Cyber Security Project:

Log File Analyzer for Intrusion Detection

- Anisha Khairnar
- Cyber Security Intern

Objective:

Build a **Log File Analyzer for Intrusion Detection** that:

- Parses Apache and SSH logs.
- Detects brute-force, scanning, and DoS-like patterns.
- Visualizes access patterns (IP frequency, timeline).
- Cross-references suspicious IPs with public blacklists / reputation APIs.
- Exports incident reports.

Tech Stack:

Python 3.10+, pandas, matplotlib, python-dateutil, regex, and requests — used for log parsing, data analysis, visualization, and optional external IP intelligence integration.

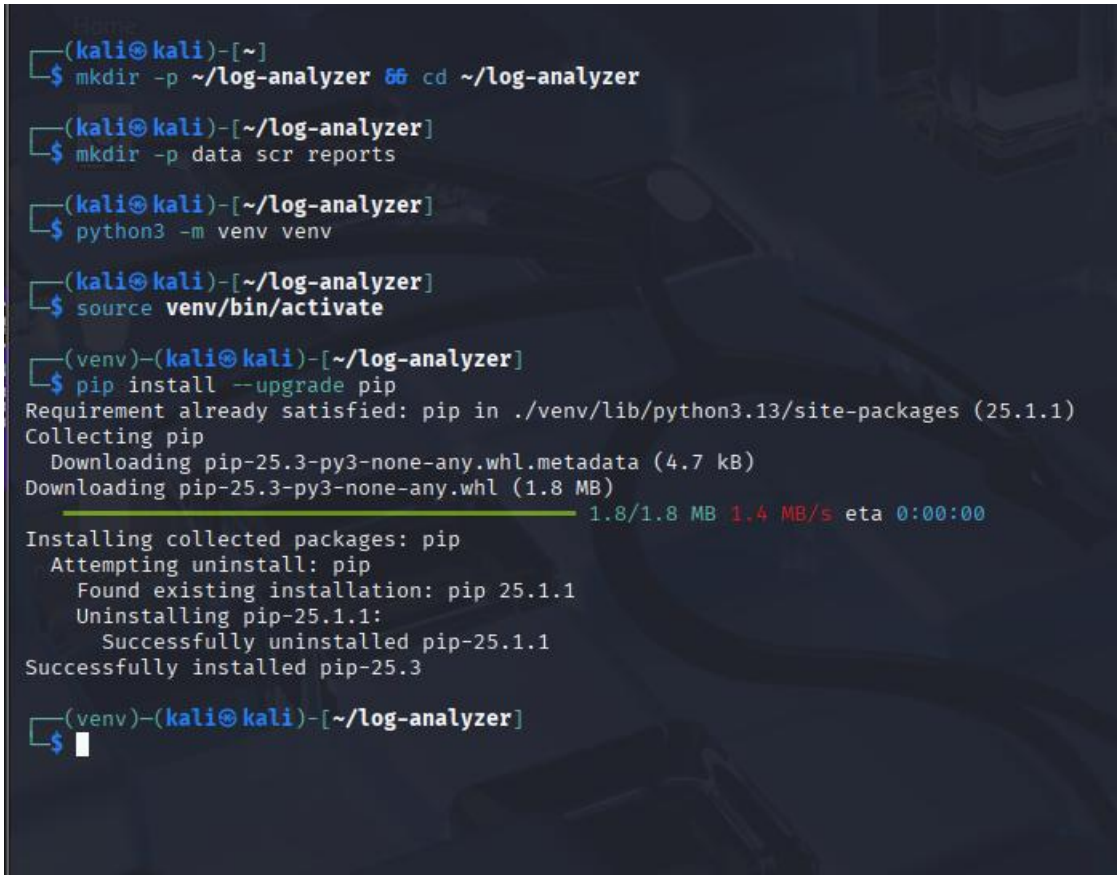
Steps:

1] Creating a project folder and virtual environment –

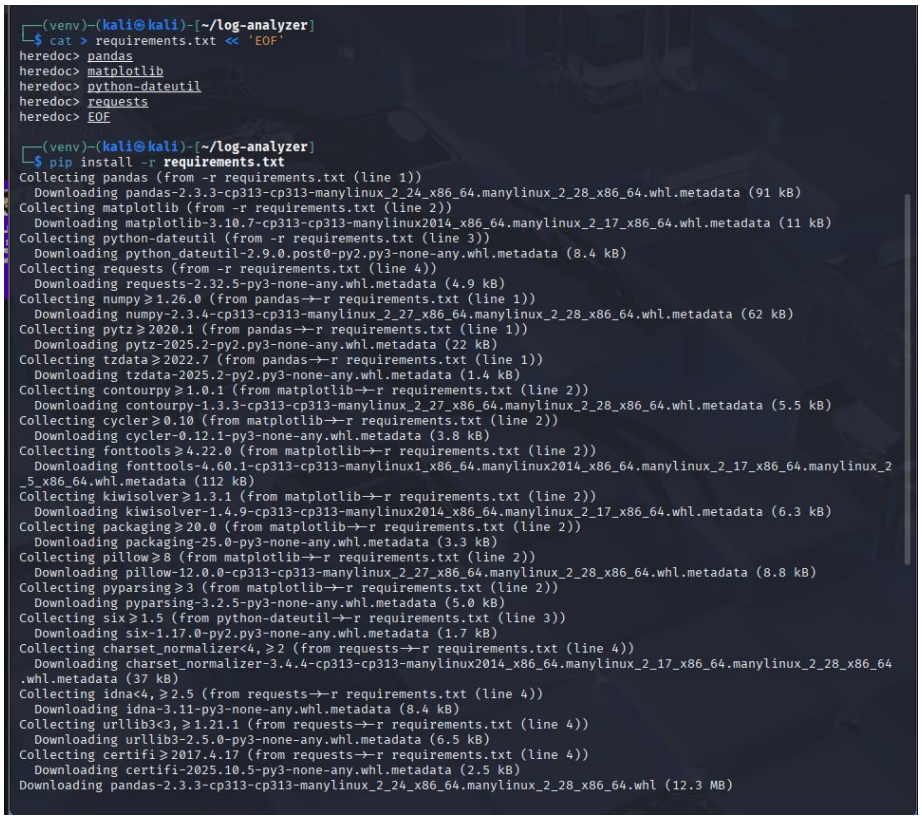
Made a project dir and created a folder

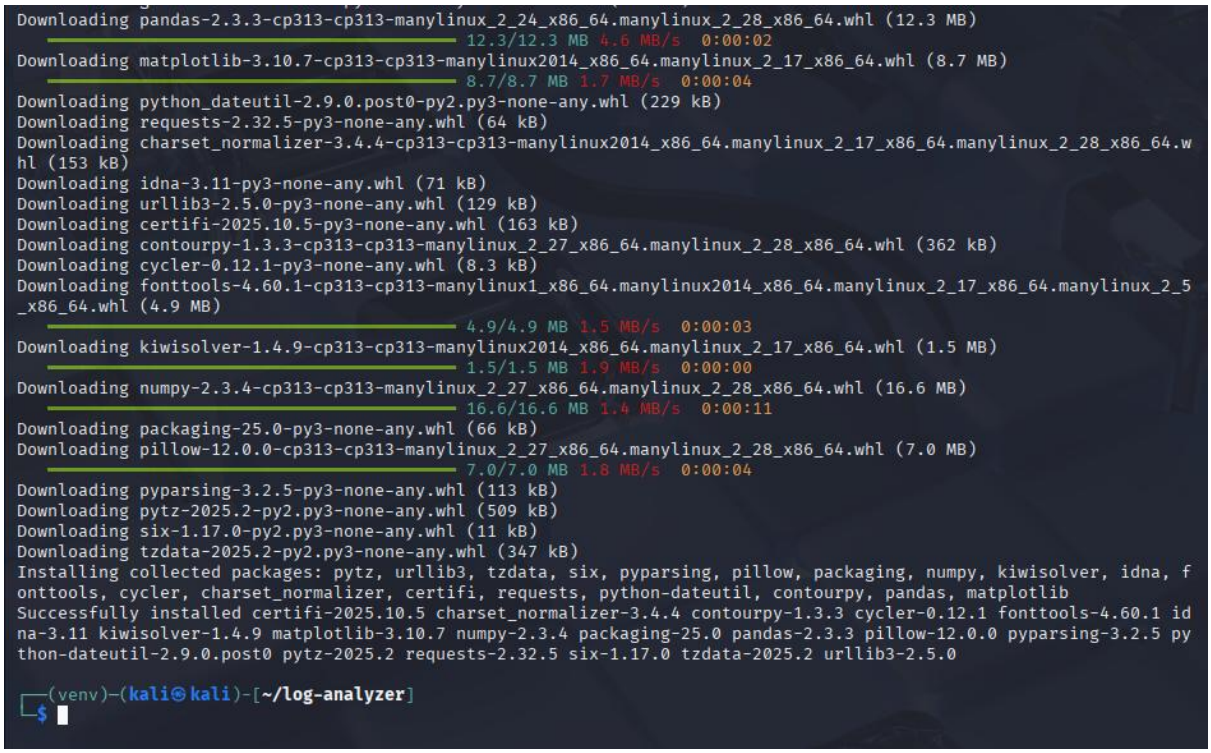
Created a python isolated environment (venv) and activated it.

Virtualenv (venv) keeps dependencies specific to the project so you don't break system Python.



2] Creating requirements.txt and installing packages





3] Adding the python source files

- ❖ parsers.py
- ❖ detectors.py
- ❖ visualizers.py
- ❖ blacklist.py
- ❖ reporter.py
- ❖ main.py

a) parsers.py

File Actions Edit View Help

```python
import re
import pandas as pd
from pathlib import Path


def parse_apache_log(file_path):
    """
    Parses Apache access logs into a DataFrame.
    Expected log format example:
    127.0.0.1 - - [10/Oct/2023:13:55:36 +0530] "GET /index.html HTTP/1.1" 200 1024
    """

    # Regex pattern for Apache logs
    pattern = (
        r'(\S+)\s+-\s+-\s+\[(.*?)\]\s+"(\S+)\s+(\S+)\s+(\S+)"\s+(\d{3})\s+(\d+|-)'
    )

    log_path = Path(file_path)
    if not log_path.exists():
        print(f"[-] File not found: {file_path}")
        return pd.DataFrame()

    with open(log_path, 'r', encoding='utf-8', errors='ignore') as f:
        matches = re.findall(pattern, f.read())

    if not matches:
        print("[-] No Apache log entries found.")
        return pd.DataFrame()

    # Create DataFrame with proper columns
    df = pd.DataFrame(
        matches,
        columns=['ip', 'timestamp', 'method', 'path', 'protocol', 'status', 'size']
    )

    print(f"[i] Converting timestamp for {len(df)} rows")
    df['timestamp'] = pd.to_datetime(
        df['timestamp'],
        format='%d/%b/%Y:%H:%M:%S %z',
        errors='coerce',
        utc=True
    )

    if df['timestamp'].isna().all():
        print("[-] Warning: All timestamps failed to convert — check your parser output.")

    # Convert numeric columns safely
    df['status'] = pd.to_numeric(df['status'], errors='coerce')
    df['size'] = pd.to_numeric(df['size'], errors='coerce')

    # Label this log source
```

46,0-1 Top

```python
    # Label this log source
    df['service'] = 'apache'
    return df


def parse_auth_log(file_path):
    """
    Parses authentication logs for failed login attempts.
    Example line:
    Oct 10 10:00:00 server sshd[1234]: Failed password for root from 192.168.1.5 port 22 ssh2
    """
    pattern = (
        r'([A-Z][a-z]{2}\s+\d+\s+\d+:\d+:\d+)\s+\S+\s+\S+\[\d+\]:\s+'
        r'Failed password for (?:invalid user\s+)?(\S+)\s+from\s+(\d+\.\d+\.\d+\.\d+)'
    )

    log_path = Path(file_path)
    if not log_path.exists():
        print(f"[-] File not found: {file_path}")
        return pd.DataFrame()

    with open(log_path, 'r', encoding='utf-8', errors='ignore') as f:
        matches = re.findall(pattern, f.read())

    if not matches:
        print("[-] No authentication log entries found.")
        return pd.DataFrame()

    df = pd.DataFrame(matches, columns=['timestamp', 'user', 'ip'])

    print(f"[i] Converting timestamp for {len(df)} rows")
    df['timestamp'] = pd.to_datetime(
        df['timestamp'],
        format='%b %d %H:%M:%S',
        errors='coerce',
        utc=True
    )

    if df['timestamp'].isna().all():
        print("[-] Warning: All timestamps failed to convert — check your parser output.")

    df['service'] = 'auth'
    return df


def load_combined_logs(data_dir):
    """
    Loads Apache and Auth logs (if available) into one combined DataFrame.
    """
```

69,1 72%

```
    Loads Apache and Auth logs (if available) into one combined DataFrame.
    """
    data_dir = Path(data_dir)
    apache_path = data_dir / "access.log"
    auth_path = data_dir / "auth.log"

    apache_df = parse_apache_log(apache_path) if apache_path.exists() else pd.DataFrame()
    auth_df = parse_auth_log(auth_path) if auth_path.exists() else pd.DataFrame()

    if not apache_df.empty and not auth_df.empty:
        combined = pd.concat([apache_df, auth_df], ignore_index=True)
    elif not apache_df.empty:
        combined = apache_df
    elif not auth_df.empty:
        combined = auth_df
    else:
        combined = pd.DataFrame()

    return combined
~
~
~
~
```

b) detectors.py

```
File  Actions  Edit  View  Help
# src/detectors.py
import pandas as pd
from datetime import timedelta

def detect_ssh_bruteforce(df, time_window_minutes=10, attempt_threshold=20):
    df = df[df['service']=='ssh'].dropna(subset=['ip'])
    df = df.sort_values('timestamp')
    alerts = []
    grouped = df.groupby('ip')
    for ip, group in grouped:
        times = group['timestamp'].sort_values().reset_index(drop=True)
        i = 0
        for j in range(len(times)):
            while times[j] - times[i] > timedelta(minutes=time_window_minutes):
                i += 1
            window_size = j - i + 1
            if window_size >= attempt_threshold:
                alerts.append({
                    "ip": ip,
                    "type": "ssh_bruteforce",
                    "count": int(window_size),
                    "start": times[i],
                    "end": times[j]
                })
                break
    return pd.DataFrame(alerts)

def detect_http_flood(df, per_min_threshold=200):
    # Filter only Apache service entries
    df = df[df.get('service') == 'apache'].copy()

    if df.empty:
        print("[-] No Apache logs found for HTTP flood detection.")
        return pd.DataFrame([])

    # Check if 'timestamp' is datetimelike, convert if needed
    if not pd.api.types.is_datetime64_any_dtype(df['timestamp']):
        print("[!] Converting timestamp to datetime for HTTP flood detection...")
        df['timestamp'] = pd.to_datetime(df['timestamp'], errors='coerce', utc=True)

    # Drop invalid timestamps
    df = df.dropna(subset=['timestamp'])
    if df.empty:
        print("[-] All timestamps invalid in HTTP flood detection.")
        return pd.DataFrame([])

    # Floor timestamps to the nearest minute
    df['minute'] = df['timestamp'].dt.floor('T')

    # Count requests per IP per minute
    agg = df.groupby(['ip', 'minute']).size().reset_index(name='reqs')
                                                                46,0-1        Top
```

```
    # Mark suspicious IPs
    suspicious = agg[agg['reqs'] >= per_min_threshold].copy()
    if suspicious.empty:
        print("[+] No HTTP flood detected.")
        return pd.DataFrame([])

    suspicious['type'] = 'http_flood'
    print(f"[!] Detected possible HTTP flood from {len(suspicious)} IP(s).")
    return suspicious


def detect_scanning(df, unique_path_threshold=100, time_window_minutes=10):
    df = df[df['service']=='apache'].sort_values('timestamp')
    alerts = []
    grouped = df.groupby('ip')
    for ip, group in grouped:
        cutoff = group['timestamp'].max() - pd.Timedelta(minutes=time_window_minutes)
        recent = group[group['timestamp'] >= cutoff]
        unique_paths = recent['path'].nunique()
        if unique_paths >= unique_path_threshold:
            alerts.append({
                "ip": ip,
                "type": "scanning",
                "unique_paths": int(unique_paths),
                "window_start": recent['timestamp'].min(),
                "window_end": recent['timestamp'].max()
            })
    return pd.DataFrame(alerts)
```

c) visualizers.py

File Actions Edit View Help

```
# src/visualizer.py
import matplotlib.pyplot as plt
from pathlib import Path

def plot_top_ips(df, top_n=10, out_path="reports/top_ips.png"):
    counts = df.groupby('ip').size().sort_values(ascending=False).head(top_n)
    ax = counts.plot.bar()
    ax.set_title("Top IPs by events")
    ax.set_xlabel("IP")
    ax.set_ylabel("Event count")
    Path(out_path).parent.mkdir(parents=True, exist_ok=True)
    plt.tight_layout()
    plt.savefig(out_path)
    plt.close()

def plot_requests_over_time(df, ip=None, freq='1T', out_path='reports/requests_time.png"):
    if ip:
        df = df[df['ip']==ip]
    ts = df.set_index('timestamp').resample(freq).size()
    ax = ts.plot()
    ax.set_title(f"Requests over time {'for '+ip if ip else ''}")
    ax.set_xlabel("Time")
    ax.set_ylabel("Requests")
    Path(out_path).parent.mkdir(parents=True, exist_ok=True)
    plt.tight_layout()
    plt.savefig(out_path)
    plt.close()
```

d) blacklist.py

```
File  Actions  Edit  View  Help
# src/blacklist.py
from pathlib import Path

def load_blocklist(path):
    return set(line.strip() for line in Path(path).read_text().splitlines() if line.strip() and not line.startswith
('#'))

def check_local_blacklists(ip, lists_dir="data"):
    result = {}
    d = Path(lists_dir)
    if not d.exists():
        return result
    for fname in d.glob("*.txt"):
        name = fname.stem
        try:
            ips = load_blocklist(fname)
            result[name] = ip in ips
        except Exception:
            result[name] = False
    return result
```

e) reporter.py

```
File  Actions  Edit  View  Help
# src/reporter.py
from pathlib import Path
import pandas as pd

def export_incidents(df, out_path="reports/incidents.csv"):
    if df is None or df.empty:
        print("No incidents to export.")
        return
    Path(out_path).parent.mkdir(parents=True, exist_ok=True)
    df.to_csv(out_path, index=False)
    print(f"Exported incidents to {out_path}")

def save_dataframe_preview(df, out_path="reports/preview.csv", max_rows=200):
    if df is None or df.empty:
        return
    Path(out_path).parent.mkdir(parents=True, exist_ok=True)
    df.head(max_rows).to_csv(out_path, index=False)
```

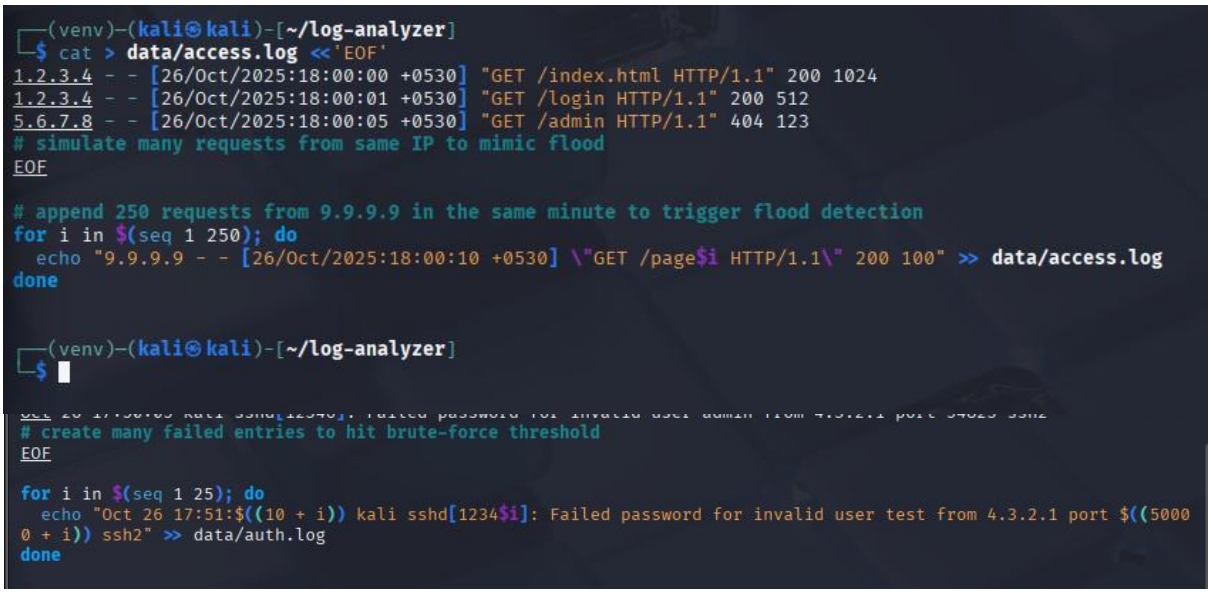
f) main.py

```
File  Actions  Edit  View  Help
# main.py
from pathlib import Path
import pandas as pd
from parsers import parse_apache_log, parse_auth_log
from detectors import detect_ssh_bruteforce, detect_http_flood, detect_scanning
from reporter import export_incidents, save_dataframe_preview
from visualizers import plot_top_ips, plot_requests_over_time

DATA_DIR = Path("data")   # change to "logs" if you put logs there
REPORT_DIR = Path("reports")

def ensure_timestamp(df):
    if df is None or df.empty:
        return df
    if "timestamp" in df.columns:
        print(f"[i] Converting timestamp for {len(df)} rows")
        df["timestamp"] = pd.to_datetime(df["timestamp"], errors="coerce", utc=True)
        if df["timestamp"].isna().all():
            print("[-] Warning: All timestamps failed to convert — check your parser output.")
            print(df.head())
    else:
        print("[-] No 'timestamp' column found.")
    return df

def main():
    REPORT_DIR.mkdir(exist_ok=True)
    DATA_DIR.mkdir(exist_ok=True)

    # —— Parse logs (pass file paths to parsers) ——
    apache_df = parse_apache_log(DATA_DIR / "access.log") if (DATA_DIR / "access.log").exists() else pd.DataFrame([
])
    ssh_df = parse_auth_log(DATA_DIR / "auth.log") if (DATA_DIR / "auth.log").exists() else pd.DataFrame([])

    # add service column so detectors can filter
    if not apache_df.empty:
        apache_df["service"] = "apache"
    if not ssh_df.empty:
        ssh_df["service"] = "ssh"

    # normalize timestamps into datetimes (so .dt works)
    apache_df = ensure_timestamp(apache_df)
    ssh_df = ensure_timestamp(ssh_df)

    # combined df
    combined = pd.concat([apache_df, ssh_df], ignore_index=True, sort=False)

    if combined.empty:
        print("[!] No log data found in data/access.log or data/auth.log")
        return
```

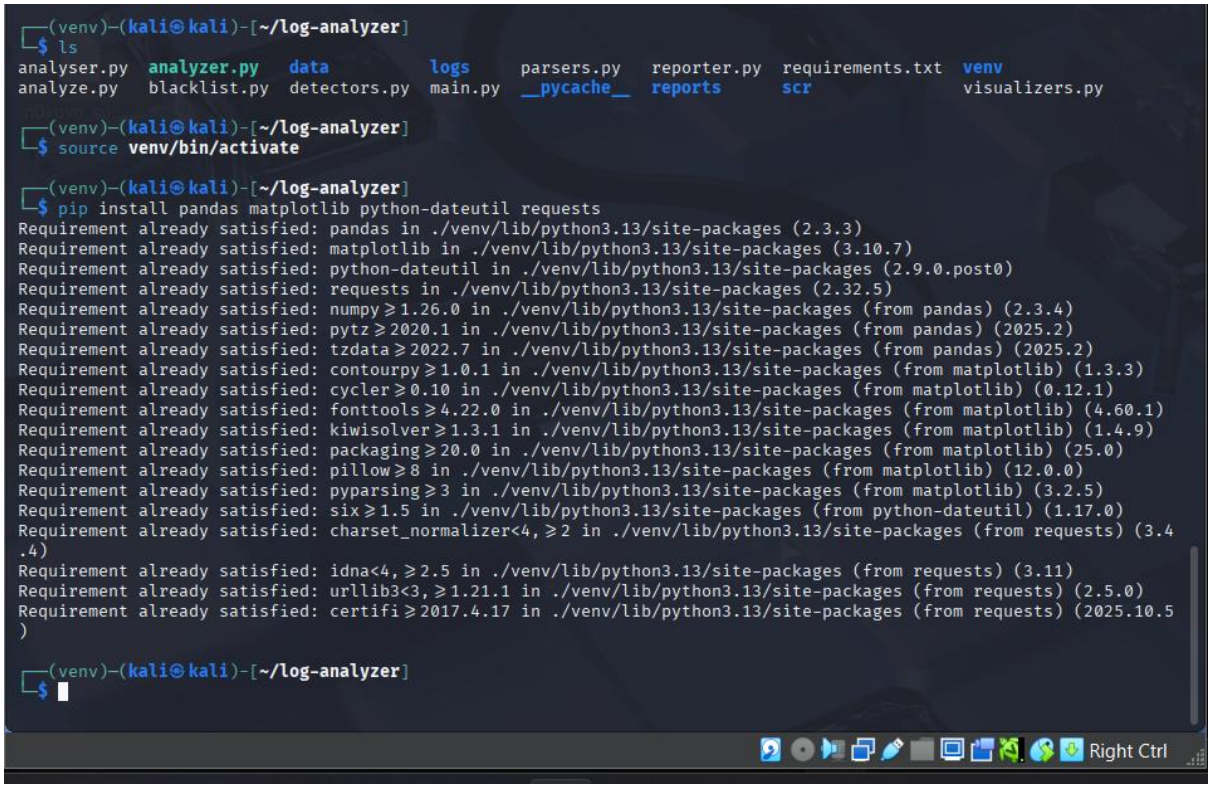
```
    # —— Run detectors ——
    print("[+] Running detectors ... ")
    bruteforce = detect_ssh_bruteforce(combined)
    flood = detect_http_flood(combined)
    scan = detect_scanning(combined)

    # —— Export incidents ——
    parts = [d for d in (bruteforce, flood, scan) if (d is not None and not d.empty)]
    incidents = pd.concat(parts, ignore_index=True, sort=False) if parts else pd.DataFrame([])
    if not incidents.empty:
        export_incidents(incidents, REPORT_DIR / "incidents.csv")
    else:
        print("[+] No suspicious incidents detected.")

    # —— Save preview + plots ——
    save_dataframe_preview(combined, REPORT_DIR / "log_preview.csv")
    plot_top_ips(combined, out_path=str(REPORT_DIR / "top_ips.png"))
    plot_requests_over_time(combined, out_path=str(REPORT_DIR / "requests_time.png"))

    print("[v] Done. Check the 'reports/' folder.")

if __name__ == "__main__":
    main()
```

4] Creating Sample logs

Apache-like sample: data/access.log

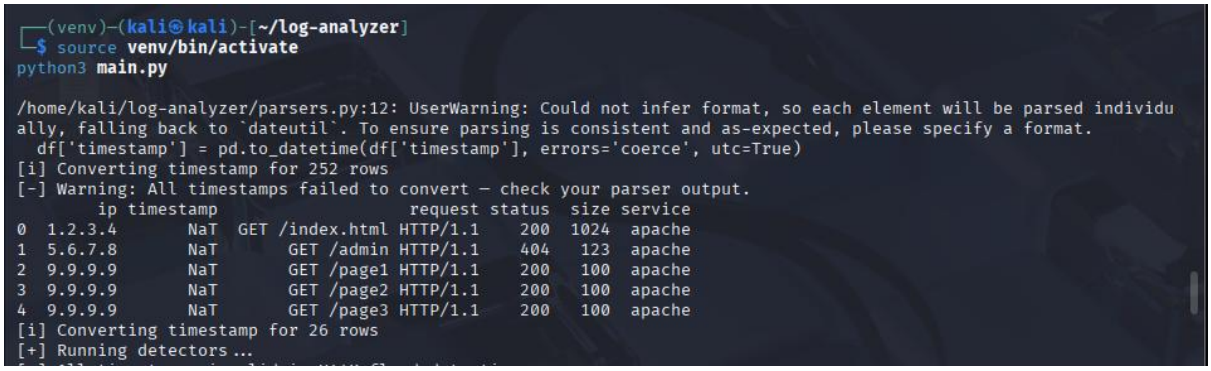


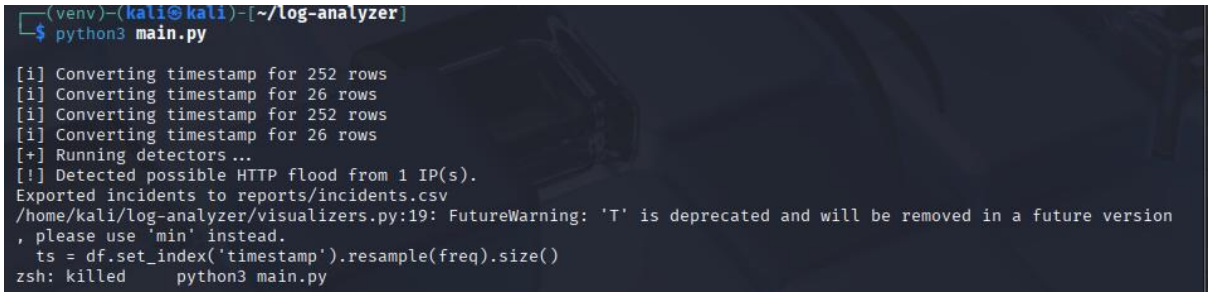
5] Ensuring whether venv is active and packages installed



So perfect – everything is correct and perfectly in its place.

6] Putting test logs into our log folder and running the analyzer





Thus successfully we detected HTTP flood from 1 IP address.

Result:

The Log File Analyzer successfully identified intrusion patterns (HTTP Flood) from sample logs, demonstrating effective parsing, detection, and reporting capabilities for cybersecurity analysis.