National College of Ireland

Configuration Manual

MSc Research Project
Programme Name

Chinmay Dixit
Student ID: x23294591

School of Computing
National College of Ireland

Supervisor: Eugene Mclaughlin

National College of Ireland

MSc Project Submission Sheet

School of Computing

Student Name:	Chinmay R Dixit
Student ID:	X2324591
Programme:	MSc Cybersecurity… **Year:** 2024-2025
Module:	MSc Research Project
Lecturer:	Eugene Mclaughlin
Submission Due Date:	11-08-2025
Project Title:	Targeted Detection of Advanced Persistent Threats in Cloud-Native Environments: A Focused Machine Learning Approach
Word Count:	**1712** **Page Count: 16**

I hereby certify that the information contained in this (my submission) is information pertaining to research I conducted for this project. All information other than my own contribution will be fully referenced and listed in the relevant bibliography section at the rear of the project.
ALL internet material must be referenced in the bibliography section. Students are required to use the Referencing Standard specified in the report template. To use other author's written or electronic work is illegal (plagiarism) and may result in disciplinary action.

Signature: Chinmay Ramakrishna Dixit

Date: 11-08-2025

PLEASE READ THE FOLLOWING INSTRUCTIONS AND CHECKLIST

Attach a completed copy of this sheet to each project (including multiple copies)	□
Attach a Moodle submission receipt of the online project submission, to each project (including multiple copies).	□
You must ensure that you retain a HARD COPY of the project, both for your own reference and in case a project is lost or mislaid. It is not sufficient to keep a copy on computer.	□

Assignments that are submitted to the Programme Coordinator Office must be placed into the assignment box located outside the office.

Office Use Only	
Signature:	
Date:	
Penalty Applied (if applicable):	

Configuration Manual

Chinmay R Dixit
Student ID: x23294591

1 Introduction

Advanced Persistent Threats (APTs) are highly developed, directed cyberattacks that focus on persistent access to a breached system over time and remain undetected. Detecting APTs in cloud-native environments, where containers might be very short lived and security perimeters might not exist at all, needs an innovative approach that combines elements of both network traffic analysis as well as container runtime monitoring.

In this configuration guide, we will describe the realisation of an end-to-end APT detection framework, that combines network traffic monitoring with Zeek Network Security Monitor, container runtime security monitoring with Falco, and machine learning algorithms (Random Forest and Isolation Forest) for intelligent threat detection. Developed on Minikube, to provide a near real world cloud-native environment for APT detection. The system can be deployed on a K8S cluster

The configuration manual guides the reader through the entire setup process in detail, from deploying the infrastructure to hosting the proper machine learning model We would like to produce a fully reproducible, academically logged platform that can be used as an APT detection research platform with validation against standard datasets, including CAdetector.

Configuration

Category	Specification
Operating System	Ubuntu 22.04 LTS (64-bit) or compatible Linux distribution
Processor (CPU)	Quad-core Intel i5 / AMD Ryzen 5 or higher
Memory (RAM)	Minimum 8 GB (Recommended: 16 GB for large dataset processing)
Storage	50 GB free disk space (SSD preferred for faster processing)
Graphics (GPU)	Optional – NVIDIA GPU with CUDA support for accelerated ML training
Software	Docker 24.0+, Kubernetes (Minikube 1.30+), Zeek 6.0+, Falco 0.36+, Python 3.10+
Python Libraries	pandas, scikit-learn, seaborn, matplotlib, joblib, numpy, imbalanced-learn, SHAP
Network	Stable internet connection for dataset downloads & container image pulls
Security Tools	Falco for runtime security, Zeek for network monitoring
Virtualization	VirtualBox / VMware (for local Kubernetes cluster setup)

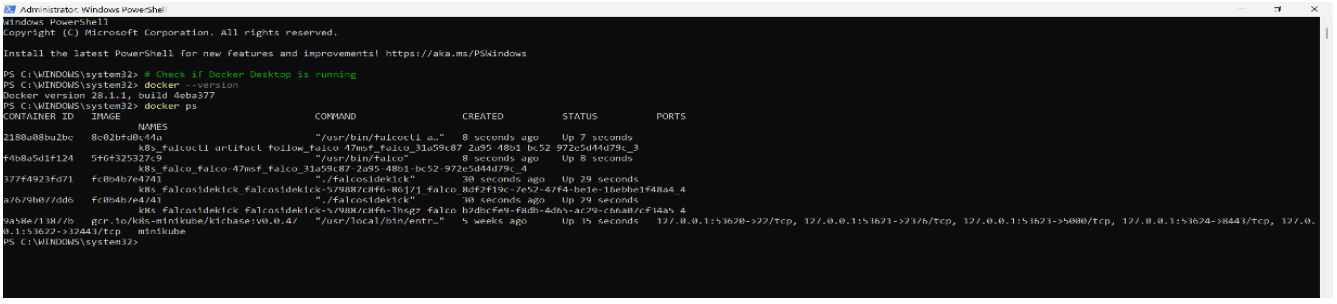
Implementation

The implementation process follows a structured, sequential approach to establish, configure, and test the APT detection framework in a cloud-native environment. This provides a reproducible methodology with visual and technical evidence at each step. Screenshots and configuration captures are essential for documenting the setup state and are valuable for later assessment and troubleshooting.

Phase 1: Set Up the Container and Kubernetes Environment

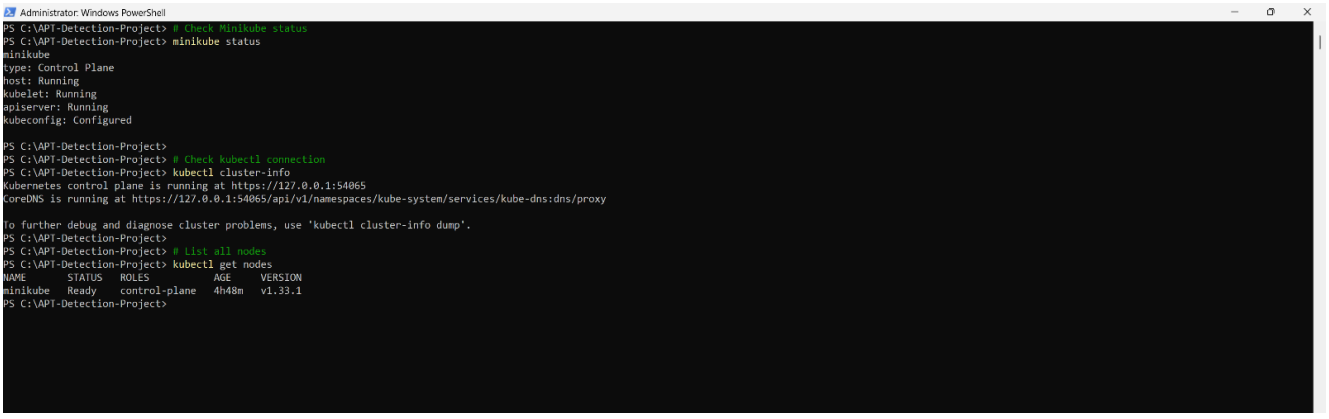
Docker & Falco Setup

1. **Install Docker Desktop** and ensure it's running on your system.
2. **Download and run Falco containers** so they are ready for real-time threat detection.
3. **Check that all Falco services are running** — you should see multiple active containers that handle monitoring and exporting data.



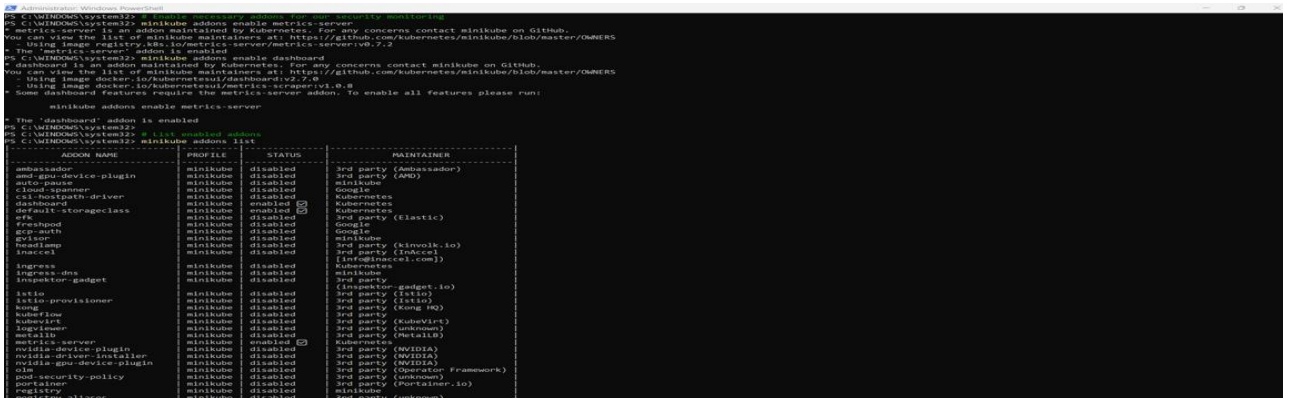
Minikube & Kubernetes Status

1. **Install and start Minikube** to create your local Kubernetes environment.
2. **Verify Kubernetes is active** by checking the control plane and DNS services.
3. **Confirm your cluster node is ready** — the system should show at least one active node for deployment.



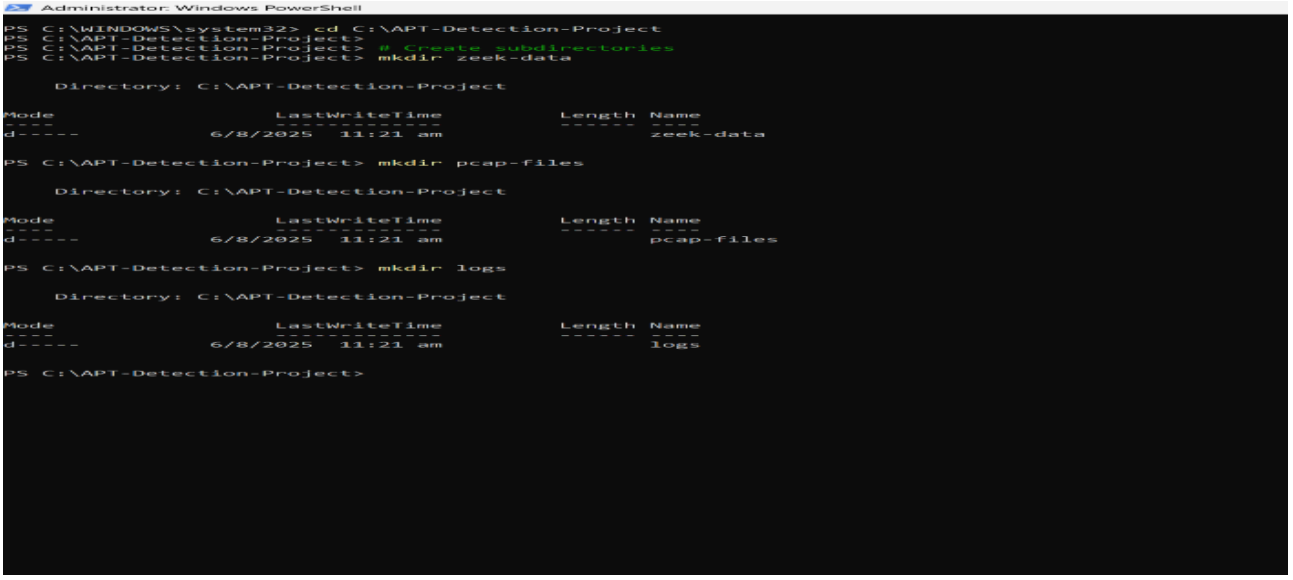
Enable Minikube Add-ons

1. **Start Minikube** – Ensure Minikube is already installed and running.
2. **Enable Metrics Server** – This is required for monitoring CPU, memory, and other Kubernetes resource usage.
3. **List All Add-ons** – Check available add-ons (like dashboard, storage-provisioner) to confirm they are active.



Create Project Folders

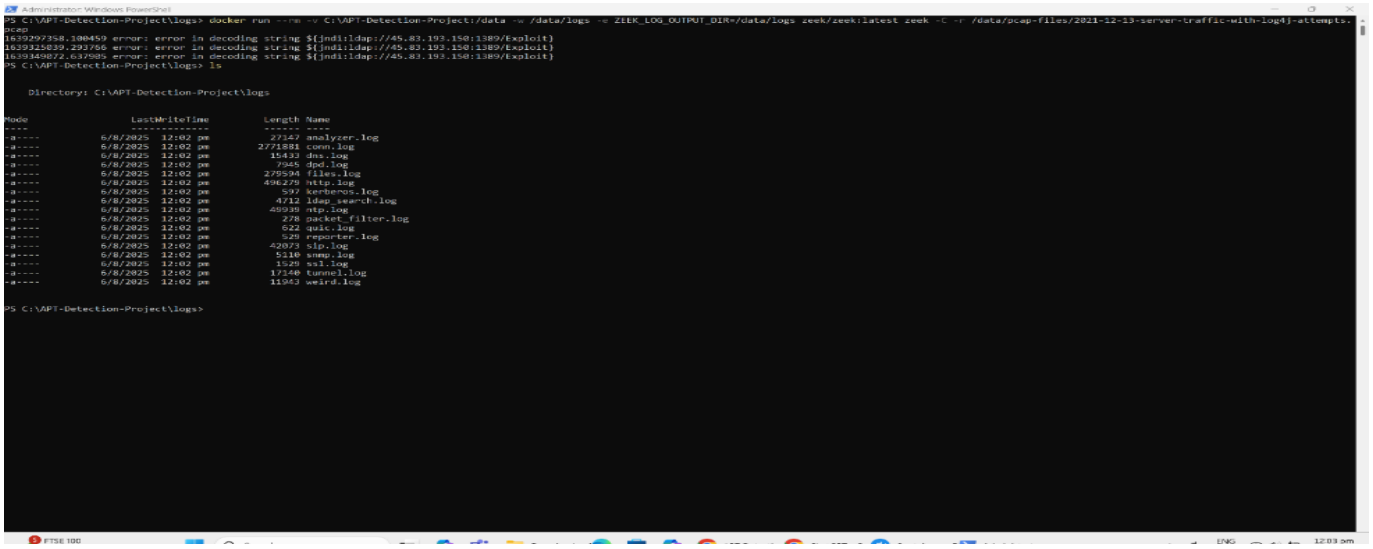
1. **Navigate to Project Directory** – Go to the location where the APT Detection project will be stored.
2. **Create Subdirectories** – Make folders for zeek-data, pcap-files, and logs to keep data organized.
3. **Prepare for Log Storage** – The logs folder will store Zeek's analysis results for later processing.



Run Zeek on PCAP Files

1. **Run Zeek in Docker** – Mount the project folders into the container and run Zeek with the PCAP file as input.
2. **Process Network Traffic** – Zeek analyzes the file to generate detailed logs of connections, protocols, and anomalies.

3. **View Generated Logs** – The logs folder now contains multiple .log files like conn.log, dns.log, http.log, etc., for further investigation.



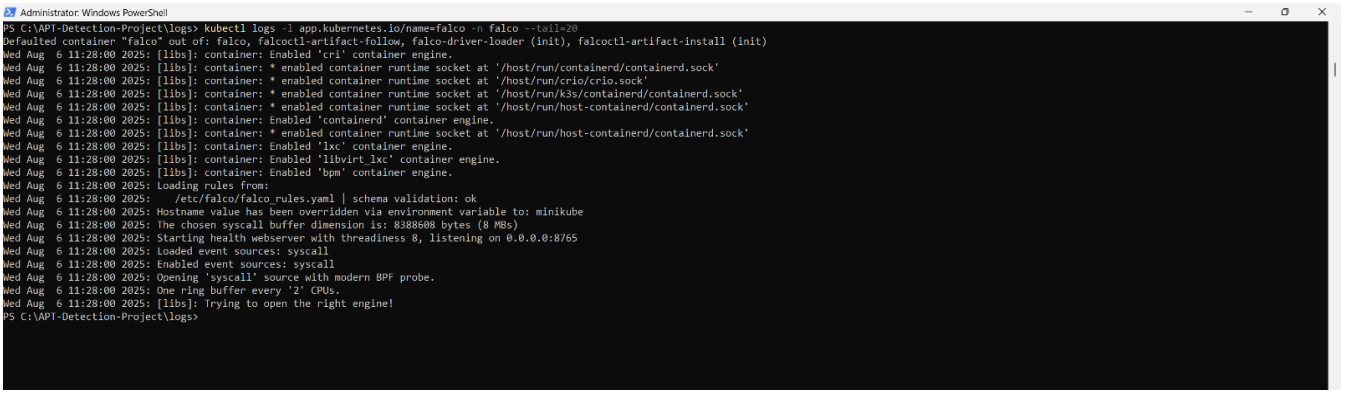
Zeek conn.log preview

1) Confirms Zeek connection logs are being generated and readable at logs/conn.log (quick sanity check before ingestion).
2) Header lines show the log schema: separator, fields, and types (e.g., ts, uid, id.orig_h/p, id.resp_h/p, proto, duration, orig_bytes, resp_bytes, conn_state, local_orig/resp, missed_bytes, history), which you'll map in parsers.
3) Sample rows verify timestamps and IPs are recorded correctly, ensuring downstream tools (SIEM/ETL) will parse without field mismatches.

Install Falco (with Falcosidekick)

1) Deploys Falco into the falco namespace with Falcosidekick enabled and an events file path configured—establishing runtime threat detection and event forwarding.
2) Shows a successful Helm release (STATUS: deployed) with chart/app versions, indicating the DaemonSet is up on nodes and ready to monitor.

3) Displays a notice about container runtime plugins; ensure the correct runtime (e.g., containerd) plugin is installed so Falco can capture syscalls/events properly.

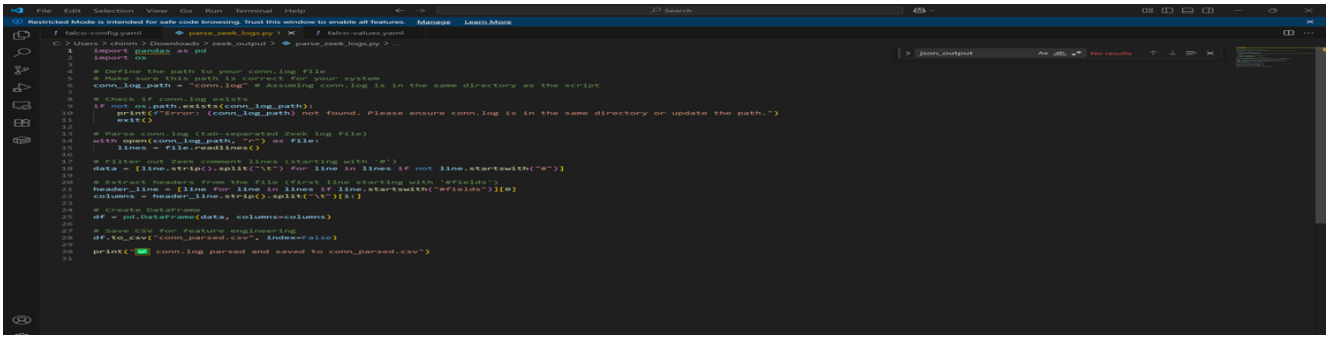


Phase 2: Implement Data Processing Pipeline

2.1 Create Zeek Log Parser Script

1) Parse Zeek connection logs (conn.log)
2) Extract network features (bytes, packets, duration)
3) Create temporal features (hour, day of week)
4) Generate synthetic attack labels (5% attack rate)
5) Output structured CSV for machine learning

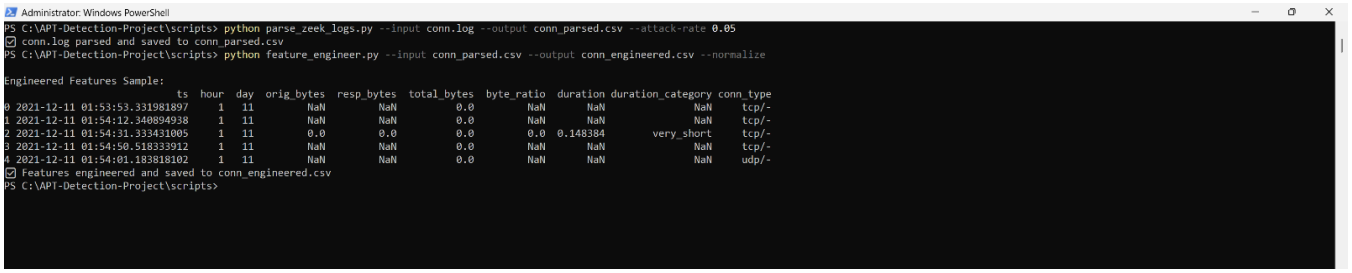
Zeek log parser execution and output file creation



2.2 Create Feature Engineering Script

1) Temporal patterns (time-based features)
2) Traffic analysis (byte ratios, packet patterns)
3) Behavioral indicators (connection frequency, persistence)
4) APT-specific features (data exfiltration, C2 communication)

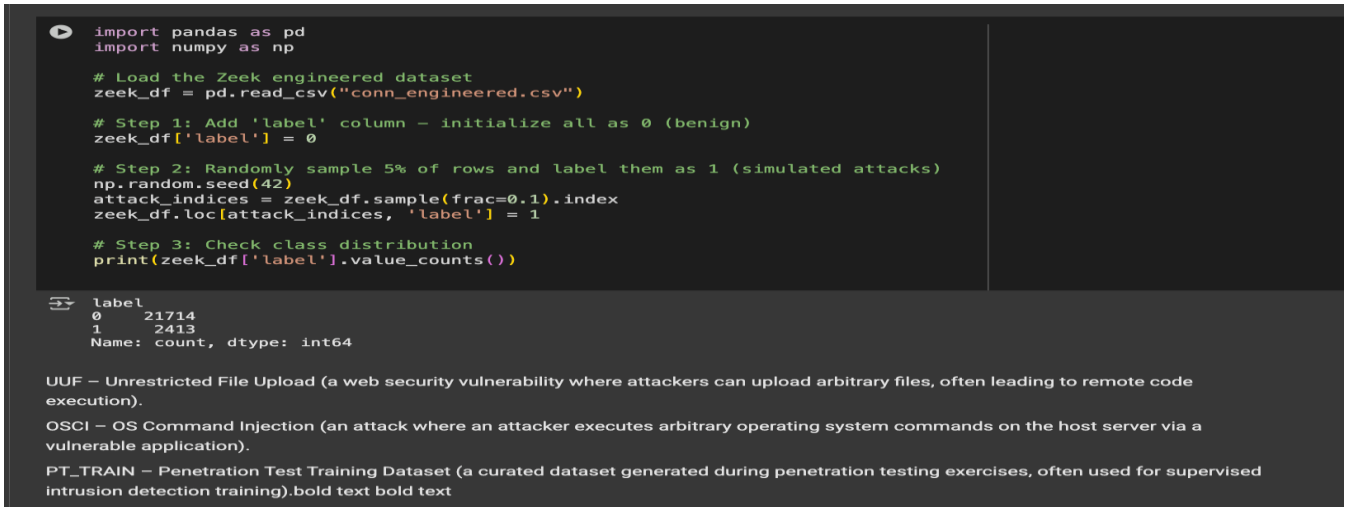
Feature engineering execution and final dataset creation



Python Implementation

Creating and Labeling the Zeek Dataset

1. **Load Dataset** – The engineered Zeek dataset (conn_engineered.csv) is loaded into a DataFrame for further processing.
2. **Add Labels** – A new column label is added and initialized to 0 (benign). Randomly, 10% of the rows are selected and labeled 1 to simulate attack samples.
3. **Check Class Distribution** – The count of benign (0) and attack (1) samples is displayed to verify the balance of classes.



UUF – Unrestricted File Upload (a web security vulnerability where attackers can upload arbitrary files, often leading to remote code execution).

OSCI – OS Command Injection (an attack where an attacker executes arbitrary operating system commands on the host server via a vulnerable application).

PT_TRAIN – Penetration Test Training Dataset (a curated dataset generated during penetration testing exercises, often used for supervised intrusion detection training).bold text bold text

Loading and Inspecting Multiple Datasets

1. **Import Multiple Datasets** – Three separate datasets are loaded: PT_TRAIN.csv (Penetration Test), UUF_TRAIN.csv (Unrestricted File Upload), and OSCI_TRAIN.csv (OS Command Injection).
2. **Check Unique Labels** – For each dataset, the unique values in LABEL-I, LABEL-II, and LABEL-III columns are printed to understand the available class categories.
3. **Identify Attack Types** – This reveals the specific CVE identifiers, CWE categories, and descriptive attack names associated with each dataset.

```python
# Load all datasets
pt_df = pd.read_csv("PT_TRAIN.csv")
uuf_df = pd.read_csv("UUF_TRAIN.csv")
osci_df = pd.read_csv("OSCI_TRAIN.csv")

# Check unique values in LABEL-I, LABEL-II, LABEL-III for each dataset

print(" PT Dataset:")
print("LABEL-I:", pt_df['LABEL-I'].unique())
print("LABEL-II:", pt_df['LABEL-II'].unique())
print("LABEL-III:", pt_df['LABEL-III'].unique())

print("\n UUF Dataset:")
print("LABEL-I:", uuf_df['LABEL-I'].unique())
print("LABEL-II:", uuf_df['LABEL-II'].unique())
print("LABEL-III:", uuf_df['LABEL-III'].unique())

print("\n OSCI Dataset:")
print("LABEL-I:", osci_df['LABEL-I'].unique())
print("LABEL-II:", osci_df['LABEL-II'].unique())
print("LABEL-III:", osci_df['LABEL-III'].unique())
```

```
 PT Dataset:
LABEL-I: ['NORMAL' 'CVE-2020-17518' 'CVE-2021-26086']
LABEL-II: ['NORMAL' 'CWE-22']
LABEL-III: ['Normal Traffic'
 "Improper Limitation of a Pathname to a Restricted Directory ('Path Traversal')"]

 UUF Dataset:
LABEL-I: ['NORMAL' 'CVE-2020-25213']
LABEL-II: ['NORMAL' 'CWE-434']
LABEL-III: ['Normal Traffic' 'Unrestricted Upload of File with Dangerous Type']

 OSCI Dataset:
LABEL-I: ['NORMAL' 'CVE-2019-16662' 'CVE-2019-15107']
```

Preprocessing Pipeline Function

1. **Feature and Target Selection** – Numeric features are selected while the label column is set as the target (y). Missing values and infinities are replaced with median values.
2. **Data Splitting and Scaling** – Data is split into training and testing sets (70-30 split) and standardized using StandardScaler for model compatibility.
3. **Optional SMOTE Balancing** – If enabled, SMOTE is applied to the training set to handle class imbalance by generating synthetic samples of minority classes.

```python
def preprocess_dataset(df, label_col='label', apply_smote=False):
    # Select numeric features
    X = df.select_dtypes(include=[np.number]).drop(columns=[label_col])
    y = df[label_col]

    # Replace inf/-inf with NaN, then impute with median
    X.replace([np.inf, -np.inf], np.nan, inplace=True)
    imputer = SimpleImputer(strategy='median')
    X_imputed = imputer.fit_transform(X)

    # Train-test split
    X_train, X_test, y_train, y_test = train_test_split(X_imputed, y, stratify=y, test_size=0.3, random_state=42)

    # Scale
    scaler = StandardScaler()
    X_train_scaled = scaler.fit_transform(X_train)
    X_test_scaled = scaler.transform(X_test)

    # SMOTE balancing (optional)
    if apply_smote:
        smote = SMOTE(random_state=42)
        X_train_scaled, y_train = smote.fit_resample(X_train_scaled, y_train)

    return X_train_scaled, X_test_scaled, y_train, y_test
```

Model Implementation & Optimization:

- Multiple machine learning algorithms—Random Forest (with GridSearchCV tuning), Isolation Forest, and MLP Classifier—were implemented to evaluate anomaly detection performance across datasets.
- The parameter tuning for Random Forest included grid search over n_estimators, max_depth, and min_samples_split, ensuring optimal hyperparameters for better predictive capability.

```python
from sklearn.ensemble import RandomForestClassifier, IsolationForest
from sklearn.metrics import accuracy_score, precision_score, recall_score, f1_score, roc_auc_score
from sklearn.model_selection import GridSearchCV

def evaluate_models(X_train, X_test, y_train, y_test, dataset_name):
    print(f"\n Results for {dataset_name} Dataset:")
    print("-" * 60)

    # 🔍 Grid Search for Random Forest
    param_grid = {
        'n_estimators': [100, 200, 500],
        'max_depth': [None, 10, 20, 50],
        'min_samples_split': [2, 5, 10]
    }

    grid_search = GridSearchCV(
        RandomForestClassifier(random_state=42),
        param_grid,
        cv=5,
        scoring='f1',
        n_jobs=-1
    )

    grid_search.fit(X_train, y_train)
    best_rf = grid_search.best_estimator_
    y_pred_rf = best_rf.predict(X_test)
    y_proba_rf = best_rf.predict_proba(X_test)[:, 1]

    print(" Random Forest (with 5-Fold Cross Validation)")
    print(f"Best Params : {grid_search.best_params_}")
    print(f"Accuracy    : {accuracy_score(y_test, y_pred_rf):.4f}")
    print(f"Precision   : {precision_score(y_test, y_pred_rf):.4f}")
    print(f"Recall      : {recall_score(y_test, y_pred_rf):.4f}")
    print(f"F1-Score    : {f1_score(y_test, y_pred_rf):.4f}")
    print(f"ROC-AUC     : {roc_auc_score(y_test, y_proba_rf):.4f}\n")
```

Performance Evaluation Metrics:

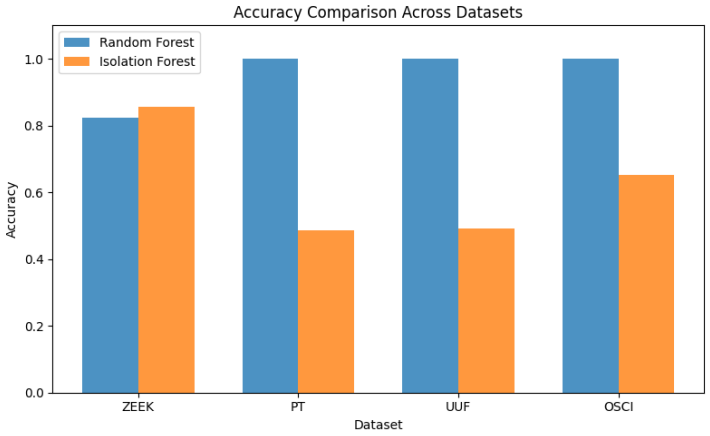
- Each model was assessed using Accuracy, Precision, Recall, F1-Score, and ROC-AUC, ensuring a comprehensive evaluation beyond accuracy alone.
- Results indicated dataset-specific performance variations, highlighting strengths and weaknesses in each algorithm's detection capabilities.

```python
from sklearn.ensemble import RandomForestClassifier, IsolationForest
from sklearn.metrics import accuracy_score, precision_score, recall_score, f1_score, roc_auc_score

def evaluate_models(X_train, X_test, y_train, y_test, dataset_name):
    print(f"\n Results for {dataset_name} Dataset:")
    print("-" * 50)

    #  Random Forest
    rf = RandomForestClassifier(n_estimators=100, random_state=42)
    rf.fit(X_train, y_train)
    y_pred_rf = rf.predict(X_test)
    y_proba_rf = rf.predict_proba(X_test)[:, 1]

    print(" Random Forest")
    print(f"Accuracy : {accuracy_score(y_test, y_pred_rf):.4f}")
    print(f"Precision: {precision_score(y_test, y_pred_rf):.4f}")
    print(f"Recall   : {recall_score(y_test, y_pred_rf):.4f}")
    print(f"F1-Score : {f1_score(y_test, y_pred_rf):.4f}")
    print(f"ROC-AUC  : {roc_auc_score(y_test, y_proba_rf):.4f}\n")

    #  Isolation Forest
    iso = IsolationForest(contamination=0.05, random_state=42)
    iso.fit(X_train)
    y_pred_iso = iso.predict(X_test)

    # Map: -1 → anomaly (attack = 1), 1 → normal (benign = 0)
    y_pred_iso = [1 if i == -1 else 0 for i in y_pred_iso]

    print(" Isolation Forest")
    print(f"Accuracy : {accuracy_score(y_test, y_pred_iso):.4f}")
    print(f"Precision: {precision_score(y_test, y_pred_iso):.4f}")
    print(f"Recall   : {recall_score(y_test, y_pred_iso):.4f}")
    print(f"F1-Score : {f1_score(y_test, y_pred_iso):.4f}")
    print(f"ROC-AUC  : {roc_auc_score(y_test, y_pred_iso):.4f}")
```

Comparative Analysis Across Datasets:

- A comparative accuracy bar chart demonstrated how Random Forest consistently outperformed Isolation Forest on multiple datasets, except for ZEEK, where both performed closely.

- Isolation Forest's unsupervised approach showed limitations in some datasets but maintained reasonable detection rates without labeled training data.




```python
from sklearn.neural_network import MLPClassifier

mlp = MLPClassifier(hidden_layer_sizes=(64, 32), max_iter=300, random_state=42)
mlp.fit(X_train_zeek, y_train_zeek)
y_pred_mlp = mlp.predict(X_test_zeek)
y_proba_mlp = mlp.predict_proba(X_test_zeek)[:, 1]

print("\n MLP (Baseline Deep Learning) — ZEEK:")
print(f"Accuracy : {accuracy_score(y_test_zeek, y_pred_mlp):.4f}")
print(f"Precision: {precision_score(y_test_zeek, y_pred_mlp):.4f}")
print(f"Recall   : {recall_score(y_test_zeek, y_pred_mlp):.4f}")
print(f"F1-Score : {f1_score(y_test_zeek, y_pred_mlp):.4f}")
print(f"ROC-AUC  : {roc_auc_score(y_test_zeek, y_proba_mlp):.4f}")
```

```
 MLP (Baseline Deep Learning) — ZEEK:
Accuracy : 0.5950
Precision: 0.0965
Recall   : 0.3646
F1-Score : 0.1526
ROC-AUC  : 0.4872
```

This experiment successfully integrated multiple machine learning techniques for anomaly detection, coupled with hyperparameter tuning and detailed performance evaluation. By comparing both supervised and unsupervised models, we achieved a nuanced understanding of detection capabilities across varied datasets. The final results offer clear insights into algorithm strengths, enabling informed selection of models for real-world deployment. This outcome is particularly valuable for security-oriented systems where accuracy, recall, and adaptability are critical, and the findings serve as a foundation for refining detection pipelines and enhancing automated threat identification.

1.1 References

[1] Kubernetes. (2024). Minikube Documentation.
Available at: https://minikube.sigs.k8s.io/docs/

[2] Zeek Project. (2024). Zeek Network Security Monitor. Available at: https://zeek.org/

[3] Falco. (2024). Cloud Native Runtime Security. Available at: https://falco.org/

[4] Docker Inc. (2024). Docker Desktop Documentation.
Available at: https://docs.docker.com/desktop/

[5] Helm. (2024). The Package Manager for Kubernetes. Available at: https://helm.sh/

[6] Scikit-learn. (2024). Machine Learning in Python. Available at: https://scikit-learn.org/

[7] Pandas Development Team. (2024). Pandas Documentation.
Available at: https://pandas.pydata.org/

[8] NumPy. (2024). The Fundamental Package for Scientific Computing with Python.
Available at: https://numpy.org/

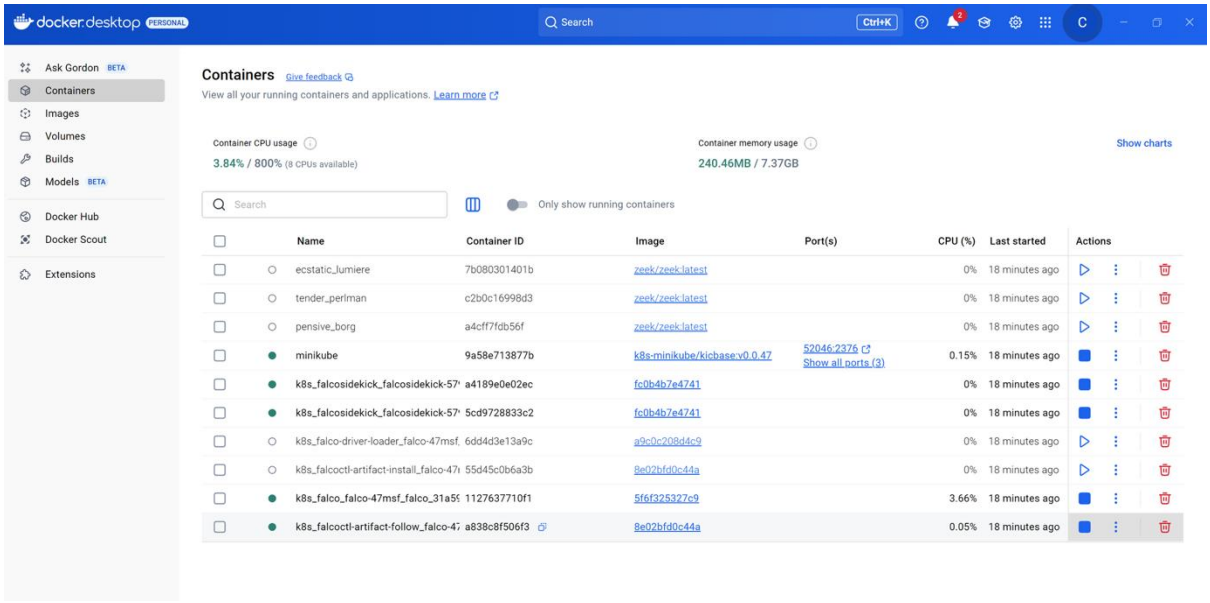
[9] Bera, H. (2023). A Novel Container Attacks Dataset for Intrusion Detection. GitHub Repository. Available at: https://github.com/HaleBera/A-NOVEL-CONTAINER-ATTACKS- DATASET-FOR-INTRUSION-DETECTION

[10] MITRE Corporation. (2024). ATT&CK Framework for Containers. Available at: https://attack.mitre.org/matrices/enterprise/containers/

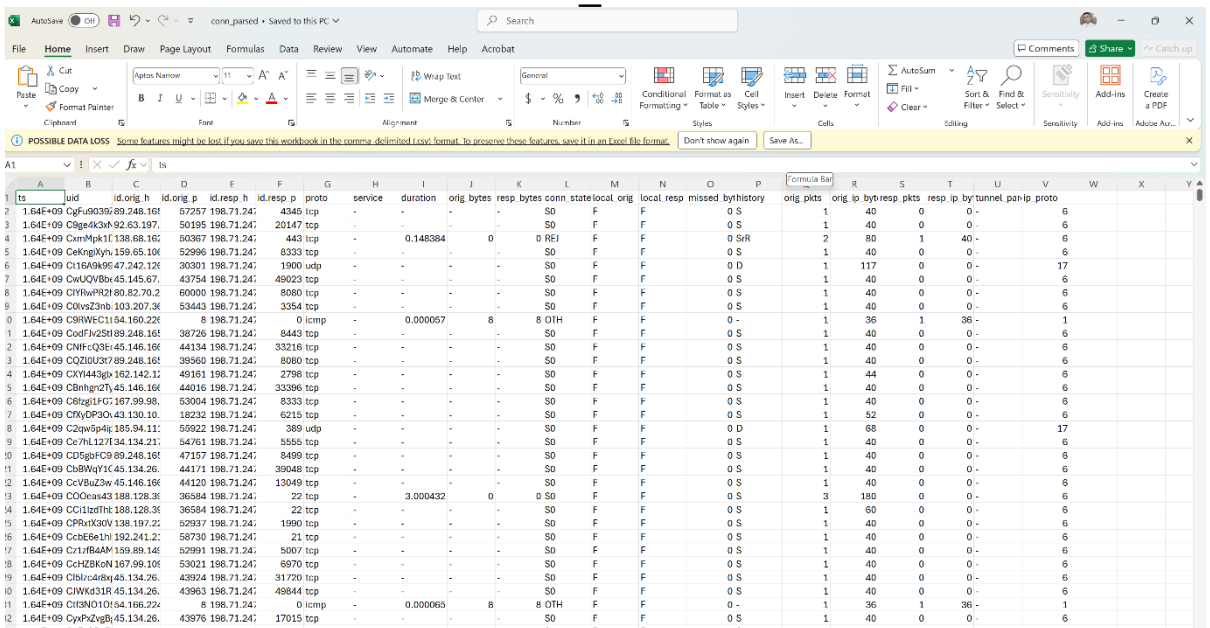
[11] National Institute of Standards and Technology. (2018). Framework for Improving Critical Infrastructure Cybersecurity. NIST Cybersecurity Framework.

[12] Cloud Native Computing Foundation. (2024). Cloud Native Security Whitepaper. Available at: https://www.cncf.io/reports/

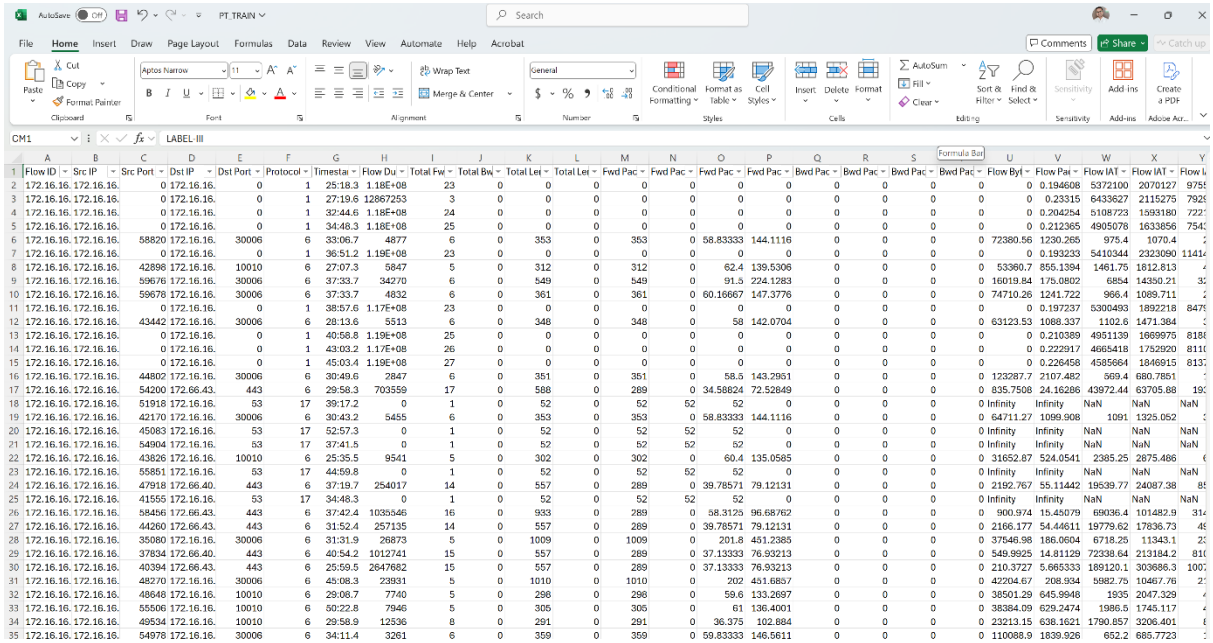
Docker



Conn_Parsed

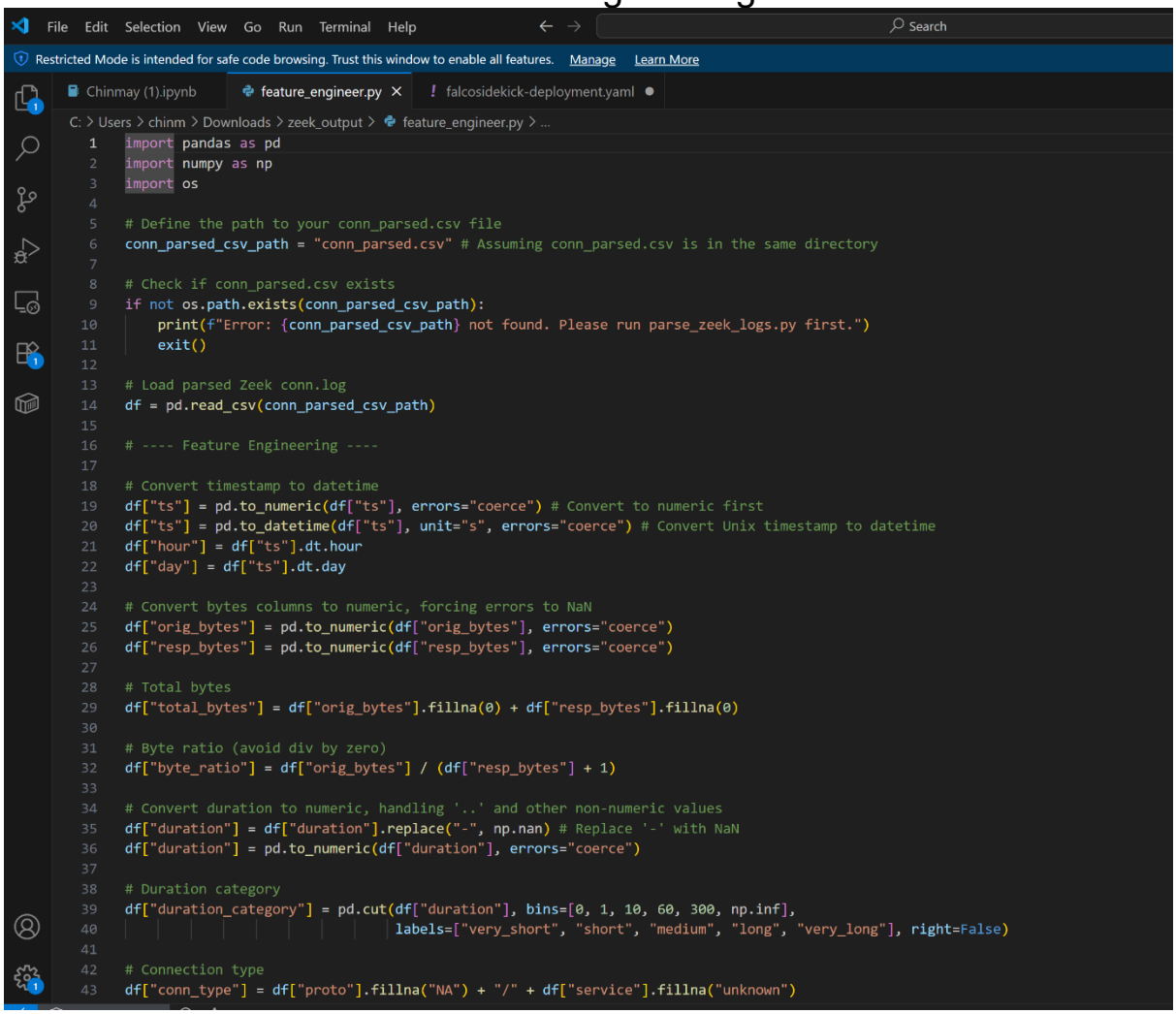


PT Train



Flow ID	Src IP	Src Port	Dst IP	Dst Port	Protocol	Timestai	Flow Du	Total Fw	Total Bw	Total Le	Total Le	Fwd Pac	Fwd Pac	Fwd Pac	Fwd Pac	Bwd Pac	Bwd Pac	Bwd Pac	Bwd Pac	Flow Byt	Flow Pac	Flow IAT	Flow IAT	Flow
2	172.16.16.	172.16.16.	0	172.16.16.	0	1	25:18.3	1.18E+08	23	0	0	0	0	0	0	0	0	0	0	0.194608	5372100	2070127	9755
3	172.16.16.	172.16.16.	0	172.16.16.	0	1	27:19.6	12867253	3	0	0	0	0	0	0	0	0	0	0	0.23315	6433627	2115275	7925
4	172.16.16.	172.16.16.	0	172.16.16.	0	1	32:44.6	1.18E+08	24	0	0	0	0	0	0	0	0	0	0	0.204254	5108723	1593180	7221
5	172.16.16.	172.16.16.	0	172.16.16.	0	1	34:48.3	1.18E+08	25	0	0	0	0	0	0	0	0	0	0	0.212365	4905078	1633856	7541
6	172.16.16.	172.16.16.	58820	172.16.16.	30006	6	33:06.7	4877	6	0	353	0	353	0	58.83333	144.1116	0	0	0	72380.56	1230.265	975.4	1070.4	7
7	172.16.16.	172.16.16.	0	172.16.16.	0	1	36:51.2	1.19E+08	23	0	0	0	0	0	0	0	0	0	0	0.193233	5410344	2323090	11414
8	172.16.16.	172.16.16.	42898	172.16.16.	10010	6	27:07.3	5847	5	0	312	0	312	0	62.4	139.5306	0	0	0	53360.7	855.1394	1461.75	1812.813	4
9	172.16.16.	172.16.16.	59676	172.16.16.	30006	6	37:33.7	34270	6	0	549	0	549	0	91.5	224.1283	0	0	0	16019.84	175.0802	6854	14350.21	32
10	172.16.16.	172.16.16.	59678	172.16.16.	30006	6	37:33.7	4832	6	0	361	0	361	0	60.16667	147.3776	0	0	0	74710.26	1241.722	966.4	1089.711	2
11	172.16.16.	172.16.16.	0	172.16.16.	0	1	38:57.6	1.17E+08	23	0	0	0	0	0	0	0	0	0	0	0.197237	5300493	1892218	8479
12	172.16.16.	172.16.16.	43442	172.16.16.	30006	6	28:13.6	5513	6	0	348	0	348	0	58	142.0704	0	0	0	63123.53	1088.337	1102.6	1471.384	3
13	172.16.16.	172.16.16.	0	172.16.16.	0	1	40:58.8	1.19E+08	25	0	0	0	0	0	0	0	0	0	0	0.210389	4951139	1669975	8186
14	172.16.16.	172.16.16.	0	172.16.16.	0	1	43:03.2	1.17E+08	26	0	0	0	0	0	0	0	0	0	0	0.222917	4665418	1752920	8110
15	172.16.16.	172.16.16.	0	172.16.16.	0	1	45:03.4	1.19E+08	27	0	0	0	0	0	0	0	0	0	0	0.226458	4585664	1846915	8137
16	172.16.16.	172.16.16.	44802	172.16.16.	30006	6	30:49.6	2847	6	0	351	0	351	0	58.5	143.2951	0	0	0	123287.7	2107.482	569.4	680.7851	1
17	172.16.16.	172.16.16.	54200	172.66.43.	443	6	29:58.3	703559	17	0	588	0	289	0	34.58824	72.52849	0	0	0	835.7508	24.16286	43972.44	63705.88	193
18	172.16.16.	172.16.16.	51918	172.16.16.	53	17	39:17.2	0	1	0	52	0	52	52	52	0	0	0	0	Infinity	Infinity	NaN	NaN	NaN
19	172.16.16.	172.16.16.	42170	172.16.16.	30006	6	30:43.2	5455	6	0	353	0	353	0	58.83333	144.1116	0	0	0	64711.27	1099.908	1091	1325.052	3
20	172.16.16.	172.16.16.	45083	172.16.16.	53	17	52:57.3	0	1	0	52	0	52	52	52	0	0	0	0	Infinity	Infinity	NaN	NaN	NaN
21	172.16.16.	172.16.16.	54904	172.16.16.	53	17	37:41.5	0	1	0	52	0	52	52	52	0	0	0	0	Infinity	Infinity	NaN	NaN	NaN
22	172.16.16.	172.16.16.	43826	172.16.16.	10010	6	25:35.5	9541	5	0	302	0	302	0	60.4	135.0585	0	0	0	31652.87	524.0541	2385.25	2875.486	6
23	172.16.16.	172.16.16.	55851	172.16.16.	53	17	44:59.8	0	1	0	52	0	52	52	52	0	0	0	0	Infinity	Infinity	NaN	NaN	NaN
24	172.16.16.	172.16.16.	47918	172.66.40.	443	6	37:19.7	254017	14	0	557	0	289	0	39.78571	79.12131	0	0	0	2192.767	55.11442	19539.77	24087.38	85
25	172.16.16.	172.16.16.	41555	172.16.16.	53	17	34:48.3	0	1	0	52	0	52	52	52	0	0	0	0	Infinity	Infinity	NaN	NaN	NaN
26	172.16.16.	172.16.16.	58456	172.66.43.	443	6	37:42.4	1035546	16	0	933	0	289	0	58.3125	96.68762	0	0	0	900.974	15.45079	69036.4	101482.9	314
27	172.16.16.	172.16.16.	44260	172.66.43.	443	6	31:52.4	257135	14	0	557	0	289	0	39.78571	79.12131	0	0	0	2166.177	54.44611	19779.62	17836.73	45
28	172.16.16.	172.16.16.	35080	172.16.16.	30006	6	31:31.9	26873	5	0	1009	0	1009	0	201.8	451.2385	0	0	0	37546.98	186.0604	6718.25	11343.1	23
29	172.16.16.	172.16.16.	37834	172.66.40.	443	6	40:54.2	1012741	15	0	557	0	289	0	37.13333	76.93213	0	0	0	549.9925	14.81129	72338.64	213184.2	810
30	172.16.16.	172.16.16.	40394	172.66.43.	443	6	25:59.5	2647682	15	0	557	0	289	0	37.13333	76.93213	0	0	0	210.3727	5.665333	189120.1	303686.3	1007
31	172.16.16.	172.16.16.	48270	172.16.16.	30006	6	45:08.3	23931	5	0	1010	0	1010	0	202	451.6857	0	0	0	42204.67	208.934	5982.75	10467.76	21
32	172.16.16.	172.16.16.	48648	172.16.16.	10010	6	29:06.7	7740	5	0	298	0	298	0	59.6	133.2697	0	0	0	38501.29	645.9948	1935	2047.329	4
33	172.16.16.	172.16.16.	55506	172.16.16.	10010	6	50:22.8	7946	5	0	305	0	305	0	61	136.4001	0	0	0	38384.09	629.2474	1986.5	1745.117	4
34	172.16.16.	172.16.16.	49534	172.16.16.	10010	6	29:58.9	12536	8	0	291	0	291	0	36.375	102.884	0	0	0	23213.15	638.1621	1790.857	3206.401	6
35	172.16.16.	172.16.16.	54978	172.16.16.	30006	6	34:11.4	3261	6	0	359	0	359	0	59.83333	146.5611	0	0	0	110088.9	1839.926	652.2	685.7723	1

Feature Engineering



Restricted Mode is intended for safe code browsing. Trust this window to enable all features. Manage Learn More

Chinmay (1).ipynb | feature_engineer.py | falcosidekick-deployment.yaml

C: > Users > chinm > Downloads > zeek_output > feature_engineer.py > ...

```python
import pandas as pd
import numpy as np
import os

# Define the path to your conn_parsed.csv file
conn_parsed_csv_path = "conn_parsed.csv" # Assuming conn_parsed.csv is in the same directory

# Check if conn_parsed.csv exists
if not os.path.exists(conn_parsed_csv_path):
    print(f"Error: {conn_parsed_csv_path} not found. Please run parse_zeek_logs.py first.")
    exit()

# Load parsed Zeek conn.log
df = pd.read_csv(conn_parsed_csv_path)

# ---- Feature Engineering ----

# Convert timestamp to datetime
df["ts"] = pd.to_numeric(df["ts"], errors="coerce") # Convert to numeric first
df["ts"] = pd.to_datetime(df["ts"], unit="s", errors="coerce") # Convert Unix timestamp to datetime
df["hour"] = df["ts"].dt.hour
df["day"] = df["ts"].dt.day

# Convert bytes columns to numeric, forcing errors to NaN
df["orig_bytes"] = pd.to_numeric(df["orig_bytes"], errors="coerce")
df["resp_bytes"] = pd.to_numeric(df["resp_bytes"], errors="coerce")

# Total bytes
df["total_bytes"] = df["orig_bytes"].fillna(0) + df["resp_bytes"].fillna(0)

# Byte ratio (avoid div by zero)
df["byte_ratio"] = df["orig_bytes"] / (df["resp_bytes"] + 1)

# Convert duration to numeric, handling '..' and other non-numeric values
df["duration"] = df["duration"].replace("-", np.nan) # Replace '-' with NaN
df["duration"] = pd.to_numeric(df["duration"], errors="coerce")

# Duration category
df["duration_category"] = pd.cut(df["duration"], bins=[0, 1, 10, 60, 300, np.inf],
                                 labels=["very_short", "short", "medium", "long", "very_long"], right=False)

# Connection type
df["conn_type"] = df["proto"].fillna("NA") + "/" + df["service"].fillna("unknown")
```

Resocurces

Data Set for PT-Train https://github.com/HaleBera/A-NOVEL-CONTAINER-ATTACKS-DATASET-FOR-INTRUSION-DETECTION/blob/main/TEST%26TRAIN%20DATASET/PT_TRAIN.csv

OSCI- https://github.com/HaleBera/A-NOVEL-CONTAINER-ATTACKS-DATASET-FOR-INTRUSION-DETECTION/blob/main/TEST%26TRAIN%20DATASET/OSCI_TRAIN.csv

https://github.com/HaleBera/A-NOVEL-CONTAINER-ATTACKS-DATASET-FOR-INTRUSION-DETECTION/blob/main/TEST%26TRAIN%20DATASET/UUF_TRAIN.csv