ANOMALY DETECTION SYSTEM USING BERT

Internship Project — Portfolio Version

SEPTEMBER 18, 2024

INAAM KABBARA
EPITA University – Ubility Company

Table of Contents

1. Abstract

Cybersecurity systems are constantly exposed to anomalies such as unauthorized logins, failed authentications, and malicious network activities, all of which may indicate potential attacks. Traditional detection approaches often rely on static rules or signatures, which are limited in identifying evolving or unknown threats. This project addresses the challenge by leveraging Large Language Models (LLMs), specifically BERT, to classify anomalies from both system logs and network traffic data.

During this internship project, I designed and implemented a complete anomaly detection pipeline: collecting and preprocessing system logs (auth.log) and network captures (.pcap), tokenizing the data using BERT's tokenizer, and fine-tuning bert-base-uncased for binary classification. The system was automated end-to-end, from raw data ingestion to model training and inference, with results saved as performance metrics and visual charts. The model successfully differentiated between normal and anomalous events, demonstrating the feasibility of applying LLMs to cybersecurity anomaly detection. This represents my first practical project with LLMs during my internship, laying the foundation for more advanced applications of AI in autonomous security response.

2. Introduction

In modern cybersecurity operations, Security Operations Centers (SOCs) are responsible for monitoring vast amounts of data generated by system logs and network traffic. Within this data, anomalies such as failed login attempts, unauthorized access, or unusual communication patterns often serve as early indicators of cyberattacks. Detecting these anomalies accurately and efficiently is therefore a critical task for protecting IT infrastructures.

Traditional anomaly detection approaches in SOCs typically rely on predefined rules or signature-based methods. While effective against known threats, these approaches struggle with new, evolving, or sophisticated attack techniques that do not match existing patterns. This limitation creates the need for more adaptive, intelligent detection methods capable of learning from diverse data sources.

The goal of this project is to explore whether **Large Language Models (LLMs)**, specifically BERT, can be adapted to classify anomalies directly from raw system logs and network traffic. By fine-tuning BERT on preprocessed datasets, the model learns to distinguish between normal and anomalous events, going beyond static rule-based detection.

This project holds particular importance in my professional journey as it represents my **first practical experience with LLMs applied to cybersecurity**. Developed during my internship, it allowed me to bridge academic knowledge with hands-on implementation, laying the groundwork for more advanced research and projects in AI-driven cybersecurity and autonomous incident response.

3. Methodology

3.1 Environment Setup

The development environment for this project was built on a Linux-based system (Kali Linux) to provide a stable and security-focused platform for experimentation. A dedicated **Python virtual environment** was created to ensure reproducibility and to isolate project dependencies from the system-wide configuration.

The following key libraries and frameworks were installed and configured:

- **Transformers** (Hugging Face) — for BERT model loading, tokenization, and fine-tuning.
- **Scapy** — for parsing and processing .pcap network traffic files.
- **PyTorch (torch)** — as the deep learning backend for training and inference.
- **scikit-learn (sklearn)** — for dataset splitting and evaluation metrics.
- **Matplotlib** — for generating performance visualizations.

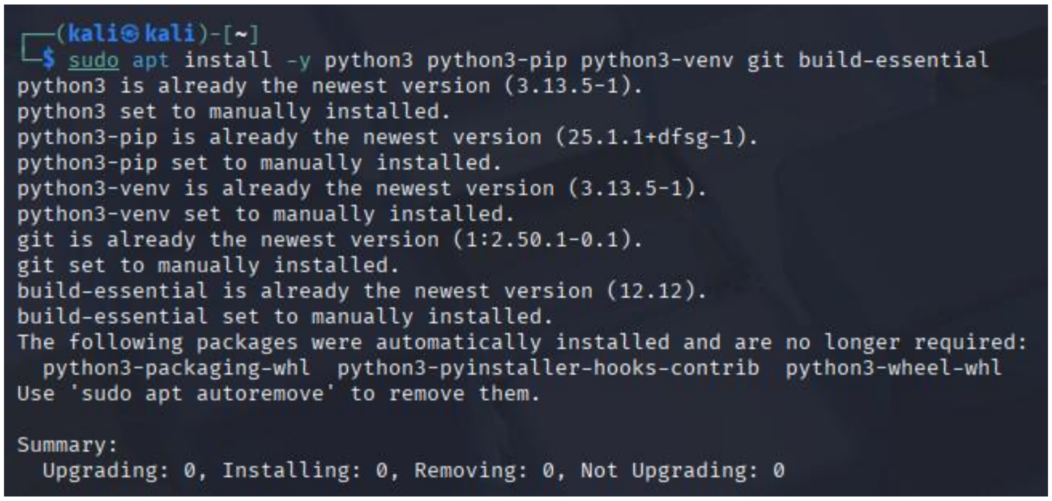
In addition, two external tools were installed for testing and data generation:

- **Tcpdump** — to capture network traffic and produce .pcap files.
- **Metasploit Framework** — to simulate attack scenarios, such as brute-force attempts, for generating realistic anomalous data.

To maintain an organized workflow, the project structure was divided into clear folders:

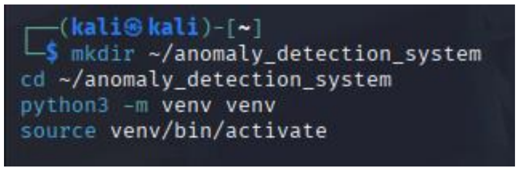
- /data — for storing raw input files (auth.log, capture.pcap).
- /src — for all Python scripts.
- /results — for preprocessed datasets, trained models, metrics, and charts.
- /docs — for report materials, diagrams, and screenshots.

Screenshots of the installation process and folder creation have been included in the appendices to illustrate the complete environment setup. This setup ensured a clean, modular, and reproducible base for the anomaly detection pipeline.



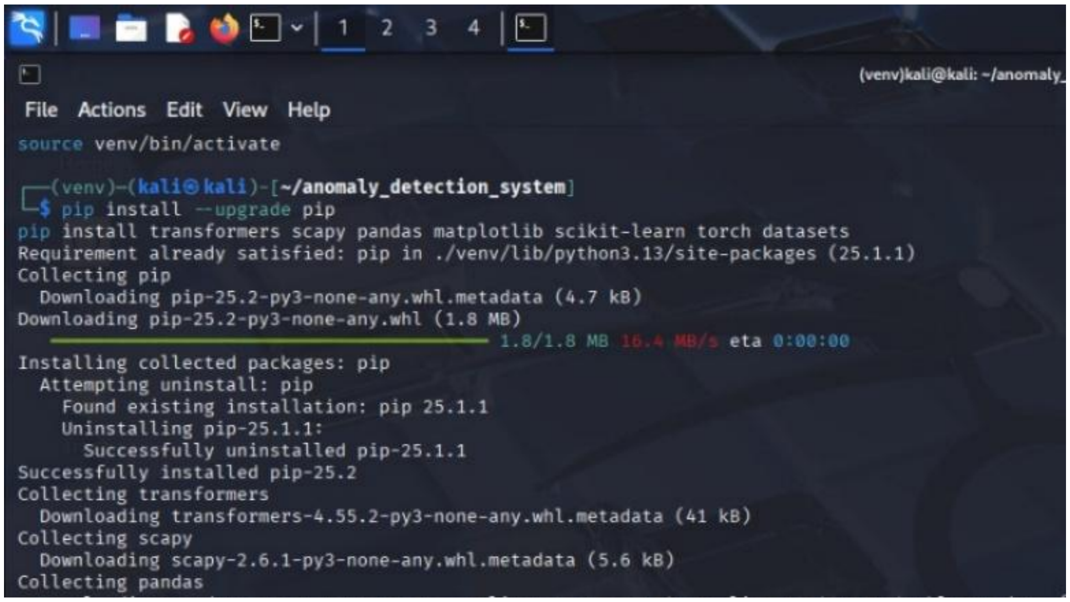
```
┌──(kali㉿kali)-[~]
└─$ sudo apt install -y python3 python3-pip python3-venv git build-essential
python3 is already the newest version (3.13.5-1).
python3 set to manually installed.
python3-pip is already the newest version (25.1.1+dfsg-1).
python3-pip set to manually installed.
python3-venv is already the newest version (3.13.5-1).
python3-venv set to manually installed.
git is already the newest version (1:2.50.1-0.1).
git set to manually installed.
build-essential is already the newest version (12.12).
build-essential set to manually installed.
The following packages were automatically installed and are no longer required:
  python3-packaging-whl  python3-pyinstaller-hooks-contrib  python3-wheel-whl
Use 'sudo apt autoremove' to remove them.

Summary:
  Upgrading: 0, Installing: 0, Removing: 0, Not Upgrading: 0
```

Python installation



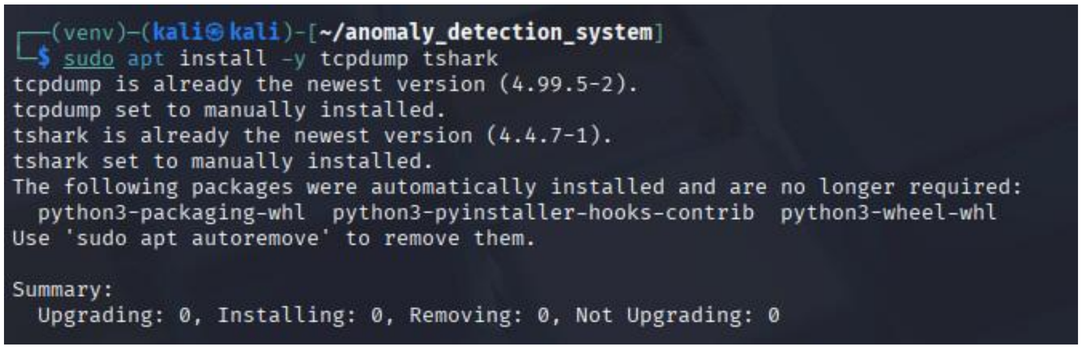
```
┌──(kali㉿kali)-[~]
└─$ mkdir ~/anomaly_detection_system
cd ~/anomaly_detection_system
python3 -m venv venv
source venv/bin/activate
```

Creating the main folder



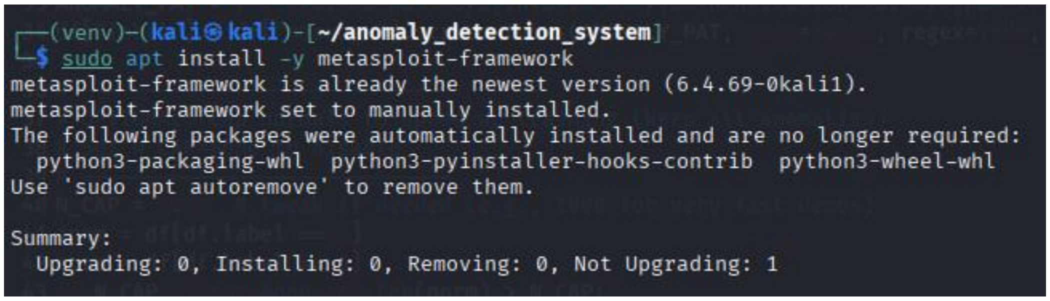
```
source venv/bin/activate

┌──(venv)─(kali㉿kali)-[~/anomaly_detection_system]
└─$ pip install --upgrade pip
pip install transformers scapy pandas matplotlib scikit-learn torch datasets
Requirement already satisfied: pip in ./venv/lib/python3.13/site-packages (25.1.1)
Collecting pip
  Downloading pip-25.2-py3-none-any.whl.metadata (4.7 kB)
Downloading pip-25.2-py3-none-any.whl (1.8 MB)
   ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 1.8/1.8 MB 16.4 MB/s eta 0:00:00
Installing collected packages: pip
  Attempting uninstall: pip
    Found existing installation: pip 25.1.1
    Uninstalling pip-25.1.1:
      Successfully uninstalled pip-25.1.1
Successfully installed pip-25.2
Collecting transformers
  Downloading transformers-4.55.2-py3-none-any.whl.metadata (41 kB)
Collecting scapy
  Downloading scapy-2.6.1-py3-none-any.whl.metadata (5.6 kB)
Collecting pandas
```

Installing the libraries



```
┌──(venv)─(kali㉿kali)-[~/anomaly_detection_system]
└─$ sudo apt install -y tcpdump tshark
tcpdump is already the newest version (4.99.5-2).
tcpdump set to manually installed.
tshark is already the newest version (4.4.7-1).
tshark set to manually installed.
The following packages were automatically installed and are no longer required:
  python3-packaging-whl  python3-pyinstaller-hooks-contrib  python3-wheel-whl
Use 'sudo apt autoremove' to remove them.

Summary:
  Upgrading: 0, Installing: 0, Removing: 0, Not Upgrading: 0
```

Installing tcpdump



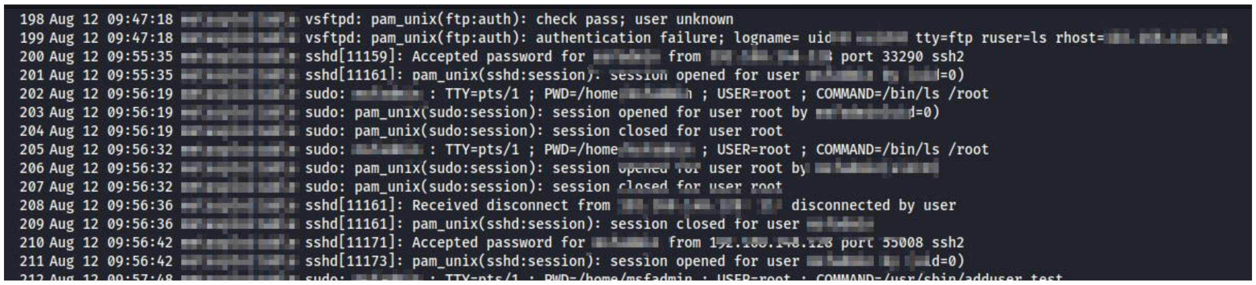
```
┌──(venv)─(kali㉿kali)-[~/anomaly_detection_system]
└─$ sudo apt install -y metasploit-framework
metasploit-framework is already the newest version (6.4.69-0kali1).
metasploit-framework set to manually installed.
The following packages were automatically installed and are no longer required:
  python3-packaging-whl  python3-pyinstaller-hooks-contrib  python3-wheel-whl
Use 'sudo apt autoremove' to remove them.

Summary:
  Upgrading: 0, Installing: 0, Removing: 0, Not Upgrading: 1
```

Installing Metasploit Framework

3.2 Data Collection

The project relies on two main sources of input data: **system logs** and **network traffic captures**, both of which were generated and collected in a controlled environment to simulate realistic attack scenarios.

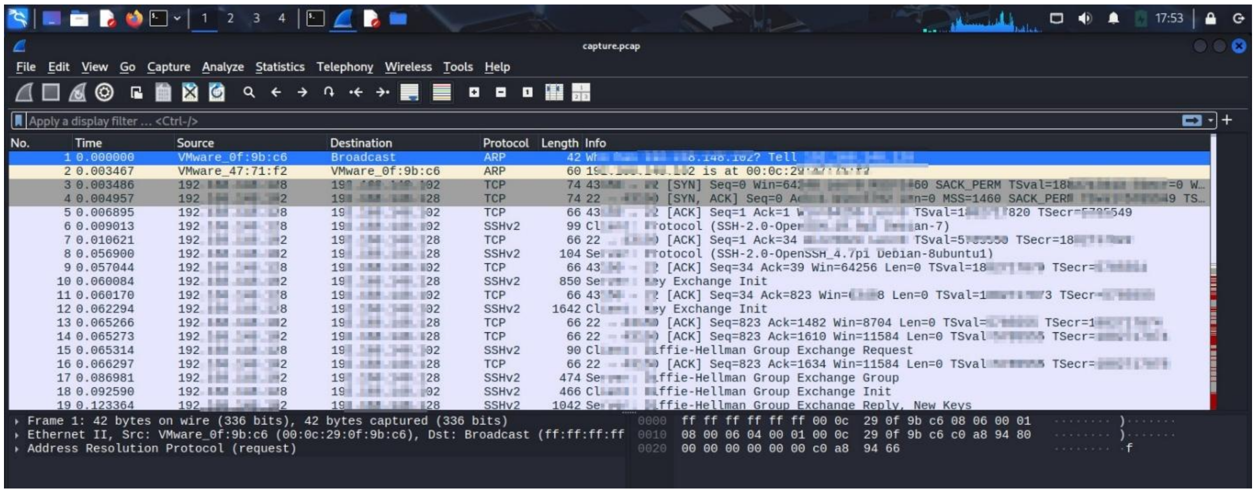
- **System Logs (auth.log)**
  The auth.log file was used to represent authentication-related events on a Linux system. It contained both normal user activity and anomalous events such as failed password attempts or invalid user logins. These anomalies are particularly relevant in SOC operations, as they often serve as indicators of brute-force attacks or unauthorized access attempts.



```
198 Aug 12 09:47:18 [redacted] vsftpd: pam_unix(ftp:auth): check pass; user unknown
199 Aug 12 09:47:18 [redacted] vsftpd: pam_unix(ftp:auth): authentication failure; logname= uid[redacted] tty=ftp ruser=ls rhost=[redacted]
200 Aug 12 09:55:35 [redacted] sshd[11159]: Accepted password for [redacted] from [redacted] port 33290 ssh2
201 Aug 12 09:55:35 [redacted] sshd[11161]: pam_unix(sshd:session): session opened for user [redacted]=0)
202 Aug 12 09:56:19 [redacted] sudo: [redacted] : TTY=pts/1 ; PWD=/home[redacted] ; USER=root ; COMMAND=/bin/ls /root
203 Aug 12 09:56:19 [redacted] sudo: pam_unix(sudo:session): session opened for user root by [redacted]=0)
204 Aug 12 09:56:19 [redacted] sudo: pam_unix(sudo:session): session closed for user root
205 Aug 12 09:56:32 [redacted] sudo: [redacted] : TTY=pts/1 ; PWD=/home[redacted] ; USER=root ; COMMAND=/bin/ls /root
206 Aug 12 09:56:32 [redacted] sudo: pam_unix(sudo:session): session opened for user root by [redacted]
207 Aug 12 09:56:32 [redacted] sudo: pam_unix(sudo:session): session closed for user root
208 Aug 12 09:56:36 [redacted] sshd[11161]: Received disconnect from [redacted] disconnected by user
209 Aug 12 09:56:36 [redacted] sshd[11161]: pam_unix(sshd:session): session closed for user [redacted]
210 Aug 12 09:56:42 [redacted] sshd[11171]: Accepted password for [redacted] from [redacted] port 55008 ssh2
211 Aug 12 09:56:42 [redacted] sshd[11173]: pam_unix(sshd:session): session opened for user [redacted]d=0)
212 Aug 12 09:57:48 [redacted] sudo: [redacted] : TTY=pts/1 ; PWD=/home/msfadmin ; USER=root ; COMMAND=/usr/sbin/adduser test
```

Auth.log

- **Network Traffic (capture.pcap)**
  The .pcap file was generated using **tcpdump** to capture network traffic during simulated SSH login attempts, including both successful and failed authentications. Additionally, tools such as **Metasploit** were used to create anomalous traffic patterns (e.g., repeated failed logins) to mimic malicious activity. This ensured that the dataset included both benign and suspicious packets for training and evaluation.


capture.pcap
File Edit View Go Capture Analyze Statistics Telephony Wireless Tools Help
Apply a display filter … <Ctrl-/>

```
No.  Time      Source            Destination       Protocol Length Info
   1 0.000000  VMware_0f:9b:c6   Broadcast         ARP        42 Wh[redacted]? Tell [redacted]
   2 0.003467  VMware_47:71:f2   VMware_0f:9b:c6   ARP        60 19[redacted]2 is at 00:0c:2[redacted]
   3 0.003486  192[redacted]8    19[redacted]02    TCP        74 43[redacted] [SYN] Seq=0 Win=64[redacted]60 SACK_PERM TSval=18[redacted]=0 W…
   4 0.004957  192[redacted]2    19[redacted]28    TCP        74 22[redacted] [SYN, ACK] Seq=0 A[redacted]n=0 MSS=1460 SACK_PERM [redacted]9 TS…
   5 0.006895  192[redacted]8    19[redacted]02    TCP        66 43[redacted] [ACK] Seq=1 Ack=1 W[redacted] TSval=1[redacted]820 TSecr=[redacted]549
   6 0.009013  192[redacted]8    19[redacted]02    SSHv2      99 Cl[redacted]rotocol (SSH-2.0-Ope[redacted]an-7)
   7 0.010621  192[redacted]2    19[redacted]28    TCP        66 22[redacted] [ACK] Seq=1 Ack=34 [redacted] TSval=5[redacted] TSecr=18[redacted]
   8 0.056900  192[redacted]2    19[redacted]28    SSHv2     104 Se[redacted]rotocol (SSH-2.0-OpenSSH_4.7p1 Debian-8ubuntu1)
   9 0.057044  192[redacted]8    19[redacted]02    TCP        66 43[redacted] [ACK] Seq=34 Ack=39 Win=64256 Len=0 TSval=18[redacted] TSecr=[redacted]
  10 0.060084  192[redacted]2    19[redacted]28    SSHv2     850 Se[redacted]ey Exchange Init
  11 0.060170  192[redacted]8    19[redacted]02    TCP        66 43[redacted] [ACK] Seq=34 Ack=823 Win=[redacted]8 Len=0 TSval=1[redacted]3 TSecr=[redacted]
  12 0.062294  192[redacted]8    19[redacted]02    SSHv2    1642 Cl[redacted]ey Exchange Init
  13 0.065266  192[redacted]2    19[redacted]28    TCP        66 22[redacted] [ACK] Seq=823 Ack=1482 Win=8704 Len=0 TSval=[redacted] TSecr=1[redacted]
  14 0.065273  192[redacted]2    19[redacted]28    TCP        66 22[redacted] [ACK] Seq=823 Ack=1610 Win=11584 Len=0 TSval[redacted] TSecr=[redacted]
  15 0.065314  192[redacted]8    19[redacted]02    SSHv2      90 Cl[redacted]ffie-Hellman Group Exchange Request
  16 0.066297  192[redacted]2    19[redacted]28    TCP        66 22[redacted] [ACK] Seq=823 Ack=1634 Win=11584 Len=0 TSval[redacted] TSecr=[redacted]
  17 0.086981  192[redacted]2    19[redacted]28    SSHv2     474 Se[redacted]ffie-Hellman Group Exchange Group
  18 0.092590  192[redacted]8    19[redacted]02    SSHv2     466 Cl[redacted]ffie-Hellman Group Exchange Init
  19 0.123364  192[redacted]2    19[redacted]28    SSHv2    1042 Se[redacted]ffie-Hellman Group Exchange Reply, New Keys

› Frame 1: 42 bytes on wire (336 bits), 42 bytes captured (336 bits)
› Ethernet II, Src: VMware_0f:9b:c6 (00:0c:29:0f:9b:c6), Dst: Broadcast (ff:ff:ff:ff
› Address Resolution Protocol (request)

0000  ff ff ff ff ff ff 00 0c  29 0f 9b c6 08 06 00 01   ........ ).......
0010  08 00 06 04 00 01 00 0c  29 0f 9b c6 c0 a8 94 80   ........ ).......
0020  00 00 00 00 00 00 c0 a8  94 66                     ......... .f
```

Capture.pcap

By combining logs and packet captures, the project created a dataset that reflects both host-level and network-level perspectives, providing a more complete view for anomaly detection.

Screenshots of the collected raw data are included in the documentation to illustrate the inputs prior to preprocessing.

3.3 Preprocessing

To transform the raw data into a structured and machine-learning–ready format, two dedicated preprocessing scripts were developed:

- **System Log Preprocessing (sys-log.py)**
  The raw auth.log file contained complete log entries with timestamps, process details, and messages. The preprocessing script extracted only the most relevant fields:
    - **Timestamp** — the date and time of the event.
    - **Message** — the actual log message (e.g., *"Failed password for invalid user admin"*).

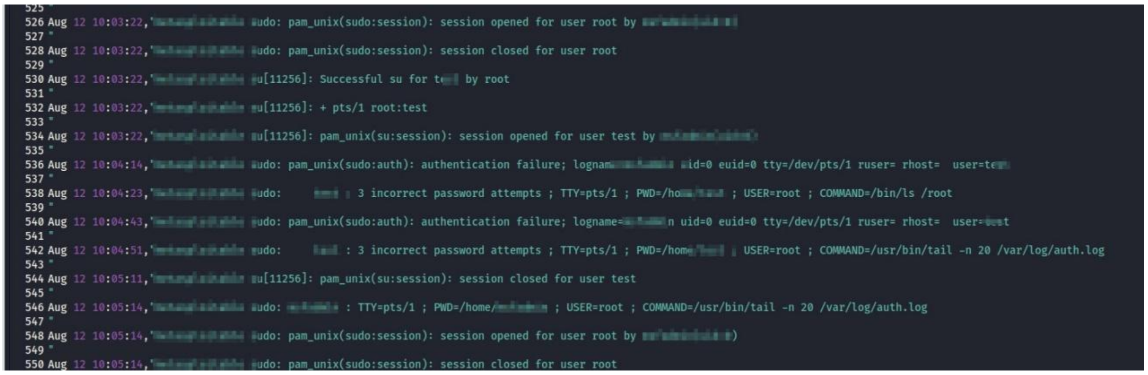
This reduced noise and preserved the essential information needed for anomaly classification.

- **Network Traffic Preprocessing (data-proc.py)**
  The raw .pcap file contained full packet-level data. Using the **Scapy** library, the script parsed each packet and extracted key features:
    - **Source IP** — the origin address of the packet.
    - **Destination IP** — the receiving address.
    - **Protocol** — the network protocol used (e.g., TCP, UDP).
    - **Length** — the size of the packet in bytes.
    - **Timestamp** — the time the packet was captured.

These features capture both communication patterns and contextual details relevant for detecting suspicious activity.

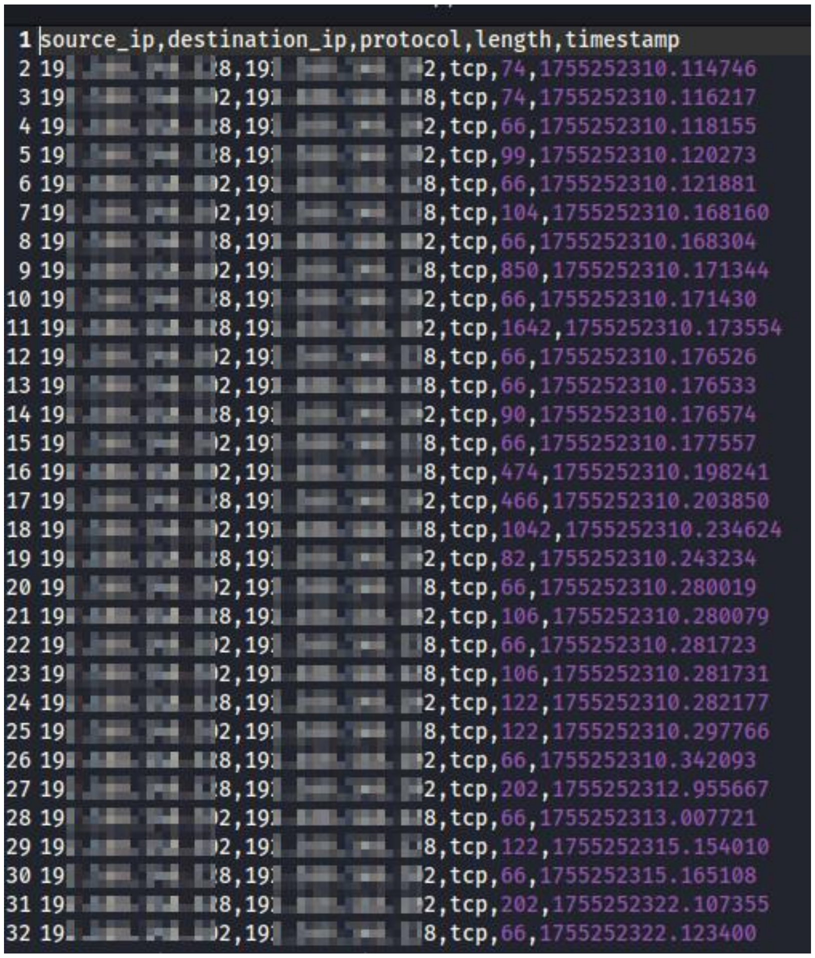
- **Clean CSV Outputs**
  Both scripts stored the extracted data as structured CSV files:
    - preprocessed_logs.csv (for system logs)
    - preprocessed_data.csv (for network traffic)

These files provided a clean and uniform format, suitable for tokenization and model training.



```
"
Aug 12 10:03:22,"[redacted] sudo: pam_unix(sudo:session): session opened for user root by [redacted]
"
Aug 12 10:03:22,"[redacted] sudo: pam_unix(sudo:session): session closed for user root
"
Aug 12 10:03:22,"[redacted] su[11256]: Successful su for t[redacted] by root
"
Aug 12 10:03:22,"[redacted] su[11256]: + pts/1 root:test
"
Aug 12 10:03:22,"[redacted] su[11256]: pam_unix(su:session): session opened for user test by [redacted]
"
Aug 12 10:04:14,"[redacted] sudo: pam_unix(sudo:auth): authentication failure; lognam[redacted] uid=0 euid=0 tty=/dev/pts/1 ruser= rhost=  user=te[redacted]
"
Aug 12 10:04:23,"[redacted] sudo:     [redacted] : 3 incorrect password attempts ; TTY=pts/1 ; PWD=/ho[redacted] ; USER=root ; COMMAND=/bin/ls /root
"
Aug 12 10:04:43,"[redacted] sudo: pam_unix(sudo:auth): authentication failure; logname=[redacted]n uid=0 euid=0 tty=/dev/pts/1 ruser= rhost=  user=[redacted]t
"
Aug 12 10:04:51,"[redacted] sudo:     [redacted] : 3 incorrect password attempts ; TTY=pts/1 ; PWD=/home[redacted] USER=root ; COMMAND=/usr/bin/tail -n 20 /var/log/auth.log
"
Aug 12 10:05:11,"[redacted] su[11256]: pam_unix(su:session): session closed for user test
"
Aug 12 10:05:14,"[redacted] sudo: [redacted] : TTY=pts/1 ; PWD=/home/[redacted] ; USER=root ; COMMAND=/usr/bin/tail -n 20 /var/log/auth.log
"
Aug 12 10:05:14,"[redacted] sudo: pam_unix(sudo:session): session opened for user root by [redacted])
"
Aug 12 10:05:14,"[redacted] sudo: pam_unix(sudo:session): session closed for user root
```

preprocessed_logs.csv



```
source_ip,destination_ip,protocol,length,timestamp
19[redacted]8,19[redacted]2,tcp,74,1755252310.114746
19[redacted]2,19[redacted]8,tcp,74,1755252310.116217
19[redacted]8,19[redacted]2,tcp,66,1755252310.118155
19[redacted]8,19[redacted]2,tcp,99,1755252310.120273
19[redacted]2,19[redacted]8,tcp,66,1755252310.121881
19[redacted]2,19[redacted]8,tcp,104,1755252310.168160
19[redacted]8,19[redacted]2,tcp,66,1755252310.168304
19[redacted]2,19[redacted]8,tcp,850,1755252310.171344
19[redacted]8,19[redacted]2,tcp,66,1755252310.171430
19[redacted]8,19[redacted]2,tcp,1642,1755252310.173554
19[redacted]2,19[redacted]8,tcp,66,1755252310.176526
19[redacted]2,19[redacted]8,tcp,66,1755252310.176533
19[redacted]8,19[redacted]2,tcp,90,1755252310.176574
19[redacted]2,19[redacted]8,tcp,66,1755252310.177557
19[redacted]2,19[redacted]8,tcp,474,1755252310.198241
19[redacted]8,19[redacted]2,tcp,466,1755252310.203850
19[redacted]2,19[redacted]8,tcp,1042,1755252310.234624
19[redacted]8,19[redacted]2,tcp,82,1755252310.243234
19[redacted]2,19[redacted]8,tcp,66,1755252310.280019
19[redacted]8,19[redacted]2,tcp,106,1755252310.280079
19[redacted]2,19[redacted]8,tcp,66,1755252310.281723
19[redacted]2,19[redacted]8,tcp,106,1755252310.281731
19[redacted]8,19[redacted]2,tcp,122,1755252310.282177
19[redacted]2,19[redacted]8,tcp,122,1755252310.297766
19[redacted]8,19[redacted]2,tcp,66,1755252310.342093
19[redacted]8,19[redacted]2,tcp,202,1755252312.955667
19[redacted]2,19[redacted]8,tcp,66,1755252313.007721
19[redacted]2,19[redacted]8,tcp,122,1755252315.154010
19[redacted]8,19[redacted]2,tcp,66,1755252315.165108
19[redacted]8,19[redacted]2,tcp,202,1755252322.107355
19[redacted]2,19[redacted]8,tcp,66,1755252322.123400
```

preprocessed_data.csv

Screenshots of the CSV samples were included in the documentation to visually confirm the successful transformation of raw inputs into clean, structured datasets.

3.4 Tokenization

Before training the model, the preprocessed data needed to be converted into a numerical format that a neural network can understand. This step is handled through **tokenization**, which transforms raw text into vectorized representations. For this project, the **BERT tokenizer** (bert-base-uncased) was used, ensuring compatibility with the pre-trained BERT model selected for fine-tuning.

Three dedicated scripts were implemented to handle tokenization:

- **log-tok.py** — tokenizes the messages extracted from preprocessed_logs.csv.
- **net-tok.py** — tokenizes the communication features (source IP, destination IP, protocol) extracted from preprocessed_data.csv.
- **net-log-tok.py** — combines both sources, creating unified representations that include host-level log data and network-level packet data.

The importance of tokenization lies in bridging the gap between human-readable text and machine learning models. Raw logs and network fields are strings, which cannot be processed directly by BERT. Through tokenization, each input is broken into tokens and mapped to unique numerical IDs from BERT's vocabulary. The tokenizer also generates **attention masks**, which guide the model in distinguishing meaningful tokens from padding.

By applying the BERT tokenizer, the system ensured that all data — whether short log messages or structured network features — was consistently encoded into vectorized form. This created a uniform input space for fine-tuning the anomaly detection model.

3.5 Model Training

The core of the anomaly detection system is the fine-tuning of a **BERT model** on the preprocessed and tokenized datasets. The training process was implemented in the script **model-trainer.py**, which integrates data preparation, labeling, model configuration, training, and evaluation.

- **Anomaly Labeling**
  Since no labeled dataset was available, anomaly labels were generated using **regular expression (regex) patterns** applied to log messages. These patterns were designed to capture events commonly associated with malicious or suspicious activity, such as:
  - *“failed password”*
  - *“invalid user”*
  - *“authentication failure”*
  - *“nmap”*
  - *“port scan”*

Entries matching these expressions were labeled as **anomalies (1)**, while all other entries were considered **normal (0)**.

- **Balancing the Dataset**
  In typical SOC data, normal events significantly outnumber anomalies. To prevent the model from being biased toward normal behavior, the dataset was balanced by capping the number of normal samples while keeping all anomalous samples. This ensured more meaningful training without overwhelming the anomaly class.
- **Fine-Tuning BERT**
  The **bert-base-uncased** model was fine-tuned on the combined dataset of system logs and network traffic. The following hyperparameters were used:
  - **Learning rate (LR):** 2e-5
  - **Batch size:** 16 (training)
  - **Epochs:** 2 (optimized for time and resources)
  - **Maximum sequence length:** 128 tokens

These settings provided a balance between training efficiency and model performance, especially when running on CPU resources.

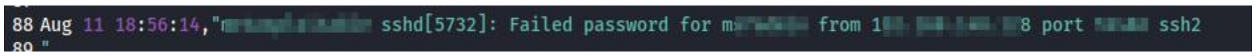
- **Model Saving and Metrics**
  After training, the fine-tuned model and tokenizer were saved in the directory results/fine_tuned_model for later inference. Evaluation metrics — including accuracy, precision, recall, and F1-score — were computed and stored in metrics_results.csv. Additionally, a bar chart summarizing the model’s performance was generated and saved as metrics_chart.png.

This training phase established the foundation of the system by creating a specialized BERT model capable of distinguishing between normal and anomalous security events.

3.6 Inference

Once the model was fine-tuned, the next step was to test its ability to classify new security events. This was implemented in **Anomaly-detection.py**, which provides a simple yet effective inference pipeline.

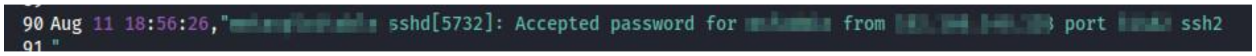
- **Model and Tokenizer Loading**
  The script loads the fine-tuned BERT model and its corresponding tokenizer from the results/fine_tuned_model directory. This ensures consistency between training and inference.
- **Decision Threshold**
  Predictions are returned as probability scores for the two classes: **Normal** and **Anomaly**. By default, a decision threshold of **0.5** was applied:
  - If the anomaly score ≥ 0.5 → the event is classified as **Anomaly**.
  - If the anomaly score < 0.5 → the event is classified as **Normal**.
    This threshold can be adjusted if needed to optimize sensitivity or specificity depending on the use case.
- **Example Outputs**
  The pipeline was tested on simulated SSH login events:
  - **Failed SSH login**



```
Aug 11 18:56:14,"[redacted] sshd[5732]: Failed password for m[redacted] from 1[redacted]8 port [redacted] ssh2
"
```

→ Classified as **Anomaly**.

  - **Successful SSH login**



```
Aug 11 18:56:26,"[redacted] sshd[5732]: Accepted password for [redacted] from [redacted] port [redacted] ssh2
"
```

→ Classified as **Normal**.

This inference step validated that the fine-tuned model could correctly distinguish between benign and suspicious events, making it suitable for practical anomaly detection scenarios.

3.7 Automation

To streamline the workflow, the entire anomaly detection process was automated using a shell script named **run.sh**. This script sequentially executes all the necessary components of the pipeline, ensuring a smooth transition from raw data to final anomaly classification without manual intervention.

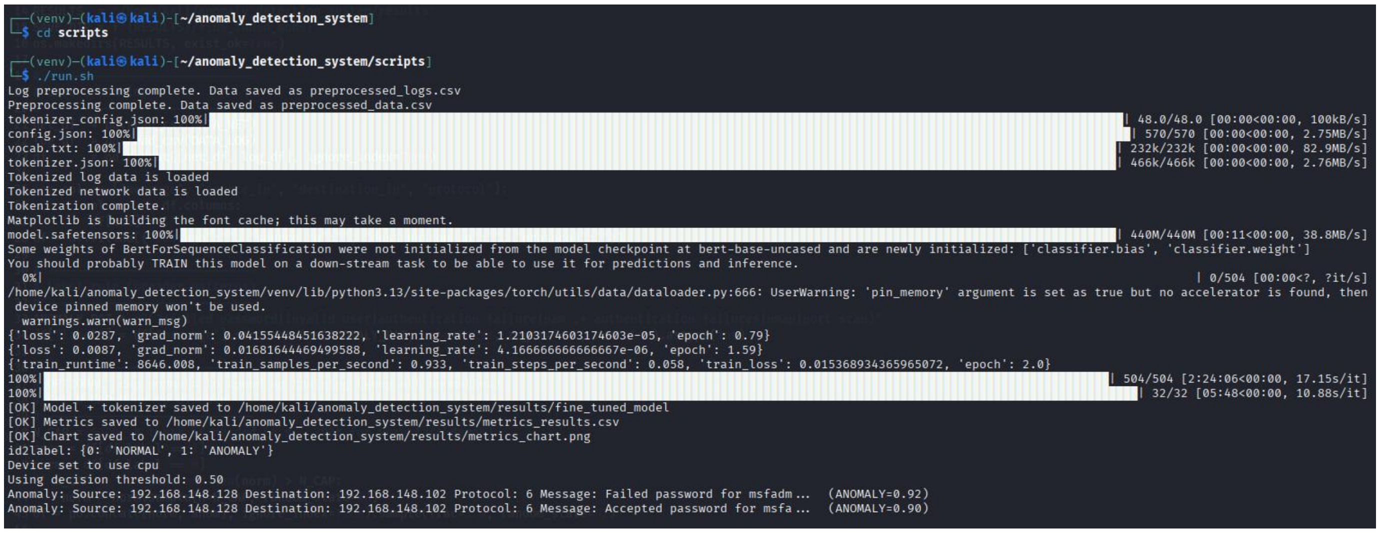
The automation covers the following steps:

1. **Preprocessing system logs** (sys-log.py) → generates preprocessed_logs.csv.
2. **Preprocessing network traffic** (data-proc.py) → generates preprocessed_data.csv.
3. **Tokenizing log data** (log-tok.py).
4. **Tokenizing network data** (net-tok.py).
5. **Tokenizing combined data** (net-log-tok.py).
6. **Model training** (model-trainer.py) → fine-tunes BERT, saves model, metrics, and performance chart.
7. **Inference** (Anomaly-detection.py) → classifies sample events as **Normal** or **Anomaly**.

Simplified workflow executed by run.sh:

Raw logs & pcap → Preprocessing (CSV) → Tokenization → Model Training → Inference

This automation ensured that the project could be reproduced end-to-end with a single command, significantly improving usability and making the system demonstration-ready.



```
┌──(venv)─(kali㉿kali)-[~/anomaly_detection_system]
└─$ cd scripts

┌──(venv)─(kali㉿kali)-[~/anomaly_detection_system/scripts]
└─$ ./run.sh
Log preprocessing complete. Data saved as preprocessed_logs.csv
Preprocessing complete. Data saved as preprocessed_data.csv
tokenizer_config.json: 100%|██████████████████████████████| 48.0/48.0 [00:00<00:00, 100kB/s]
config.json: 100%|████████████████████████████████████████| 570/570 [00:00<00:00, 2.75MB/s]
vocab.txt: 100%|██████████████████████████████████████████| 232k/232k [00:00<00:00, 82.9MB/s]
tokenizer.json: 100%|█████████████████████████████████████| 466k/466k [00:00<00:00, 2.76MB/s]
Tokenized log data is loaded
Tokenized network data is loaded
Tokenization complete.
Matplotlib is building the font cache; this may take a moment.
model.safetensors: 100%|██████████████████████████████████| 440M/440M [00:11<00:00, 38.8MB/s]
Some weights of BertForSequenceClassification were not initialized from the model checkpoint at bert-base-uncased and are newly initialized: ['classifier.bias', 'classifier.weight']
You should probably TRAIN this model on a down-stream task to be able to use it for predictions and inference.
  0%|                                                    | 0/504 [00:00<?, ?it/s]
/home/kali/anomaly_detection_system/venv/lib/python3.13/site-packages/torch/utils/data/dataloader.py:666: UserWarning: 'pin_memory' argument is set as true but no accelerator is found, then
 device pinned memory won't be used.
  warnings.warn(warn_msg)
{'loss': 0.0287, 'grad_norm': 0.04155448451638222, 'learning_rate': 1.2103174603174603e-05, 'epoch': 0.79}
{'loss': 0.0087, 'grad_norm': 0.01681644469499588, 'learning_rate': 4.166666666666667e-06, 'epoch': 1.59}
{'train_runtime': 8646.008, 'train_samples_per_second': 0.933, 'train_steps_per_second': 0.058, 'train_loss': 0.015368934365965072, 'epoch': 2.0}
100%|█████████████████████████████████████████████████████| 504/504 [2:24:06<00:00, 17.15s/it]
100%|█████████████████████████████████████████████████████| 32/32 [05:48<00:00, 10.88s/it]
[OK] Model + tokenizer saved to /home/kali/anomaly_detection_system/results/fine_tuned_model
[OK] Metrics saved to /home/kali/anomaly_detection_system/results/metrics_results.csv
[OK] Chart saved to /home/kali/anomaly_detection_system/results/metrics_chart.png
id2label: {0: 'NORMAL', 1: 'ANOMALY'}
Device set to use cpu
Using decision threshold: 0.50
Anomaly: Source: 192.168.148.128 Destination: 192.168.148.102 Protocol: 6 Message: Failed password for msfadm ...  (ANOMALY=0.92)
Anomaly: Source: 192.168.148.128 Destination: 192.168.148.102 Protocol: 6 Message: Accepted password for msfa ...  (ANOMALY=0.90)
```

Run.sh result

4. Results

The performance of the anomaly detection system was evaluated after fine-tuning the BERT model on the preprocessed datasets. The results include both quantitative metrics and qualitative examples.

- **Performance Metrics**
  A bar chart was generated to summarize the key evaluation metrics: **accuracy, precision, recall, and F1-score**. These metrics were also stored in metrics_results.csv for reference.
  - **Accuracy**: The model achieved high accuracy, reflecting its ability to correctly classify most events.
  - **Precision & Recall**: These values were lower compared to accuracy, due to the use of regex-based labeling. Since anomalies were defined by specific keyword patterns, any events outside those patterns were not labeled, limiting the diversity of the anomaly class.
  - **F1-Score**: This provided a balanced measure, highlighting the trade-off between precision and recall.

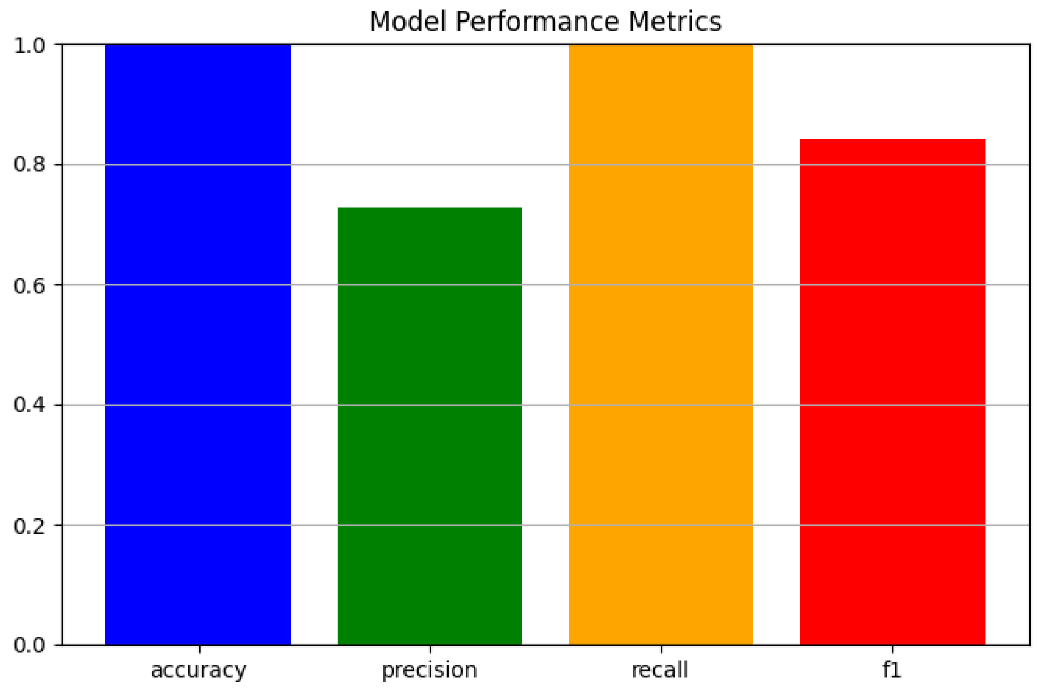


Figure 1: Model performance chart (accuracy, precision, recall, F1).

- **Example Outputs**
  The inference script produced clear classifications for simulated events:

    - Failed SSH login → **Anomaly**
    - Successful SSH login → **Normal**

These outputs confirmed that the model could distinguish between benign and suspicious events, even in a simplified setup.

- **Terminal Run**
  A screenshot of the terminal execution has been included to illustrate the automated pipeline in action, showing preprocessing, training, and anomaly detection results in sequence.
- **Discussion of Results**
  Overall, the system demonstrated:
    - **High accuracy**, validating that the model successfully learned from the available dataset.
    - **Limited precision and recall**, constrained by the regex-based anomaly labeling approach.
    - **Practical feasibility**, proving that Large Language Models such as BERT can be adapted for anomaly detection in cybersecurity.

These results establish the project as a solid proof of concept and an important first step toward applying LLMs for autonomous security response.

5. Discussion

The development of this anomaly detection system provided valuable insights into the application of Large Language Models (LLMs) within cybersecurity. The project's strengths, limitations, and lessons learned are summarized below.

- **Strengths**
    - **End-to-end working system**: The project successfully integrated data collection, preprocessing, tokenization, model training, and inference into a cohesive workflow.
    - **Automated pipeline**: With the run.sh script, the entire process could be executed with a single command, ensuring reproducibility and ease of demonstration.
    - **First successful use of LLM for anomaly detection**: This internship project marked the first time I applied an LLM (BERT) in practice, achieving meaningful results in classifying security events.
- **Limitations**
    - **Synthetic dataset with regex-based labels**: The anomalies were defined using keyword patterns, which restricted the diversity of the anomaly class and introduced labeling biases.
    - **Need for real-world data**: To transition this proof of concept into a production-ready solution, larger and more representative datasets with ground-truth labels would be essential.
- **Lessons Learned**
    - **Importance of preprocessing**: Clean, structured data proved to be fundamental for achieving reliable model performance.
    - **Adapting LLMs to cybersecurity**: This project demonstrated how pre-trained language models can be fine-tuned to understand and classify log and network data.
    - **Foundation for future work**: The system served as the first step toward more advanced projects, particularly my later work on **LLM-driven autonomous incident response**, where the goal is to not only detect anomalies but also suggest or execute remediation actions.

This discussion highlights how the project, despite its constraints, succeeded in validating the feasibility of applying LLMs to cybersecurity anomaly detection and in laying the groundwork for more advanced research and practical applications.

6. Conclusion

This project successfully demonstrated the application of Large Language Models to cybersecurity anomaly detection, delivering a complete proof of concept. The main achievements can be summarized as follows:

- **Built an anomaly detection system** capable of classifying normal and anomalous events from system logs and network traffic.
- **Preprocessed logs and network data** into structured, machine-learning–ready formats.
- **Fine-tuned BERT** (bert-base-uncased) to adapt a general-purpose LLM to the domain of cybersecurity.
- **Developed an automated pipeline** that integrates data collection, preprocessing, tokenization, training, and inference, while also producing visual performance results.

Beyond the technical outcomes, this project holds particular significance as it represented my **introductory experience with LLMs during my internship**. It served as the foundation for understanding how language models can be adapted to cybersecurity tasks and paved the way for more advanced applications.

Looking ahead, future work should focus on:

- Incorporating **real SOC datasets** with ground-truth labels to improve model reliability.
- Exploring **advanced LLMs**, including GPT-based architectures, for enhanced contextual understanding.
- Adding **explainability features**, enabling analysts to better interpret the model's decisions and trust its outputs.

In conclusion, this project not only validated the feasibility of using LLMs for anomaly detection but also marked an important milestone in my professional journey, bridging academic knowledge with practical, AI-driven cybersecurity solutions.

7. References

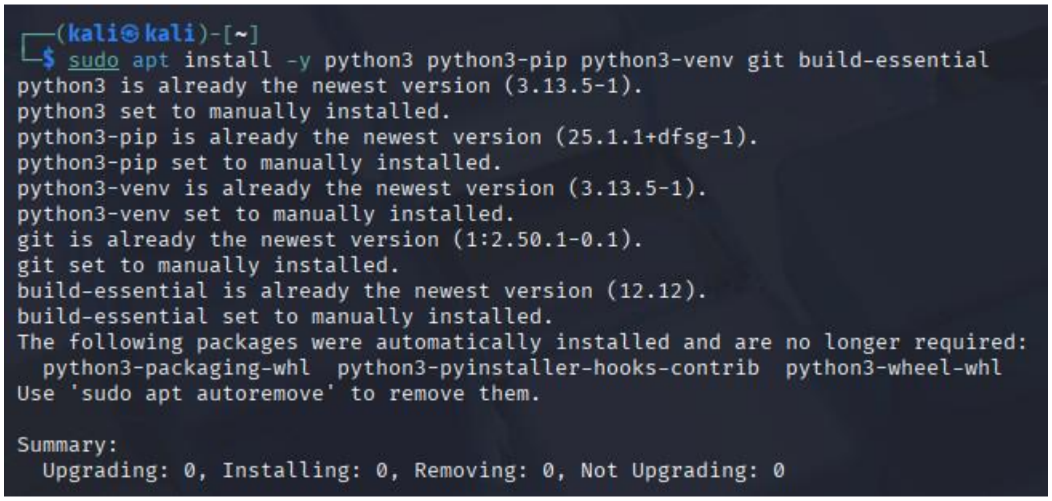
1. Hugging Face — *Transformers Library*. Available at: https://huggingface.co/transformers
2. Scapy — *Packet Manipulation and Analysis Tool*. Available at: https://scapy.net
3. Scikit-learn — *Machine Learning in Python*. Available at: https://scikit-learn.org
4. Vaswani, A., et al. (2017). *Attention Is All You Need*. Advances in Neural Information Processing Systems (NeurIPS).
5. Brownlee, J. (2020). *A Gentle Introduction to Anomaly Detection in Machine Learning*. Machine Learning Mastery.
6. Chandola, V., Banerjee, A., & Kumar, V. (2009). *Anomaly Detection: A Survey*. ACM Computing Surveys, 41(3), 1–58.

8. Appendices

The following appendices include selected screenshots, data samples, and concise code excerpts that substantiate the end-to-end implementation described in the main text. Items are organized to mirror the pipeline (setup → data → preprocessing → tokenization → training/inference → results), with cross-references to relevant sections for context. Full source code and assets are available in the project repository.

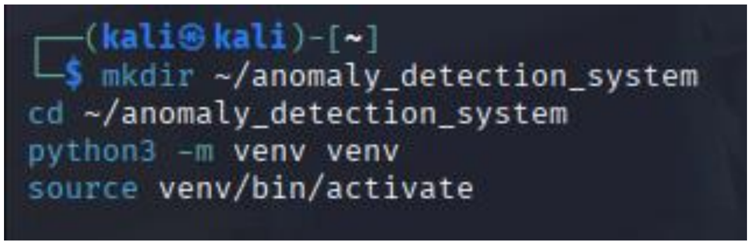
Appendix A — Environment & Setup (Screenshots)

Context: See §3.1 Environment Setup.



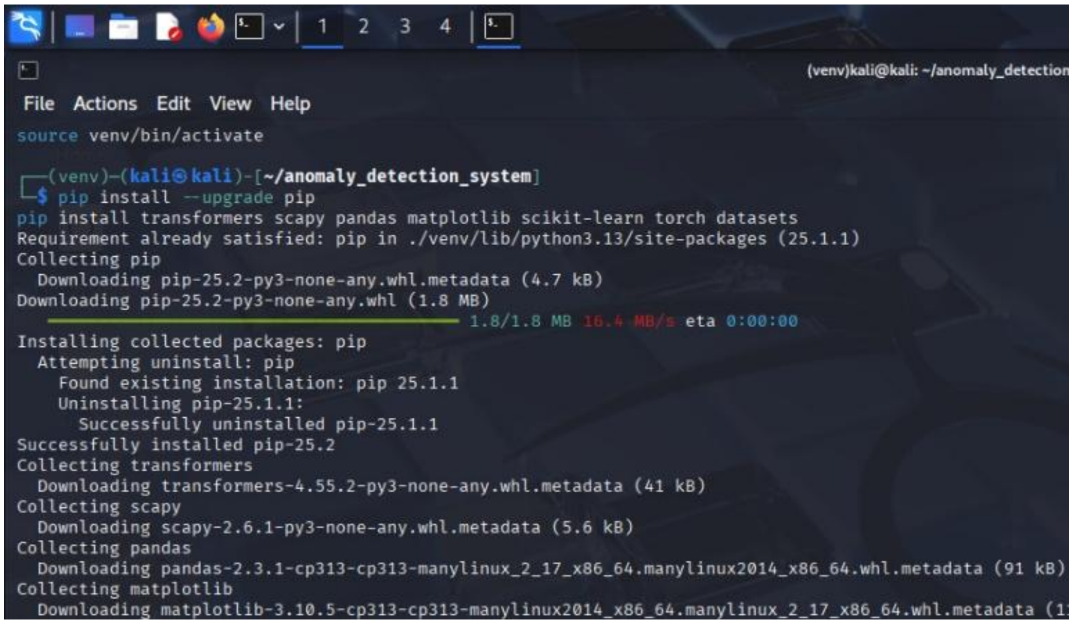
```
┌──(kali㉿kali)-[~]
└─$ sudo apt install -y python3 python3-pip python3-venv git build-essential
python3 is already the newest version (3.13.5-1).
python3 set to manually installed.
python3-pip is already the newest version (25.1.1+dfsg-1).
python3-pip set to manually installed.
python3-venv is already the newest version (3.13.5-1).
python3-venv set to manually installed.
git is already the newest version (1:2.50.1-0.1).
git set to manually installed.
build-essential is already the newest version (12.12).
build-essential set to manually installed.
The following packages were automatically installed and are no longer required:
  python3-packaging-whl  python3-pyinstaller-hooks-contrib  python3-wheel-whl
Use 'sudo apt autoremove' to remove them.

Summary:
  Upgrading: 0, Installing: 0, Removing: 0, Not Upgrading: 0
```

Figure A1. Python installation



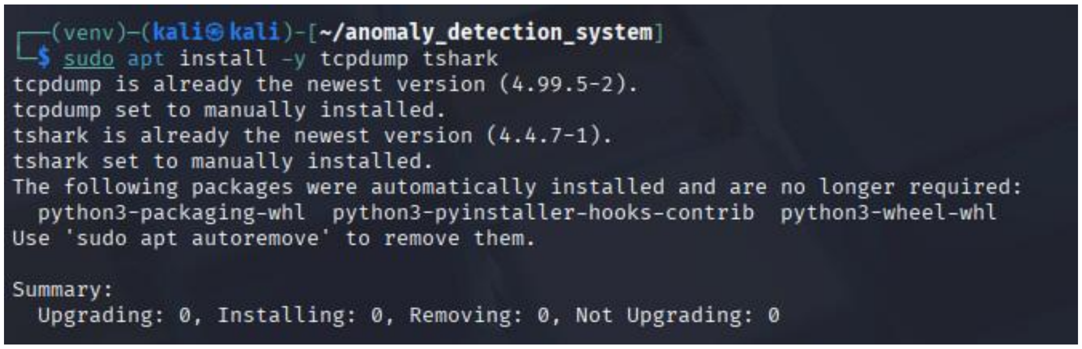
```
┌──(kali㉿kali)-[~]
└─$ mkdir ~/anomaly_detection_system
cd ~/anomaly_detection_system
python3 -m venv venv
source venv/bin/activate
```

Figure A2. Project folders & virtual environment



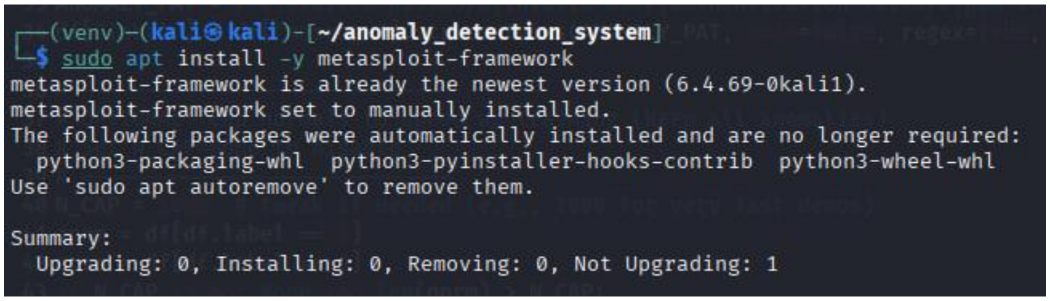
```
source venv/bin/activate

┌──(venv)─(kali㉿kali)-[~/anomaly_detection_system]
└─$ pip install --upgrade pip
pip install transformers scapy pandas matplotlib scikit-learn torch datasets
Requirement already satisfied: pip in ./venv/lib/python3.13/site-packages (25.1.1)
Collecting pip
  Downloading pip-25.2-py3-none-any.whl.metadata (4.7 kB)
Downloading pip-25.2-py3-none-any.whl (1.8 MB)
   ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 1.8/1.8 MB 16.4 MB/s eta 0:00:00
Installing collected packages: pip
  Attempting uninstall: pip
    Found existing installation: pip 25.1.1
    Uninstalling pip-25.1.1:
      Successfully uninstalled pip-25.1.1
Successfully installed pip-25.2
Collecting transformers
  Downloading transformers-4.55.2-py3-none-any.whl.metadata (41 kB)
Collecting scapy
  Downloading scapy-2.6.1-py3-none-any.whl.metadata (5.6 kB)
Collecting pandas
  Downloading pandas-2.3.1-cp313-cp313-manylinux_2_17_x86_64.manylinux2014_x86_64.whl.metadata (91 kB)
Collecting matplotlib
  Downloading matplotlib-3.10.5-cp313-cp313-manylinux2014_x86_64.manylinux_2_17_x86_64.whl.metadata (1
```

Figure A3. Library installation (Transformers, Torch, Scapy, scikit-learn, Matplotlib)



```
┌──(venv)─(kali㉿kali)-[~/anomaly_detection_system]
└─$ sudo apt install -y tcpdump tshark
tcpdump is already the newest version (4.99.5-2).
tcpdump set to manually installed.
tshark is already the newest version (4.4.7-1).
tshark set to manually installed.
The following packages were automatically installed and are no longer required:
  python3-packaging-whl  python3-pyinstaller-hooks-contrib  python3-wheel-whl
Use 'sudo apt autoremove' to remove them.

Summary:
  Upgrading: 0, Installing: 0, Removing: 0, Not Upgrading: 0
```

Figure A4. Tcpdump installation

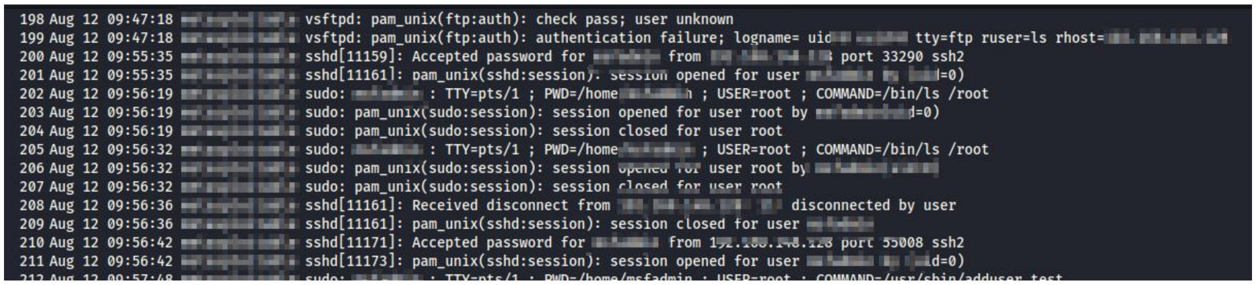


```
┌──(venv)─(kali㉿kali)-[~/anomaly_detection_system]
└─$ sudo apt install -y metasploit-framework
metasploit-framework is already the newest version (6.4.69-0kali1).
metasploit-framework set to manually installed.
The following packages were automatically installed and are no longer required:
  python3-packaging-whl  python3-pyinstaller-hooks-contrib  python3-wheel-whl
Use 'sudo apt autoremove' to remove them.

Summary:
  Upgrading: 0, Installing: 0, Removing: 0, Not Upgrading: 1
```

Figure A5. Metasploit installation

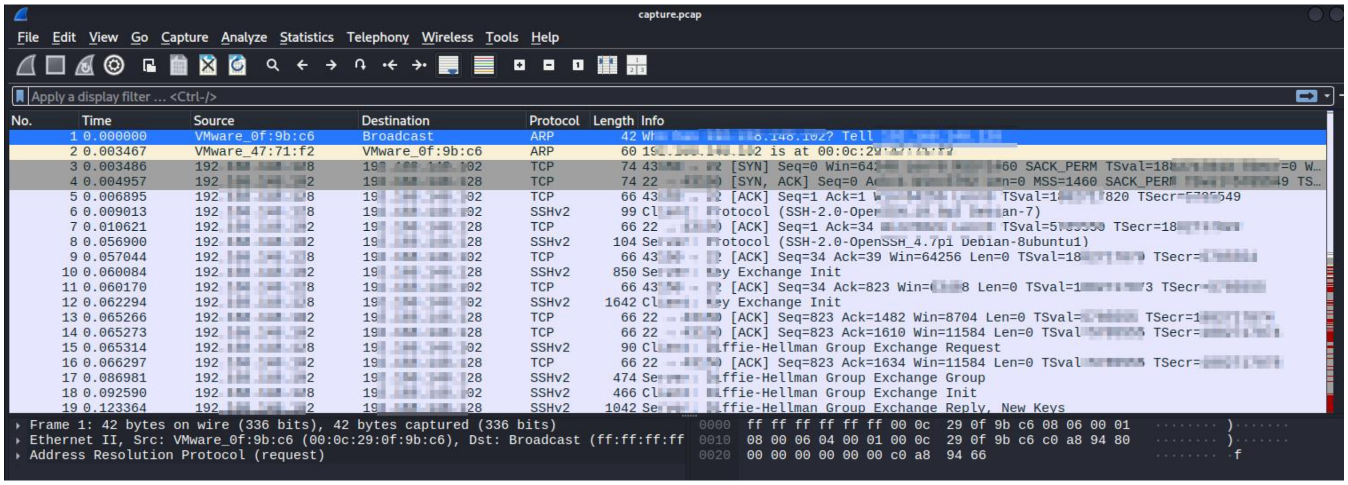
Appendix B — Raw Data Samples

Context: See §3.2 Data Collection.



```
Aug 12 09:47:18 [redacted] vsftpd: pam_unix(ftp:auth): check pass; user unknown
Aug 12 09:47:18 [redacted] vsftpd: pam_unix(ftp:auth): authentication failure; logname= uid[redacted] tty=ftp ruser=ls rhost=[redacted]
Aug 12 09:55:35 [redacted] sshd[11159]: Accepted password for [redacted] from [redacted] port 33290 ssh2
Aug 12 09:55:35 [redacted] sshd[11161]: pam_unix(sshd:session): session opened for user [redacted]=0)
Aug 12 09:56:19 [redacted] sudo: [redacted] : TTY=pts/1 ; PWD=/home[redacted] ; USER=root ; COMMAND=/bin/ls /root
Aug 12 09:56:19 [redacted] sudo: pam_unix(sudo:session): session opened for user root by [redacted]=0)
Aug 12 09:56:19 [redacted] sudo: pam_unix(sudo:session): session closed for user root
Aug 12 09:56:32 [redacted] sudo: [redacted] : TTY=pts/1 ; PWD=/home[redacted] ; USER=root ; COMMAND=/bin/ls /root
Aug 12 09:56:32 [redacted] sudo: pam_unix(sudo:session): session opened for user root by [redacted]
Aug 12 09:56:32 [redacted] sudo: pam_unix(sudo:session): session closed for user root
Aug 12 09:56:36 [redacted] sshd[11161]: Received disconnect from [redacted] disconnected by user
Aug 12 09:56:36 [redacted] sshd[11161]: pam_unix(sshd:session): session closed for user [redacted]
Aug 12 09:56:42 [redacted] sshd[11171]: Accepted password for [redacted] from 192.168.148.128 port 55008 ssh2
Aug 12 09:56:42 [redacted] sshd[11173]: pam_unix(sshd:session): session opened for user [redacted]d=0)
Aug 12 09:57:48 [redacted] sudo: [redacted] : TTY=pts/1 ; PWD=/home/msfadmin ; USER=root ; COMMAND=/usr/sbin/adduser test
```

Figure B1. auth.log



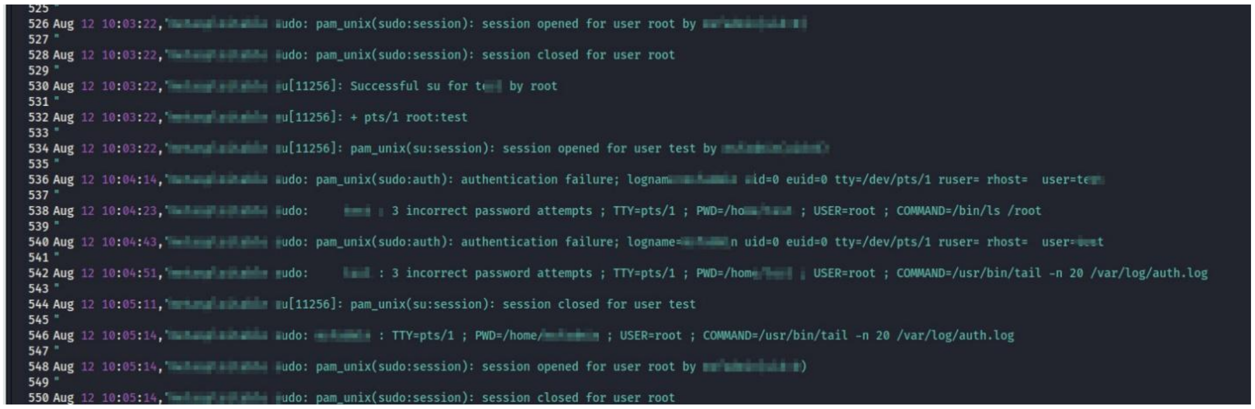
capture.pcap
File Edit View Go Capture Analyze Statistics Telephony Wireless Tools Help
Apply a display filter … <Ctrl-/>

```
No.  Time      Source             Destination        Protocol Length Info
   1 0.000000  VMware_0f:9b:c6    Broadcast          ARP        42 Wh[redacted]8.148.102? Tell [redacted]
   2 0.003467  VMware_47:71:f2    VMware_0f:9b:c6    ARP        60 19[redacted]2 is at 00:0c:29[redacted]
   3 0.003486  192[redacted]8     19[redacted]02     TCP        74 43[redacted] [SYN] Seq=0 Win=64[redacted]60 SACK_PERM TSval=18[redacted]=0 W…
   4 0.004957  192[redacted]2     19[redacted]28     TCP        74 22[redacted] [SYN, ACK] Seq=0 A[redacted]n=0 MSS=1460 SACK_PERM [redacted]9 TS…
   5 0.006895  192[redacted]8     19[redacted]02     TCP        66 43[redacted] [ACK] Seq=1 Ack=1 [redacted] TSval=1[redacted]820 TSecr=[redacted]549
   6 0.009013  192[redacted]8     19[redacted]02     SSHv2      99 Cl[redacted]rotocol (SSH-2.0-Ope[redacted]an-7)
   7 0.010621  192[redacted]2     19[redacted]28     TCP        66 22[redacted] [ACK] Seq=1 Ack=34 [redacted] TSval=5[redacted] TSecr=18[redacted]
   8 0.056900  192[redacted]2     19[redacted]28     SSHv2     104 Se[redacted]rotocol (SSH-2.0-OpenSSH_4.7p1 Debian-8ubuntu1)
   9 0.057044  192[redacted]8     19[redacted]02     TCP        66 43[redacted] [ACK] Seq=34 Ack=39 Win=64256 Len=0 TSval=18[redacted] TSecr=[redacted]
  10 0.060084  192[redacted]2     19[redacted]28     SSHv2     850 Se[redacted]ey Exchange Init
  11 0.060170  192[redacted]8     19[redacted]02     TCP        66 43[redacted] [ACK] Seq=34 Ack=823 Win=[redacted]8 Len=0 TSval=1[redacted]3 TSecr=[redacted]
  12 0.062294  192[redacted]8     19[redacted]02     SSHv2    1642 Cl[redacted]ey Exchange Init
  13 0.065266  192[redacted]2     19[redacted]28     TCP        66 22[redacted] [ACK] Seq=823 Ack=1482 Win=8704 Len=0 TSval=[redacted] TSecr=1[redacted]
  14 0.065273  192[redacted]2     19[redacted]28     TCP        66 22[redacted] [ACK] Seq=823 Ack=1610 Win=11584 Len=0 TSval[redacted] TSecr=[redacted]
  15 0.065314  192[redacted]8     19[redacted]02     SSHv2      90 Cl[redacted]ffie-Hellman Group Exchange Request
  16 0.066297  192[redacted]2     19[redacted]28     TCP        66 22[redacted] [ACK] Seq=823 Ack=1634 Win=11584 Len=0 TSval[redacted] TSecr=[redacted]
  17 0.086981  192[redacted]2     19[redacted]28     SSHv2     474 Se[redacted]ffie-Hellman Group Exchange Group
  18 0.092590  192[redacted]8     19[redacted]02     SSHv2     466 Cl[redacted]ffie-Hellman Group Exchange Init
  19 0.123364  192[redacted]2     19[redacted]28     SSHv2    1042 Se[redacted]ffie-Hellman Group Exchange Reply, New Keys

› Frame 1: 42 bytes on wire (336 bits), 42 bytes captured (336 bits)
› Ethernet II, Src: VMware_0f:9b:c6 (00:0c:29:0f:9b:c6), Dst: Broadcast (ff:ff:ff:ff
› Address Resolution Protocol (request)

0000  ff ff ff ff ff ff 00 0c  29 0f 9b c6 08 06 00 01   ········ )·······
0010  08 00 06 04 00 01 00 0c  29 0f 9b c6 c0 a8 94 80   ········ )·······
0020  00 00 00 00 00 00 c0 a8  94 66                     ········ ·f
```

Figure B2. capture.pcap overview

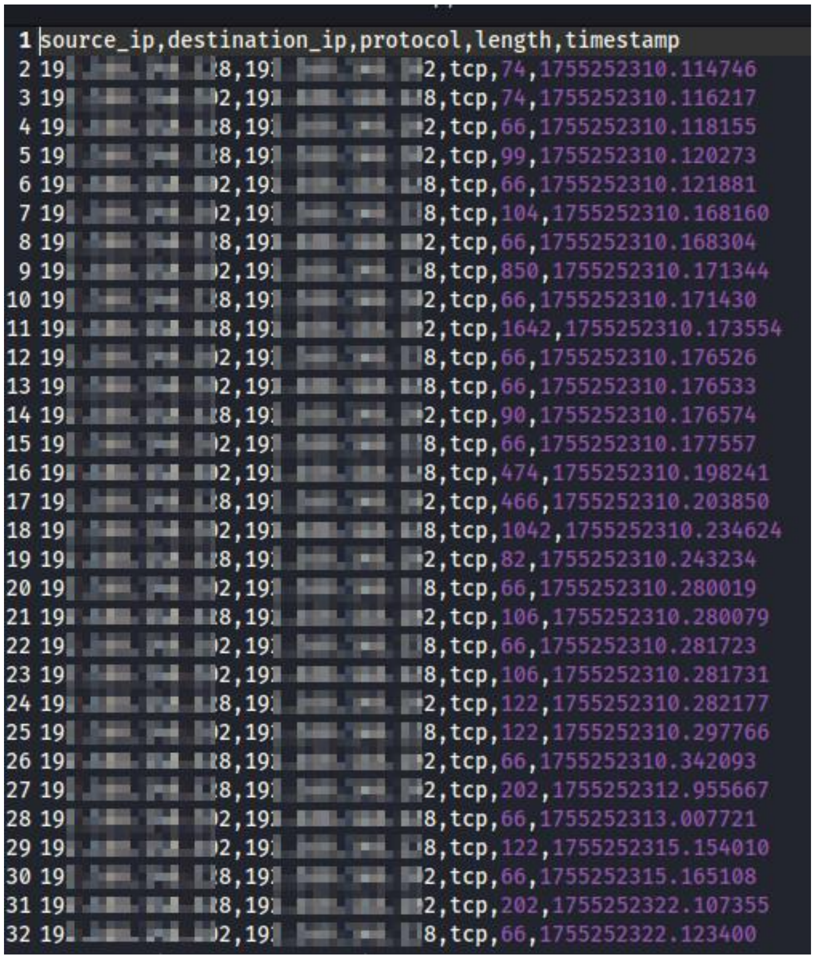
Appendix C — Preprocessed CSV Samples

Context: See §3.3 Preprocessing.



```
"
Aug 12 10:03:22,'[redacted] sudo: pam_unix(sudo:session): session opened for user root by [redacted]
"
Aug 12 10:03:22,'[redacted] sudo: pam_unix(sudo:session): session closed for user root
"
Aug 12 10:03:22,'[redacted] su[11256]: Successful su for te[redacted] by root
"
Aug 12 10:03:22,'[redacted] su[11256]: + pts/1 root:test
"
Aug 12 10:03:22,'[redacted] su[11256]: pam_unix(su:session): session opened for user test by [redacted]
"
Aug 12 10:04:14,'[redacted] sudo: pam_unix(sudo:auth): authentication failure; lognam[redacted]id=0 euid=0 tty=/dev/pts/1 ruser= rhost=  user=te[redacted]
"
Aug 12 10:04:23,'[redacted] sudo:     [redacted] 3 incorrect password attempts ; TTY=pts/1 ; PWD=/ho[redacted] ; USER=root ; COMMAND=/bin/ls /root
"
Aug 12 10:04:43,'[redacted] sudo: pam_unix(sudo:auth): authentication failure; logname=[redacted]n uid=0 euid=0 tty=/dev/pts/1 ruser= rhost=  user=[redacted]t
"
Aug 12 10:04:51,'[redacted] sudo:     [redacted] : 3 incorrect password attempts ; TTY=pts/1 ; PWD=/home[redacted] USER=root ; COMMAND=/usr/bin/tail -n 20 /var/log/auth.log
"
Aug 12 10:05:11,'[redacted] su[11256]: pam_unix(su:session): session closed for user test
"
Aug 12 10:05:14,'[redacted] sudo: [redacted] : TTY=pts/1 ; PWD=/home/[redacted] ; USER=root ; COMMAND=/usr/bin/tail -n 20 /var/log/auth.log
"
Aug 12 10:05:14,'[redacted] sudo: pam_unix(sudo:session): session opened for user root by [redacted])
"
Aug 12 10:05:14,'[redacted] sudo: pam_unix(sudo:session): session closed for user root
```

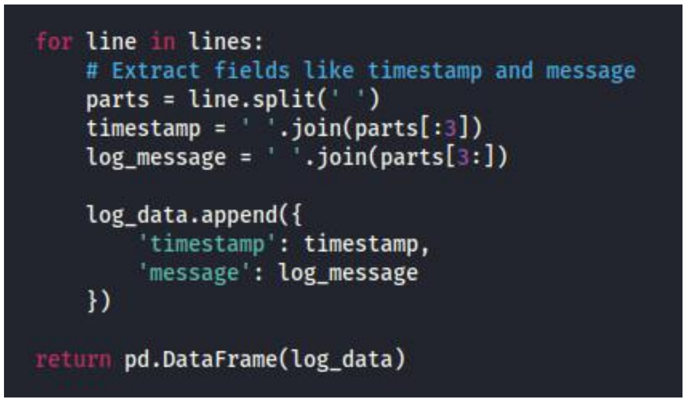
Figure C1. preprocessed_logs.csv — first 20 rows (columns: timestamp, message).



```
source_ip,destination_ip,protocol,length,timestamp
19[redacted]8,19[redacted]2,tcp,74,1755252310.114746
19[redacted]2,19[redacted]8,tcp,74,1755252310.116217
19[redacted]8,19[redacted]2,tcp,66,1755252310.118155
19[redacted]8,19[redacted]2,tcp,99,1755252310.120273
19[redacted]2,19[redacted]8,tcp,66,1755252310.121881
19[redacted]2,19[redacted]8,tcp,104,1755252310.168160
19[redacted]8,19[redacted]2,tcp,66,1755252310.168304
19[redacted]2,19[redacted]8,tcp,850,1755252310.171344
19[redacted]8,19[redacted]2,tcp,66,1755252310.171430
19[redacted]8,19[redacted]2,tcp,1642,1755252310.173554
19[redacted]2,19[redacted]8,tcp,66,1755252310.176526
19[redacted]2,19[redacted]8,tcp,66,1755252310.176533
19[redacted]8,19[redacted]2,tcp,90,1755252310.176574
19[redacted]2,19[redacted]8,tcp,66,1755252310.177557
19[redacted]2,19[redacted]8,tcp,474,1755252310.198241
19[redacted]8,19[redacted]2,tcp,466,1755252310.203850
19[redacted]2,19[redacted]8,tcp,1042,1755252310.234624
19[redacted]8,19[redacted]2,tcp,82,1755252310.243234
19[redacted]2,19[redacted]8,tcp,66,1755252310.280019
19[redacted]8,19[redacted]2,tcp,106,1755252310.280079
19[redacted]2,19[redacted]8,tcp,66,1755252310.281723
19[redacted]2,19[redacted]8,tcp,106,1755252310.281731
19[redacted]8,19[redacted]2,tcp,122,1755252310.282177
19[redacted]2,19[redacted]8,tcp,122,1755252310.297766
19[redacted]8,19[redacted]2,tcp,66,1755252310.342093
19[redacted]8,19[redacted]2,tcp,202,1755252312.955667
19[redacted]2,19[redacted]8,tcp,66,1755252313.007721
19[redacted]2,19[redacted]8,tcp,122,1755252315.154010
19[redacted]8,19[redacted]2,tcp,66,1755252315.165108
19[redacted]8,19[redacted]2,tcp,202,1755252322.107355
19[redacted]2,19[redacted]8,tcp,66,1755252322.123400
```

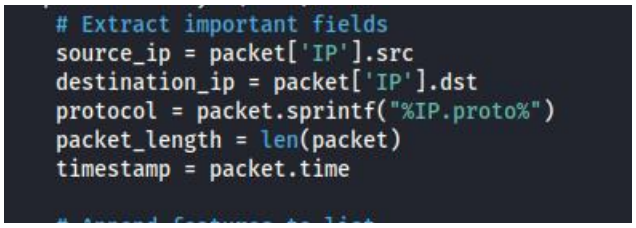
Figure C2. preprocessed_data.csv — first 20 rows (columns: source_ip, destination_ip, protocol, length, timestamp).

Appendix D — Key Code Snippets (Short Excerpts)



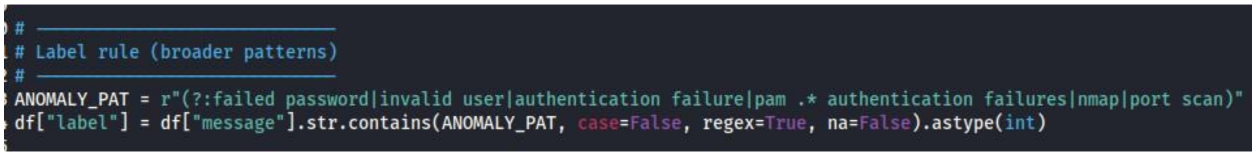
```python
for line in lines:
    # Extract fields like timestamp and message
    parts = line.split(' ')
    timestamp = ' '.join(parts[:3])
    log_message = ' '.join(parts[3:])

    log_data.append({
        'timestamp': timestamp,
        'message': log_message
    })

return pd.DataFrame(log_data)
```

Figure D1. sys-log.py — parse timestamp & message (See §3.3).



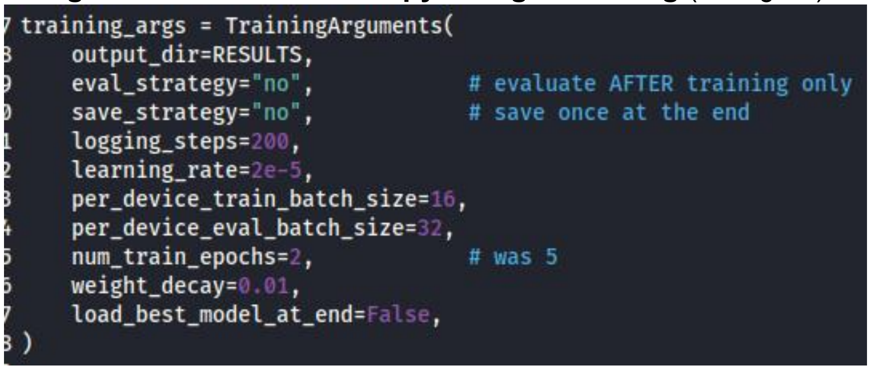
```
    # Extract important fields
    source_ip = packet['IP'].src
    destination_ip = packet['IP'].dst
    protocol = packet.sprintf("%IP.proto%")
    packet_length = len(packet)
    timestamp = packet.time
```

Figure D2. data-proc.py — extract network features with Scapy (See §3.3).



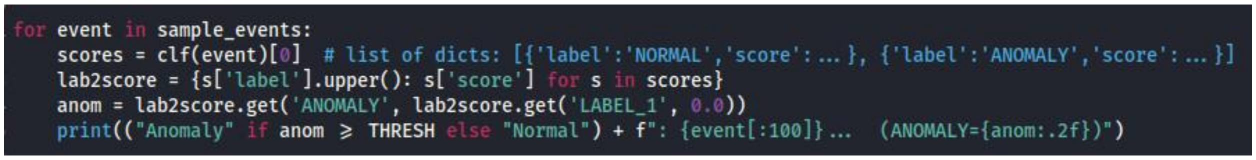
```
# ─────────────────────────
# Label rule (broader patterns)
# ─────────────────────────
ANOMALY_PAT = r"(?:failed password|invalid user|authentication failure|pam .* authentication failures|nmap|port scan)"
df["label"] = df["message"].str.contains(ANOMALY_PAT, case=False, regex=True, na=False).astype(int)
```

Figure D3.1 model-trainer.py — regex labeling (See §3.5).



```
training_args = TrainingArguments(
    output_dir=RESULTS,
    eval_strategy="no",           # evaluate AFTER training only
    save_strategy="no",           # save once at the end
    logging_steps=200,
    learning_rate=2e-5,
    per_device_train_batch_size=16,
    per_device_eval_batch_size=32,
    num_train_epochs=2,           # was 5
    weight_decay=0.01,
    load_best_model_at_end=False,
)
```

Figure D3.2 model-trainer.py — training args (See §3.5).



```
for event in sample_events:
    scores = clf(event)[0]  # list of dicts: [{'label':'NORMAL','score':…}, {'label':'ANOMALY','score':…}]
    lab2score = {s['label'].upper(): s['score'] for s in scores}
    anom = lab2score.get('ANOMALY', lab2score.get('LABEL_1', 0.0))
    print(("Anomaly" if anom ≥ THRESH else "Normal") + f": {event[:100]}…  (ANOMALY={anom:.2f})")
```

Figure D4. Anomaly-detection.py — thresholded inference (See §3.6).

Appendix E — Training & Evaluation Outputs

Context: See §4 Results.

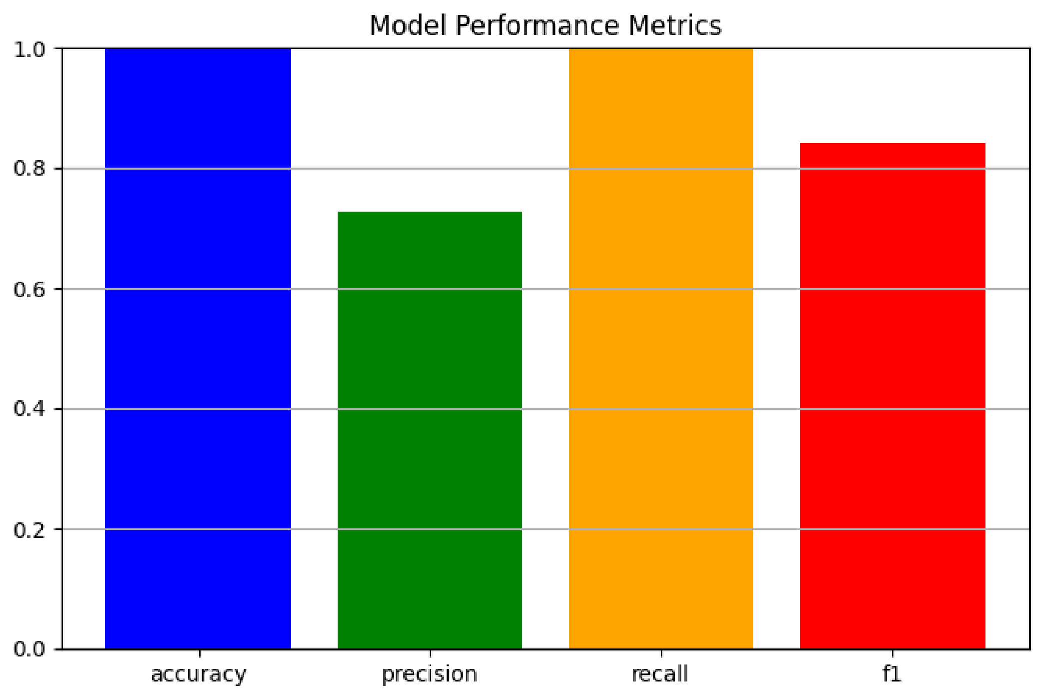
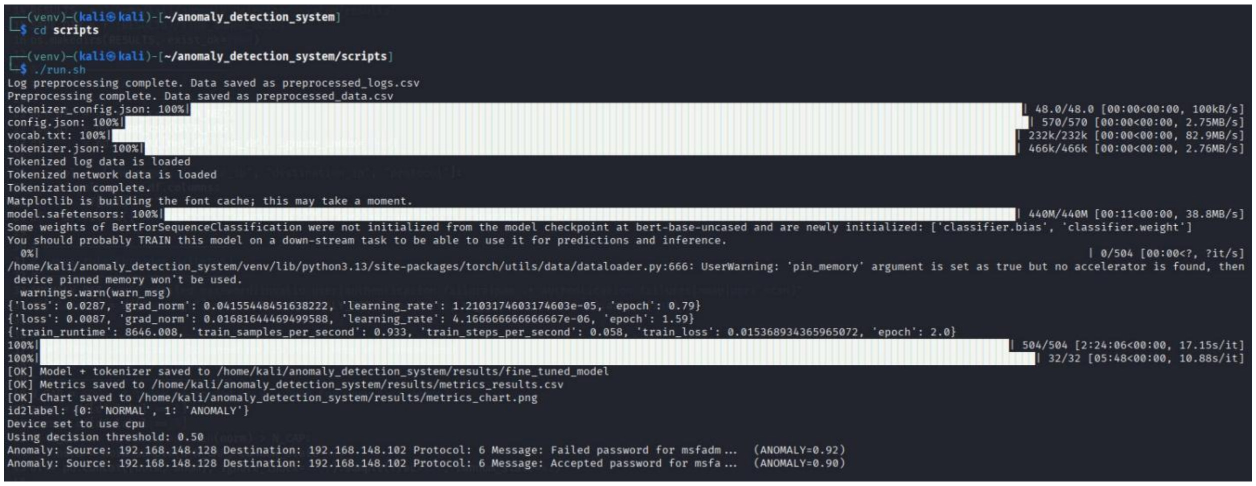


Figure E1. Performance metrics chart (Accuracy, Precision, Recall, F1).



```
┌──(venv)─(kali㉿kali)-[~/anomaly_detection_system]
└─$ cd scripts

┌──(venv)─(kali㉿kali)-[~/anomaly_detection_system/scripts]
└─$ ./run.sh
Log preprocessing complete. Data saved as preprocessed_logs.csv
Preprocessing complete. Data saved as preprocessed_data.csv
tokenizer_config.json: 100%|                    | 48.0/48.0 [00:00<00:00, 100kB/s]
config.json: 100%|                               | 570/570 [00:00<00:00, 2.75MB/s]
vocab.txt: 100%|                                | 232k/232k [00:00<00:00, 82.9MB/s]
tokenizer.json: 100%|                           | 466k/466k [00:00<00:00, 2.76MB/s]
Tokenized log data is loaded
Tokenized network data is loaded
Tokenization complete.
Matplotlib is building the font cache; this may take a moment.
model.safetensors: 100%|                        | 440M/440M [00:11<00:00, 38.8MB/s]
Some weights of BertForSequenceClassification were not initialized from the model checkpoint at bert-base-uncased and are newly initialized: ['classifier.bias', 'classifier.weight']
You should probably TRAIN this model on a down-stream task to be able to use it for predictions and inference.
  0%|                                                     | 0/504 [00:00<?, ?it/s]
/home/kali/anomaly_detection_system/venv/lib/python3.13/site-packages/torch/utils/data/dataloader.py:666: UserWarning: 'pin_memory' argument is set as true but no accelerator is found, then
 device pinned memory won't be used.
  warnings.warn(warn_msg)
{'loss': 0.0287, 'grad_norm': 0.04155448451638222, 'learning_rate': 1.2103174603174603e-05, 'epoch': 0.79}
{'loss': 0.0087, 'grad_norm': 0.01681644469499588, 'learning_rate': 4.166666666666667e-06, 'epoch': 1.59}
{'train_runtime': 8646.008, 'train_samples_per_second': 0.933, 'train_steps_per_second': 0.058, 'train_loss': 0.015368934365965072, 'epoch': 2.0}
100%|                                         | 504/504 [2:24:06<00:00, 17.15s/it]
100%|                                            | 32/32 [05:48<00:00, 10.88s/it]
[OK] Model + tokenizer saved to /home/kali/anomaly_detection_system/results/fine_tuned_model
[OK] Metrics saved to /home/kali/anomaly_detection_system/results/metrics_results.csv
[OK] Chart saved to /home/kali/anomaly_detection_system/results/metrics_chart.png
id2label: {0: 'NORMAL', 1: 'ANOMALY'}
Device set to use cpu
Using decision threshold: 0.50
Anomaly: Source: 192.168.148.128 Destination: 192.168.148.102 Protocol: 6 Message: Failed password for msfadm ...  (ANOMALY=0.92)
Anomaly: Source: 192.168.148.128 Destination: 192.168.148.102 Protocol: 6 Message: Accepted password for msfa ...  (ANOMALY=0.90)
```

Figure E2. End-to-end run output (preprocessing → training → inference).