nmap-harvester

A supervised machine learning model designed for detecting `NMAP` port scanning, developed as part of a university project.

Introduction

This project aims to build a supervised machine learning model to detect real-time `NMAP` port scanning activities.

In many cyber-attacks, the initial step often involves port scanning using tools like `NMAP`. Detecting such scans can be challenging because network packets carry extensive information, and a single packet isn't enough to confirm an `NMAP` scan attempt.

To address this, this project proposes a machine learning-based approach for identifying TCP port scans initiated by `NMAP`.

Dataset Creation

Understanding TCP connections is important for building the dataset. When a TCP packet is sent over the network, it carries specific flags that facilitate the `3-Way Handshake`. `NMAP` can manipulate these flags to evade detection while performing rapid port scans.

Key Dataset Characteristics

- **Session-Based Rows:** Instead of logging each packet individually, each row in the dataset represents a **session** (requests + responses).
- **Flag Summation:** Flags (`SYN`, `ACK`, `FIN`, etc.) are aggregated across the session. For example:
  - If a `SYN` packet is sent by `NMAP` and another `SYN` is part of the response, the `SYN` column will record a value of `2`.

Example Dataset Row

```
start_request_time,end_request_time,start_response_time,end_respons
2025-01-08 13:52:55.274814,2025-01-08 13:52:55.274814,2025-01-08
13:52:55.274874,2025-01-08 13:52:55.274874,6e-05,"['172.31.0.1',
'172.31.0.2']","['172.31.0.1', '172.31.0.2']","['44031',
'3306']","['44031', '3306']",1,1,0,1,0,0,1
```

- Sessions are grouped by `src_ip`, `dst_ip`, `src_port`, and `dst_port`. However, these grouping keys are excluded from the model's training phase.
- The `duration` feature provides valuable information for distinguishing between legitimate traffic and `NMAP` scans, as legitimate HTTP requests may exhibit similar flag behavior but differ in timing.
- The session window is set to **0.5 seconds** by default, as this is typically enough to capture an `NMAP` scan attempt.

Common `NMAP` Scans

```
nmap -sT 172.31.0.1 -p 0-10000 # TCP Scan
nmap -sS 172.31.0.1 -p 0-10000 # Stealth Scan
nmap -sF 172.31.0.1 -p 0-10000 # FIN Scan
nmap -sN 172.31.0.1 -p 0-10000 # NULL Scan
nmap -sX 172.31.0.1 -p 0-10000 # XMAS Scan
```

The dataset consists of: - `bad.csv`: Sessions labeled as `1` (`NMAP` traffic). - `good.csv`: Sessions labeled as `0` (legitimate traffic).

These datasets are combined into a single file, `merged.csv`, for training the model.

Machine Learning Model

The `XGBClassifier` was selected as the final model due to its reliable performance in key areas:

1. High MCC score (`~0.91`)
2. High accuracy score (`~0.95`)
3. Fast prediction speed (`~3ms` on average for 15,000 rows)

Model Performance Example

```
XGBClassifier (n_estimators=210): Accuracy: 0.9599, Train time: 68ms, Prediction time: 3ms, MCC: 0.919739, TP: 718, TN: 741, FN: 28, FP: 33

Best Classifier based on MCC
Classifier: XGBClassifier
n_estimators: 210
MCC Score: 0.919739
```

The speed of prediction played a significant role in choosing this model, as it allows efficient analysis of large volumes of network traffic in real time.

Training Dataset

The training dataset, `datasets/train/merged.csv`, is generated using the following approach:

1. Create an isolated Docker environment for capturing clean data:

   ```
   docker compose up --build
   ```

2. Access the container:

   ```
   docker attach traffic_generator
   ```

3. Run the interceptor on the host:

   ```
   sudo python3 interceptor.py
   ```

   - Adjust:
     - `interface`: Docker network interface name
     - `scanner_ip`: IP assigned to `traffic_generator`
     - `output_file`: Output CSV file path
     - `label`: `0` for legitimate traffic, `1` for `NMAP` scans

4. Run `NMAP` scans from the container:

   ```
   nmap -sT 172.31.0.1 -p 0-10000
   nmap -sS 172.31.0.1 -p 0-10000
   ```

5. Run noise traffic for legitimate requests:

   ```
   python3 noiser.py
   ```

6. Merge datasets:

```
cd datasets
python3 merger.py
```

7. Train the model:

```
python3 algo_chooser.py
```

Delayed Dataset

A delayed dataset can be created by introducing delays between requests:

```
nmap -p 1-10000 --scan-delay 1s 172.31.0.1
```

You can also adjust the delay in legitimate requests by modifying `SLEEP_SECOND` in `noiser.py`.

Results:

```
Dataset loaded with 11351
records.
Dataset preprocessed
successfully.
   duration  SYN  ACK  FIN  RST  URG  PSH
label
0  0.000060    1    1    0    1    0    0
1
1  0.000068    1    1    0    1    0    0
1
2  0.000062    1    1    0    1    0    0
1
3  0.000057    1    1    0    1    0    0
1
4  0.000074    1    1    0    1    0    0
1
Dataset split into training and testing sets.

....
XGBClassifier (n_estimators=210): Accuracy: 1.0000, Train time:
28ms, Prediction time: 3ms, MCC: 1.000000, TP: 743, TN: 393, FN:
0, FP: 0

Best Classifier based on MCC
Classifier: XGBClassifier
```

```
n_estimators: 210
MCC Score: 1.000000
```

Running the Detector

To run the detector:

```
sudo python3 detector.py
```

- The detector uses `interceptor.py` to monitor session packets.
- `injector.py` simulates normal HTTP traffic with occasional `NMAP` scans (~10% probability).
- If at least **30%** of session packets are flagged as anomalies, the system will detect an ongoing `NMAP` attack.

Requirements

Install dependencies with:

```
python3 -m venv venv && source venv/bin/activate && pip install -r requirements.txt
```

Demonstration Video

https://github.com/user-attachments/assets/f10773c6-742e-4394-913e-42beb0cc3683

References

- Unix Stack Exchange - Detecting `NMAP` Scans
- Medium Article on `NMAP` Detection

External Dependencies

- `pyshark`
- `python-nmap`