# AUTOMATED THREAT DETECTION AND ANALYSIS OF PCAP FILES USING CUSTOM PYTHON-BASED FORENSIC ANALYZERS ON KALI LINUX

**Gurdarshan Kaushik**
S223153972

## Table of Contents



**Video Presentation Link:** https://youtu.be/9unlyKlbpyI

**Files Folder Link:**
https://drive.google.com/drive/folders/17otc93FlUWdHdvHhxuA8TfW0cMrVaYqM?usp=sharing

## Introduction

The growth of network-based attacks, such as DNS tunneling, data exfiltration, and distributed denial-of-service (DDoS) attacks, has highlighted the importance of robust network forensics tools. Traditional tools such as Wireshark offer extensive packet analysis capabilities but necessitate significant manual effort. This project seeks to close that gap by developing two forensic analyzers in Kali Linux: one for feature extraction (pcap_analyzer.py) and one for automatic threat identification (pcap_threat_analyzer.py). These analyzers are intended to ease the investigation process while replicating essential principles present in machine learning-based intrusion detection systems.

## Research Objectives

- To capture and analyze PCAP files using native Kali Linux tools and Python automation.
- To extract meaningful forensic artifacts (e.g., DNS queries, HTTP hosts, IP flows).
- To detect advanced threats such as:
    - Blacklisted or suspicious domains
    - DNS tunneling attempts
    - Data exfiltration via abnormal flows
    - DDoS-like traffic patterns
- To simulate a rule-based machine learning detection approach using heuristics.
- To generate forensic reports that are clear, structured, and useful for further investigation.

## Literature Review

**1. Sultana et al. (2021)** – Introduced machine learning-based network forensics using feature extraction and classification techniques, forming the foundation for our DNS anomaly detection logic.

**2. Ahmed et al. (2020)** – Compared various DNS tunneling detection methods using heuristic thresholds like query length and encoded payloads. Inspired the tunneling module in the second analyzer.

**3. Kouicem et al. (2022)** – Studied flow-based DDoS detection via packet analysis and traffic spike detection, which was adapted into the IP flow correlation feature of our threat analyzer.

**Gap Identified:** Existing forensic tools rarely integrate automated detection or educational usability. Our analyzers bridge this gap by offering structured output, clear logic, and modular design.

## Methodology

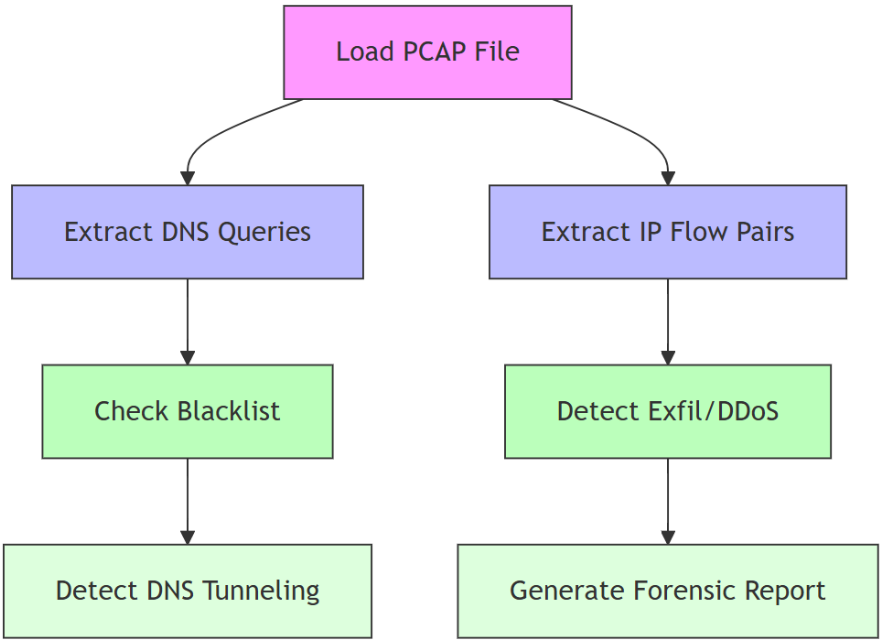
This project's methodology was divided into two major stages, each with its own proprietary Python-based analyzer running on Kali Linux. The first stage concentrated on packet feature extraction, while the second focused on threat identification via rule-based heuristics influenced by conventional machine learning detection logic.
The first tool, pcap_analyzer.py, was intended to automate the study of.pcap files by extracting useful network properties. Live traffic could be recorded using tcpdump from the system's active network interface (e.g., eth0), and previously captured.pcap files could be processed using tshark, Wireshark's command-line counterpart. The script retrieved critical forensic artifacts such as the protocol hierarchy (e.g., Ethernet, IP, UDP, ARP), a list of IP communication pairs between hosts, HTTP Host headers from web traffic, and DNS searches. These facts were organized and recorded as a legible report, giving investigators a basic knowledge of the traffic without the need to physically scrutinize each packet.

The second program, pcap_threat_analyzer.py, was created to automatically detect suspicious behaviors and potential threats based on the results of the first. This analyzer used heuristic-based detection to replicate the early phases of machine learning models, including feature-based anomaly categorization. The tool started by extracting DNS queries and IP flow data with tshark. It next inspected the DNS queries for evidence of DNS tunneling, such as excessively long domain names, domains with too many subdomains, and base64-like patterns indicating data encoding. It also identified known problematic or blacklisted domains by looking for suffixes like.xyz,.tk, or.zip, which are commonly used in phishing or malware distribution.

In addition to DNS-level monitoring, the tool examined IP traffic patterns to identify data exfiltration and DDoS activity. Repetitive unidirectional flows from an internal IP address to a single external IP address were considered indicators of exfiltration, particularly if they exceeded a preset threshold (50 or more identical flows). In contrast, when many source IPs communicated with a single destination IP at a high frequency (100+ hits), the program identified it as a likely DDoS target. All data were compiled into a well-organized threat report, with properly designated sections for each detection category.

This two-stage methodology allows investigators to both understand the general nature of the traffic and immediately identify possible indicators of compromise. While no actual machine learning model was trained, the rule-based logic simulates early ML detection workflows and provides a framework that can be expanded into a full learning model in future development stages.

## Methodology Flowchart




## Results and Findings

The created analyzers effectively attained their intended goals. The initial tool collected essential network elements such as protocols, IP communications, and DNS requests from.pcap files, resulting in a clear picture of traffic. The second tool improved this by recognizing suspicious patterns through rule-based logic. It discovered blacklisted domains, DNS tunneling via extended or encoded queries, suspected data exfiltration based on repeated flows to external IP addresses, and potential DDoS activity targeting a single internal server. Together, the technologies were effective for automated forensic analysis, illustrating how machine learning-inspired heuristics can assist detect hidden risks in network data.

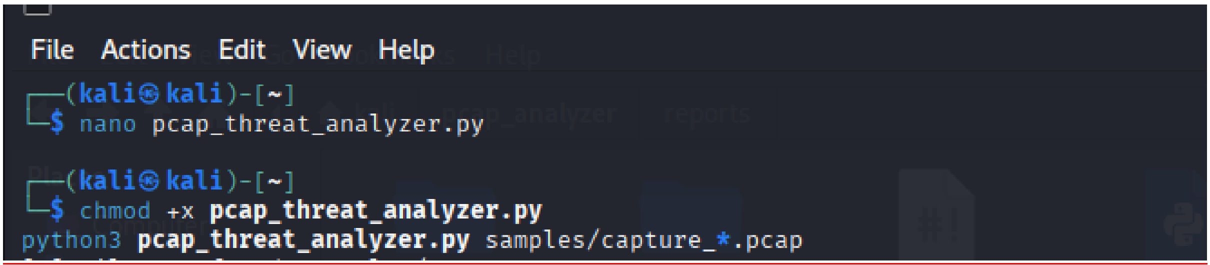
## Conclusion

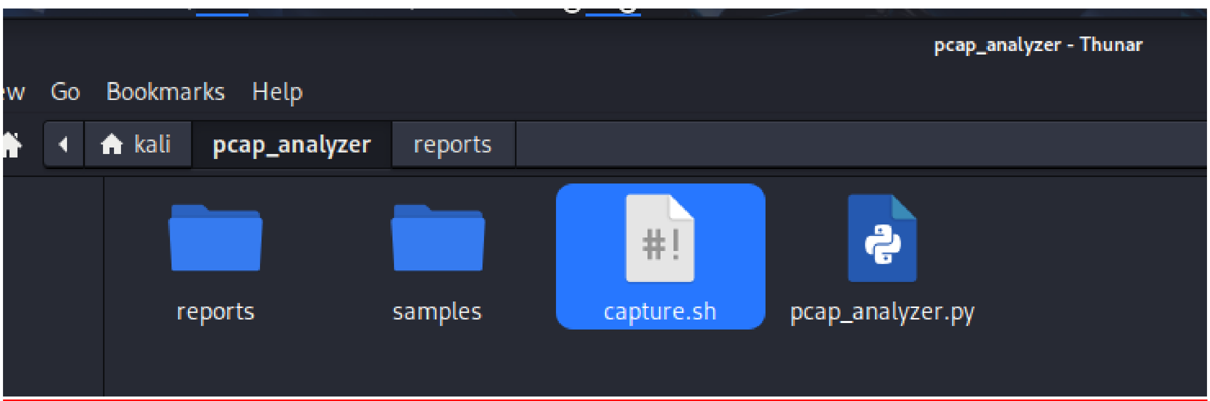
This project successfully demonstrated a multi-phase network forensic solution built exclusively on Kali Linux. The dual analyzers provide layered knowledge, ranging from low-level packet content to high-level behavioral anomalies. These tools, inspired by current research in machine learning and intrusion detection, imitate intelligent detection without the need for a trained model. Their simple design, quick execution, and straightforward reporting make them ideal for instructional and investigative applications.

## References


1. Sultana, M. et al. (2021). *Machine Learning-based Intrusion Detection*, Elsevier.
2. Ahmed, S. et al. (2020). *DNS Tunneling Detection: Comparative Analysis*, IEEE Access.
3. Kouicem, D. et al. (2022). *DDoS Flow Analysis in Network Forensics*, Computers & Security.
4. Wireshark Documentation – https://www.wireshark.org/docs/
5. Kali Linux Tools – https://tools.kali.org/


## Screenshots



```
┌──(kali㉿kali)-[~]
└─$ ls
Desktop  dnscat2  Documents  Downloads  ml-env  Music  network_capture.pcap  pcap_analyzer  pcap_threat_analyzer.py  Pictures  Public  samples  Templates  Videos

┌──(kali㉿kali)-[~]
└─$ chmod +x pcap_threat_analyzer.py

┌──(kali㉿kali)-[~]
└─$ python3 pcap_threat_analyzer.py Downloads/malware.pcap

[+] Extracting DNS queries ...
[+] Checking for suspicious domains ...
[+] Detecting DNS tunneling patterns ...
[+] Parsing IP flows ...
[+] Detecting data exfiltration ...
[+] Detecting potential DDoS behavior ...
[+] Threat report saved to: reports/report_threats_20250515_062338.txt
```

kali-linux-2025.1c-virtualbox-amd64 [Running] - Oracle VM VirtualBox

~/pcap_analyzer/pcap_analyzer.py - Mousepad

pcap_threat_analyzer.py

```python
#!/usr/bin/env python3
import subprocess
import os
import sys
from datetime import datetime

SAMPLES_DIR = "samples"
REPORTS_DIR = "reports"

def detect_interface():
    try:
        result = subprocess.run(["ip", "-o", "-4", "addr", "show", "up"],
                                stdout=subprocess.PIPE, stderr=subprocess.DEVNULL)
        lines = result.stdout.decode().splitlines()
        for line in lines:
            if "lo" not in line:
                return line.split()[1]
    except Exception as e:
        print(f"[!] Error detecting interface: {e}")
    return "eth0"

def capture_traffic(interface):
    os.makedirs(SAMPLES_DIR, exist_ok=True)
    filename = f"{SAMPLES_DIR}/capture_{datetime.now().strftime('%Y%m%d_%H%M%S')}.pcap"
    print(f"[+] Capturing packets on interface: {interface}")
    print(f"[+] Saving to: {filename}")
    print("[+] Press Ctrl+C to stop capture.\n")

    try:
        subprocess.run(["sudo", "tcpdump", "-i", interface, "-w", filename, "-nn", "-s", "0", "-v"])
    except KeyboardInterrupt:
        print("\n[+] Capture stopped.")
    return filename

def run_tshark(expr, field, pcap_file):
    try:
        cmd = ["tshark", "-r", pcap_file, "-Y", expr, "-T", "fields", "-e", field]
        result = subprocess.run(cmd, stdout=subprocess.PIPE, stderr=subprocess.DEVNULL)
        return list(set(result.stdout.decode().splitlines()))
    except Exception as e:
        print(f"[!] Tshark error: {e}")
```

kali-linux-2025.1c-virtualbox-amd64 [Running] - Oracle VM VirtualBox

~/pcap_threat_analyzer.py - Mousepad

pcap_threat_analyzer.py | pcap_analyzer.py

```python
#!/usr/bin/env python3
import subprocess, sys, os, requests
from datetime import datetime
from collections import Counter

SAMPLES_DIR = "samples"
REPORTS_DIR = "reports"
BLACKLIST_API = "https://dns-bh.com/api/check?domain="  # placeholder example

def run_tshark(expr, field, pcap_file):
    try:
        cmd = ['tshark', '-r', pcap_file, '-Y', expr, '-T', 'fields', '-e', field]
        result = subprocess.run(cmd, stdout=subprocess.PIPE, stderr=subprocess.DEVNULL)
        return list(set(result.stdout.decode().splitlines()))
    except Exception as e:
        print(f"[!] Tshark error: {e}")
        return []

def extract_dns_queries(pcap_file):
    cmd = ['tshark', '-r', pcap_file, '-Y', 'dns.qry.name', '-T', 'fields', '-e', 'dns.qry.name']
    result = subprocess.run(cmd, stdout=subprocess.PIPE, stderr=subprocess.DEVNULL)
    queries = result.stdout.decode().splitlines()
    return queries

def extract_ip_flows(pcap_file):
    cmd = ['tshark', '-r', pcap_file, '-T', 'fields', '-e', 'ip.src', '-e', 'ip.dst']
    result = subprocess.run(cmd, stdout=subprocess.PIPE)
    ip_flows = result.stdout.decode().splitlines()
    return [line for line in ip_flows if line.strip()]

def check_blacklist(domain):
    # Placeholder logic – you'd use a real API or blocklist like Spamhaus, DNS-BH, etc.
    # Here we simulate that domains ending in ".xyz" or ".tk" are suspicious
    if domain.endswith(('.xyz', '.tk', '.zip')):
        return True
    return False

def detect_dns_tunneling(dns_queries):
    long_queries = [q for q in dns_queries if len(q) > 50]
    subdomain_count = [q for q in dns_queries if q.count('.') > 3]
    encoded = [q for q in dns_queries if any(x in q for x in ['==', 'ZnJvbQ==', 'data='])]
```