EX.NO.: 08
DATE: 29.07.2025

RANSOMWARE DETECTION

AIM

To develop a real-time ransomware detection system that monitors file system activities and classifies them as either benign or ransomware using a trained machine learning model based on file entropy, size, rename actions, and time-based behavior patterns.

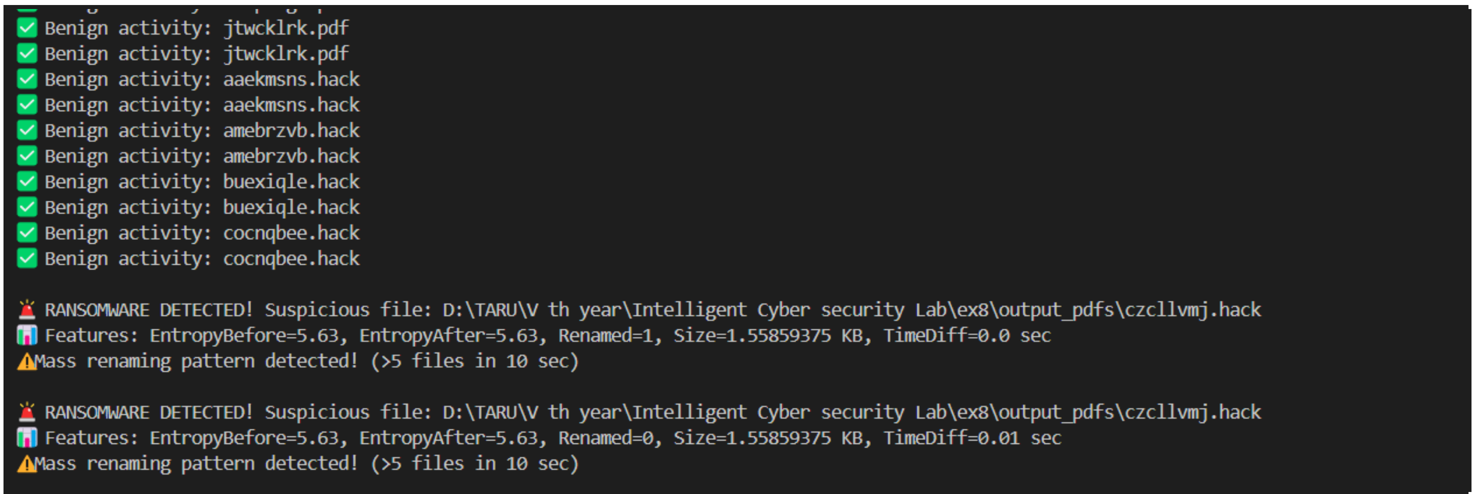
ALGORITHM

1. Load the ransomware dataset from a CSV file.
2. Preprocess the dataset by encoding categorical values and converting time strings to numerical format.
3. Select relevant features such as entropy before and after, renamed status, file size, and time between actions.
4. Split the dataset into training and testing sets.
5. Train a Random Forest classifier on the training data.
6. Evaluate the model using classification metrics and plot feature importance.
7. Save the trained model using joblib for later use.
8. Monitor a specified directory in real time using the watchdog library.
9. Extract features from each file event such as creation, modification, or renaming.
10. Predict whether the file activity is ransomware or benign using the trained model.
11. Trigger an alert if ransomware is detected or if mass renaming behavior is observed.

CODE AND OUTPUT

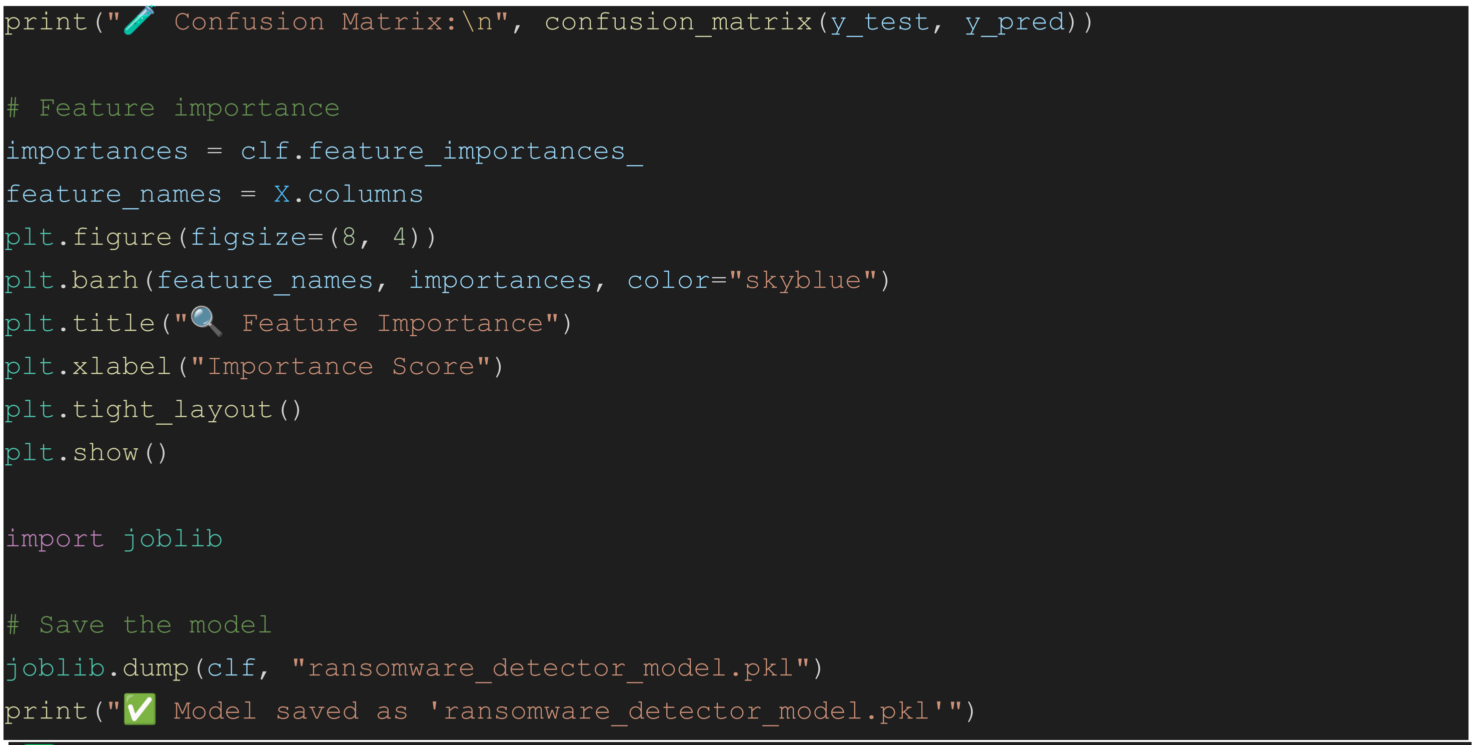
```
import os
import time
import joblib
import pandas as pd
from collections import deque
from watchdog.observers import Observer
from watchdog.events import FileSystemEventHandler
from utils import compute_entropy  # Make sure this is in utils.py

# Load trained ML model
model = joblib.load(
    r"D:\TARU\V th year\Intelligent Cyber security
Lab\ex8\ransomware_detector_model.pkl"
)
print("✅ Loaded ML model.")

# Directory to monitor
WATCH_DIR = r"D:\TARU\V th year\Intelligent Cyber security Lab\ex8\output_pdfs"
os.makedirs(WATCH_DIR, exist_ok=True)

# Store previous file info
file_state = {}
rename_log = deque(maxlen=20)  # store recent rename timestamps



class MonitorHandler(FileSystemEventHandler):
```

```python
    def on_created(self, event):
        if not event.is_directory:
            handle_event(event.src_path, "created")

    def on_modified(self, event):
        if not event.is_directory:
            handle_event(event.src_path, "modified")

    def on_moved(self, event):
        if not event.is_directory:
            rename_log.append(time.time())
            handle_event(event.dest_path, "renamed")


def extract_features(path, event_type):
    try:
        entropy_now = compute_entropy(path)
        size_kb = os.path.getsize(path) / 1024
        now = time.time()

        if path not in file_state:
            file_state[path] = {"entropy": entropy_now, "time": now, "size": size_kb}
            return [
                entropy_now,
                entropy_now,
                int(event_type == "renamed"),
                size_kb,
                0.0,
            ]

        prev = file_state[path]
        time_diff = now - prev["time"]
        entropy_before = prev["entropy"]

        file_state[path] = {"entropy": entropy_now, "time": now, "size": size_kb}

        return [
            entropy_before,
            entropy_now,
            int(event_type == "renamed"),
            size_kb,
            round(time_diff, 2),
        ]
    except Exception as e:
        print(f"[ERROR] Failed to extract features from {path}: {e}")
        return None


def is_mass_renaming(threshold=5, interval=10):
    now = time.time()
    recent = [t for t in rename_log if now - t <= interval]
    return len(recent) >= threshold


def handle_event(path, event_type):
    features = extract_features(path, event_type)
    if features is None:
        return

    feature_names = [
        "Entropy_Before",
        "Entropy_After",
        "Renamed",
        "Size (KB)",
        "Time_Between_Actions",
    ]
    features_df = pd.DataFrame([features], columns=feature_names)
    prediction = model.predict(features_df)[0]

    mass_rename = is_mass_renaming()

    if prediction == 1 or mass_rename:
        print(f"\n🚨 RANSOMWARE DETECTED! Suspicious file: {path}")
        print(
            f"📊 Features: EntropyBefore={features[0]}, EntropyAfter={features[1]},
Renamed={features[2]}, Size={features[3]} KB, TimeDiff={features[4]} sec"
        )
        if mass_rename:
            print("⚠️ Mass renaming pattern detected! (>5 files in 10 sec)")
    else:
        print(f"✅ Benign activity: {os.path.basename(path)}")


if __name__ == "__main__":
    observer = Observer()
    handler = MonitorHandler()
    observer.schedule(handler, path=WATCH_DIR, recursive=True)
    observer.start()
    print(f"🔍 Monitoring directory: {WATCH_DIR} for ransomware activity...")

    try:
        while True:
            time.sleep(1)
    except KeyboardInterrupt:
        observer.stop()
        print("🛑 Stopped monitoring.")
    observer.join()
```

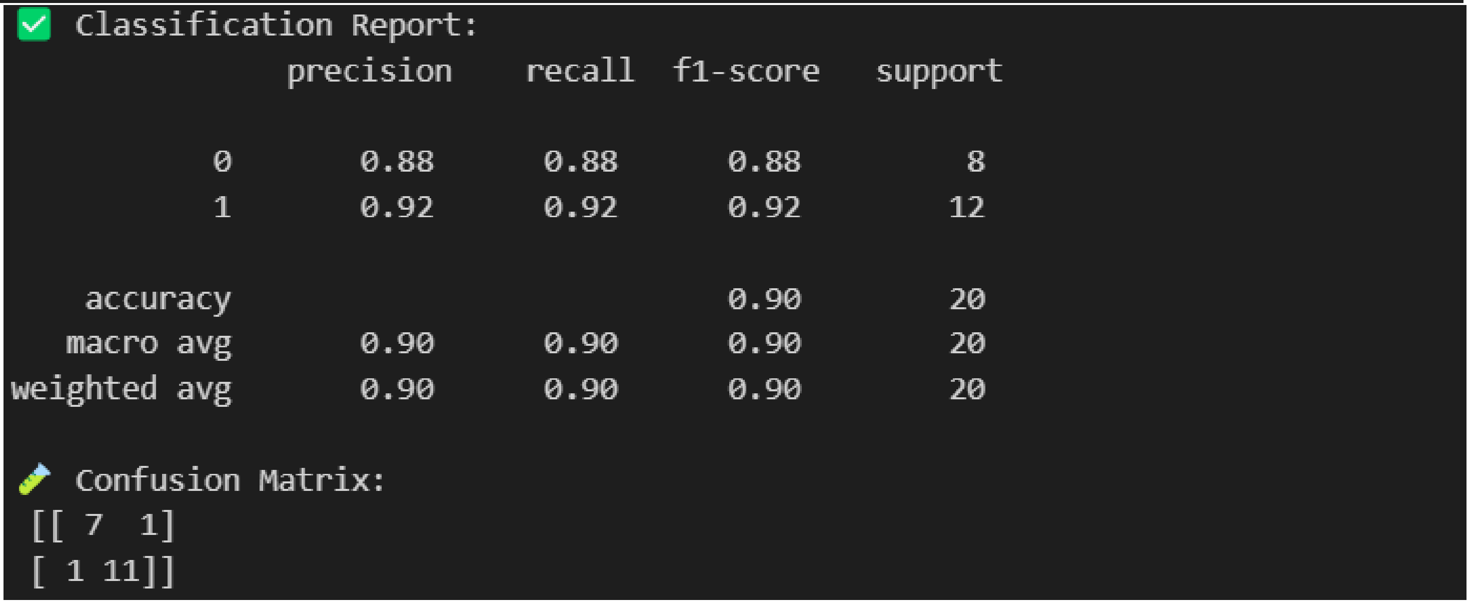


```
✅ Benign activity: jtwcklrk.pdf
✅ Benign activity: jtwcklrk.pdf
✅ Benign activity: aaekmsns.hack
✅ Benign activity: aaekmsns.hack
✅ Benign activity: amebrzvb.hack
✅ Benign activity: amebrzvb.hack
✅ Benign activity: buexiqle.hack
✅ Benign activity: buexiqle.hack
✅ Benign activity: cocnqbee.hack
✅ Benign activity: cocnqbee.hack

🚨 RANSOMWARE DETECTED! Suspicious file: D:\TARU\V th year\Intelligent Cyber security Lab\ex8\output_pdfs\czcllvmj.hack
📊 Features: EntropyBefore=5.63, EntropyAfter=5.63, Renamed=1, Size=1.55859375 KB, TimeDiff=0.0 sec
⚠️Mass renaming pattern detected! (>5 files in 10 sec)

🚨 RANSOMWARE DETECTED! Suspicious file: D:\TARU\V th year\Intelligent Cyber security Lab\ex8\output_pdfs\czcllvmj.hack
📊 Features: EntropyBefore=5.63, EntropyAfter=5.63, Renamed=0, Size=1.55859375 KB, TimeDiff=0.01 sec
⚠️Mass renaming pattern detected! (>5 files in 10 sec)
```

```python
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.ensemble import RandomForestClassifier
from sklearn.preprocessing import LabelEncoder
from sklearn.metrics import classification_report, confusion_matrix
import matplotlib.pyplot as plt

# Load dataset
df = pd.read_csv("ransomware_dataset.csv")

# Convert "Yes"/"No" in 'Renamed' and "Benign"/"Ransomware" in 'Label' to 1/0
df["Renamed"] = df["Renamed"].map({"Yes": 1, "No": 0})
df["Label"] = df["Label"].map({"Ransomware": 1, "Benign": 0})

# Convert "Time_Between_Actions" from "xx sec" to float
df["Time_Between_Actions"] = (
    df["Time_Between_Actions"].str.replace(" sec", "").astype(float)
)

# Features and target
X = df[
    ["Entropy_Before", "Entropy_After", "Renamed", "Size (KB)", "Time_Between_Actions"]
]
y = df["Label"]

# Split
X_train, X_test, y_train, y_test = train_test_split(
    X, y, test_size=0.2, random_state=42
)

# Train model
clf = RandomForestClassifier(n_estimators=100, random_state=42)
clf.fit(X_train, y_train)

# Evaluate
y_pred = clf.predict(X_test)
print("✅ Classification Report:\n", classification_report(y_test, y_pred))
```



```python
print("🧪 Confusion Matrix:\n", confusion_matrix(y_test, y_pred))

# Feature importance
importances = clf.feature_importances_
feature_names = X.columns
plt.figure(figsize=(8, 4))
plt.barh(feature_names, importances, color="skyblue")
plt.title("🔍 Feature Importance")
plt.xlabel("Importance Score")
plt.tight_layout()
plt.show()

import joblib

# Save the model
joblib.dump(clf, "ransomware_detector_model.pkl")
print("✅ Model saved as 'ransomware_detector_model.pkl'")
```



```
✅ Classification Report:
              precision    recall  f1-score   support

           0       0.88      0.88      0.88         8
           1       0.92      0.92      0.92        12

    accuracy                           0.90        20
   macro avg       0.90      0.90      0.90        20
weighted avg       0.90      0.90      0.90        20

🧪 Confusion Matrix:
 [[ 7  1]
 [ 1 11]]
```

INFERENCE

The system accurately detects ransomware activity in real-time by analyzing file behavior features like entropy, size, and renaming patterns. It leverages a trained Random Forest model to distinguish between benign and malicious actions, enabling early threat identification.