znacznej zmianie, zmieniły się cechy klasyfikacji oraz znacznie wzrosła liczba próbek.



Rysunek 22: **Nowy model drzewa decyzyjnego.** Wymaganie ML3

Każdy z parametrów modelu zmniejszył się nieznacznie, wartości są nadal wysokie, co może być zjawiskiem niepożądanym (*overfitting*) jeśli np. dokładność zbioru testowego będzie znacznie mniejsza.



Rysunek 23: **Nowe metryki**. Wymaganie ML3

Za ponowne trenowanie modelu z nowymi danymi odpowiada funkcji `retrain_model` w klasie `MLModel`.

```python
def retrain_model(self, normal_stream, malicious_stream):
    # Requirement ML.3
    norm_network_flow_stream = NFStreamer(source=normal_stream, statistical_analysis=True)
    mal_network_flow_stream = NFStreamer(source=malicious_stream, statistical_analysis=True)
    self.norm_network_flow_stream = norm_network_flow_stream
    self.mal_network_flow_stream = mal_network_flow_stream
    self._train_model()
```

Rysunek 24: **Implementacja ponownego trenowania z nowymi danymi**. Wymaganie ML3

6. Enrichment

6.1. Wymaganie E1

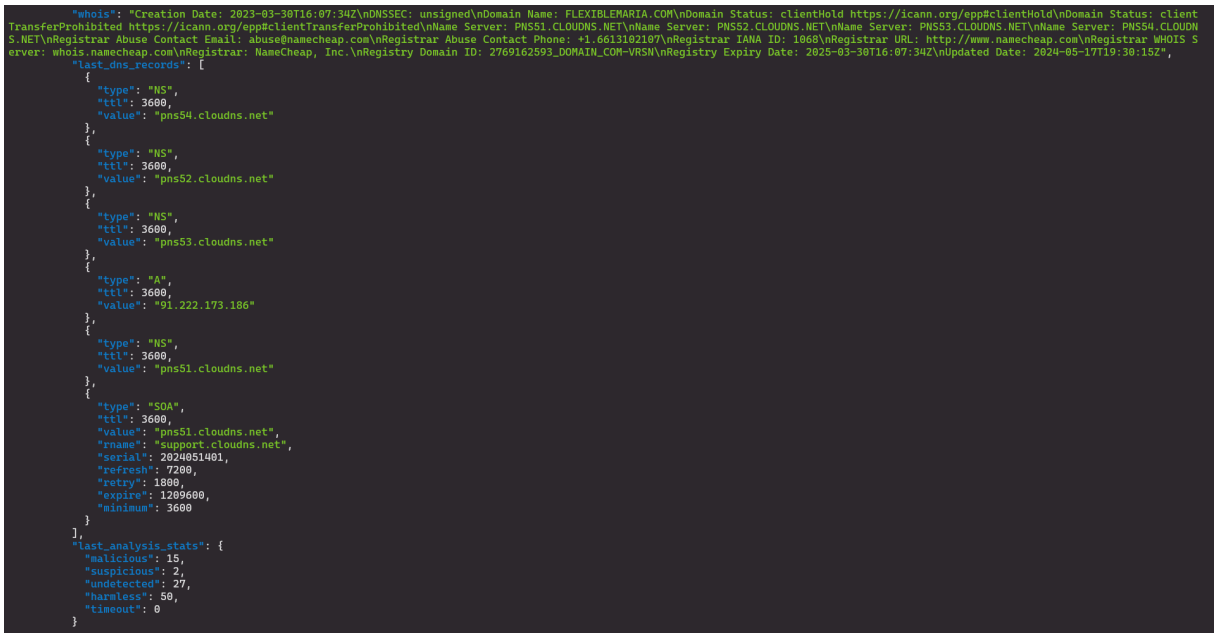
ID	Kategoria	Opis wymagania	Wykonanie
E.1	Enrichment	Pobieranie podstawowych informacji o IP/domenach, np. z geopy lub innych źródeł Threat Intelligence przy użyciu API.	Enrichment alertu o podejrzanym ruchu do złośliwych domen poprzez wykorzystanie API VirusTotal

6.1.1. Domain Enrichment

Domyślny alert o złośliwej domenie zostaje wzbogacony o informacje pochodzące od VirusTotal, zwracane są takie wartości jak ostatni certyfikat, ostatni klucz publiczny, ostatnie zapisane rekordy DNS, wynik polecenia `whois` oraz wyniki skanów systemów antywirusowych.

```
"MALICIOUS_DNS": [
  {
    "id": "18a83eea-3416-4d97-9246-52cdd975081c",
    "title": "Malicious domain detected",
    "timestamp": 1715731281,
    "data": {
      "domain": "flexiblemaria.com",
      "last_https_certificate": {
        "size": 1580,
        "public_key": {
          "rsa": {
            "key_size": 4096,
            "modulus": "00ca681e56d7f254fadbaab8dcab285ead44b0553af12ec4c97c9766cd53ecdc7c87b96388cfe9c0ffe294d0511a14039d48888e32041916831e5d09d449a98635dceabd3abe6e5961cfb57c7b9bf62924146cf755276b0b9df6c8fea709fdd13117ab056d5f628b17d29eb94b2be0e270860f4b14c9844f6dae5b6cd854668ebdde841e8c8984e818ca6d8cdf7b19032f53a3826df430a0c966af1c513896deb065d73820e5cd58d128147b8b5d968c10fa9ab9b521c0a90efc37f7f43f3edab75e4083bce9d94661a52411e24b856202a42766e455ba52d5db4849b562f225e5f893b1ee212cfb3ef5b6a6749d8f1b3625c80210c66433df4da189fdab841c8f275a969a20db6d81bc1fc3d380c1ab76351d154d5e6e4e663ba8e2f244a4969cdb15a0a997858fe0635a42a81e3f6d871d209432931f1bd0ca7ce92a34172ca37d8bc311c2635d0fbcc7a81d4c146a82d32ef94c2cb4289156d2c8be936383099aaa0eec0ad866629dd461eebb174783295b18fdc006528158045 9c0c4d0da2013bd12820000 4d4d47629fef72d7e9a4b59ef602028fcece87e69bafe7a8b4bce34df44ea593460305c856ae5e6d0e66d6bf898735cc70f0b09ebbb7e7c0dda9fff0f42db9a1090c4e2630f91e96dcf72a14118ba2436b92c4e6acb69bc71b6a87978482dcb83dcb60c6f1ce2f888e5f1164e26e47ab593da1d9e5cde2f6a27711",
            "exponent": "010001"
          },
          "algorithm": "RSA"
        },
        "thumbprint_sha256": "fbeaa13b6ba674f561306a870188f7806eaeed2a44668b8b0da6a47c3336e7ae",
        "tags": [],
        "cert_signature": {
          "signature_algorithm": "sha256RSA",
          "signature": "474aac1c93404ff404bf2ff0f82222910f0d6bc614b1e70a2089d3de9a5bd5369a2c693541a1a60e413008cd80d5d412c9aa4753e037c0cb87ba30fe7570a29f7fe9ebb35230f32eeb49c221a002b0799963df0814765302916626 0c3b93e8fd3d0fa7fe4a18902ad8531bc1913645b63920b785ecb41e337cda42189494b699e0188efc8c42f6a7860c4a24bb699749dc0f1c866463e977627 1bc236db54ab665e2621cd63f22d8ef9138cda41bc937962e7edabb58aec25ce5201263ebe911276450f733da6a5c9dce3e086db640a58a0811e828f25748185e3400ea5f9ac560e7960516dc5898ab76b639321194095c28c7e34abe253716354a9a27eead051"
        },
        "validity": {
          "not_after": "2023-06-29 04:31:17",
          "not_before": "2023-03-31 04:31:18"
        },
        "version": "V3",
        "extensions": {
          "certificate_policies": [
            "2.23.140.1.2.1",
            "1.3.6.1.4.1.44947.1.1.1"
          ],
          "extended_key_usage": [
            "serverAuth",
            "clientAuth"
          ],
          "tags": [],
          "subject_alternative_name": [
            "flexiblemaria.com"
          ],
```


Rysunek 25: **Wzbogacenie alertu o złośliwych domenach**. Wymaganie E1



Rysunek 26: **Wzbogacenie alertu o złośliwych domenach**. Wymaganie E1

Jak zostało wyżej wspomniane, implementacja opiera się o komunikację do REST API VirusTotala, wymaga to podania klucza API, żeby nie hardcodować tej wartości, jest ona wczytywana przez zmienną środowiskową, co jest istotne, gdyż w razie jej braku, narzędzie nie będzie działać w pełni poprawnie.

```python
def _dns_enrichment(self, alert: Dict) -> None:
    # Requirement E.1
    domain = alert["data"]["domain"]
    url = f"https://www.virustotal.com/api/v3/domains/{domain}"
    headers = {
        "accept": "application/json",
        "x-apikey": environ.get("VT_API_KEY")
    }
    response = requests.get(url, headers=headers)
    alert["data"]["last_https_certificate"] = response.json()["data"]["attributes"]["last_https_certificate"]
    alert["data"]["whois"] = response.json()["data"]["attributes"]["whois"]
    alert["data"]["last_dns_records"] = response.json()["data"]["attributes"]["last_dns_records"]
    alert["data"]["last_analysis_stats"] = response.json()["data"]["attributes"]["last_analysis_stats"]
```

Rysunek 27: **Implementacja wzbogacania domen**. Wymaganie E1

6.1.2. IP Geolocation Enrichment

Zadaniem tego wzbogacenia jest zwrócenie geograficznej lokalizacji danego IP. Zwracane dane są wykorzystywane później w wizualizacji zagrożeń na mapie.

```python
def _get_location(self, ip: str) -> tuple:
    data=requests.get(f"https://geolocation-db.com/json/{ip}&position=true").json()
    return data["latitude"], data["longitude"]
```

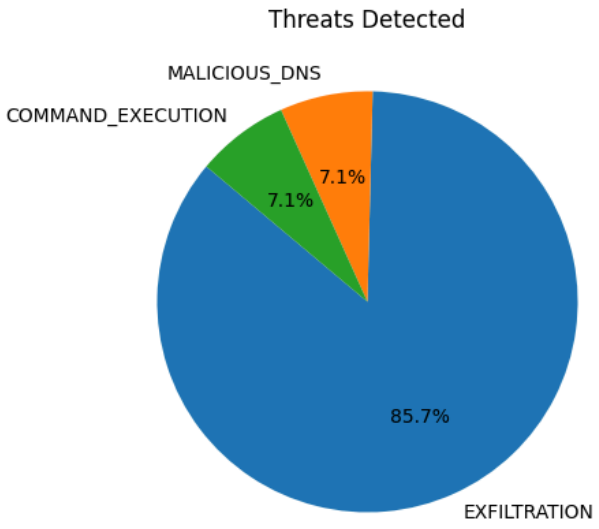
Program 6: **Funkcja odpowiedzialna za IP enrichment.** Wymaganie E1

7. Wizualizacja

7.1. Wymaganie V1

ID	Kategoria	Opis wymagania	Wykonanie
V.1	Wizualizacja	Wykres liczby wykrytych zagrożeń w czasie (np. wykres słupkowy)	Diagram kołowy z wykrytych zagrożeń

Wizualizacja wykrytych zagrożeń na diagramie kołowym odbywa się wraz z uruchomieniem narzędzia z opcją `-tpie` lub `--threats-pie`, wówczas obraz jest zapisywany również do pliku.



Rysunek 28: **Diagram kołowy z zagrożeniami**. Wymaganie V1

Diagram kołowy jest generowany przy pomocy biblioteki `matplotlib`.

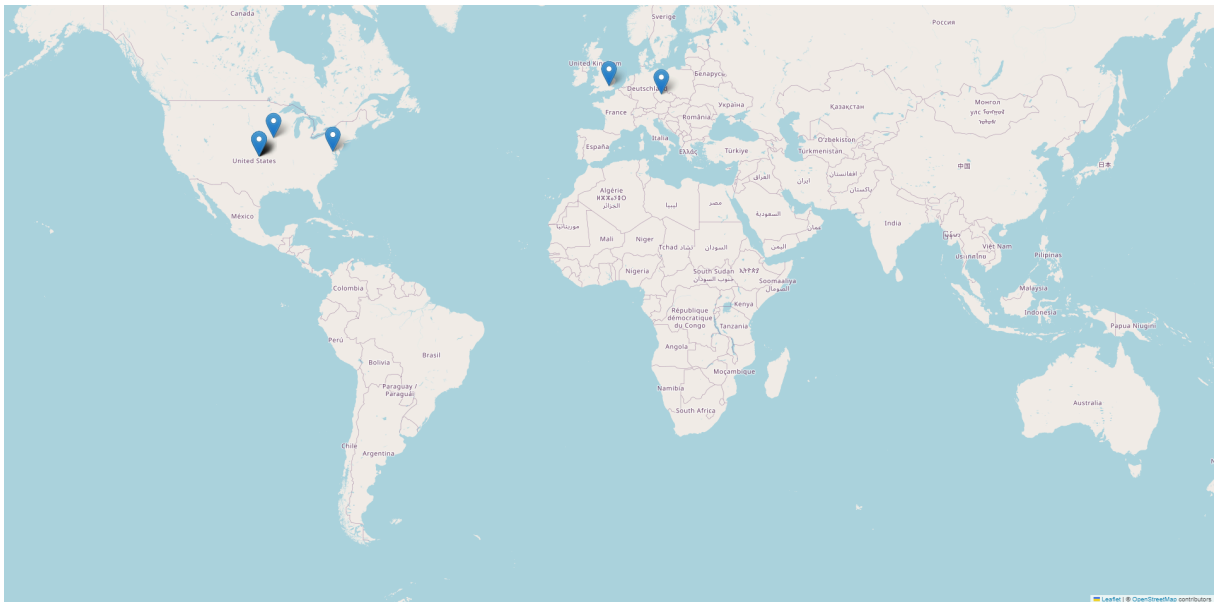
```
  def visualize_threats(self):
    threat_count = {}
    for alert_type in self.alerts:
        threat_count[alert_type] = len(self.alerts[alert_type])
      plt.figure(figsize=(10, 5))
      plt.pie(threat_count.values(), labels=threat_count.keys(), autopct="%1.1f%%",
startangle=140)
      plt.title("Threats Detected")
      plt.savefig("threats_pie.png")
      plt.show()
```

Program 7: **Funkcja odpowiadająca za wizualizację zagrożeń na diagramie kołowym.** Wymaganie V1

7.2. Wymaganie V2

ID	Kategoria	Opis wymagania	Wykonanie
V.2	Wizualizacja	Mapa geograficzna przedstawiająca lokalizację adresów IP wykrytych jako podejrzane.	Wizualizacja podejrzanych IP na mapie z wykorzystaniem biblioteki Folium

Mapa z zagrożeniami zapisuje się do pliku `threat_map.html`, wraz z wywołaniem narzędzia z opcją `-tmap` lub `--threats-map`



Rysunek 29: **Wizualizacja podejrzanych IP.** Wymaganie V2

Mapa jest generowana dzięki bibliotece `Folium` i funkcji wzbogacającej IP opisanej w sekcji IP Geolocation Enrichment.

```python
def visualize_threat_map(self):
    map = folium.Map(location = [0.0, 0.0], zoom_start=3, zoom_control=False,scrollWheelZoom=False,dragging=False)
    feature_group = folium.FeatureGroup("Threats")
    for ip in self._suspicious_ips:
        lat, long = self._get_location(ip)
        if type(lat) != float or type(long) != float:
            continue
        feature_group.add_child(folium.Marker([lat, long], popup=ip))
    map.add_child(feature_group)
    map.save("threat_map.html")
```

Rysunek 30: **Funkcja odpowiadająca za wizualizację zagrożeń na mapie.** Wymaganie V2.

8. Podsumowanie

W ramach projektu udało się stworzyć narzędzie pozwalające na wykrywanie zagrożeń w sieci w czasie rzeczywistym, niemniej jednak narzędzie nie jest doskonałe i wymaga dalszych usprawnień, w tym:

- wczytywania większej ilości dodatkowych danych w różnych formatach dla modelu uczenia maszynowego
- parsowania różnych reguł Sigma, nie tylko konkretnej reguły
- zwiększenie liczby reguł detekcyjnych i poprawa obecnych

Projekt/laboratorium pozwoliło nam zgłębić zagadnienia dotyczące:

- analizy/inspekcji pakietów lub przepływów przy pomocy `Scapy` i `NFStream`
- uczenia maszynowego i jego wykorzystania do wykrywania zagrożeń w strumieniach sieciowych
- pisania reguł detekcji (*Detection as a Code*) przy pomocy Pythona i bibliotek takich jak `NFStream` i `Scapy`
- wzbogacania danych z użyciem narzędzi CTI (*Cyber Threat Intelligence*)
- działania systemów NDR