Introduction

In modern cybersecurity, detecting and preventing threats in real-time is critical to safeguarding systems from cyberattacks. This project integrates multiple security mechanisms, including Real-Time Log Monitoring, Rate Limiting, Live Attack Dashboards, AI-Powered Anomaly Detection, Blockchain for Tamper-Proof Logs, MITRE ATT&CK Framework Mapping, and Threat Intelligence Feeds, to enhance cybersecurity defenses.

The Real-Time Log Monitor continuously collects, analyzes, and visualizes logs to detect suspicious activities instantly. A Rate Limiter helps prevent brute-force attacks by restricting excessive login attempts, reducing the risk of credential stuffing. The Live Attack Dashboard (Matplotlib) provides graphical insights into ongoing cyber threats, helping security teams respond efficiently.

To enhance threat detection, an AI-Powered Anomaly Detection Dashboard leverages machine learning to identify unusual patterns and zero-day attacks. Additionally, Blockchain for Tamper-Proof Logs ensures log integrity, preventing unauthorized modifications and maintaining a verifiable record of system activities.

Further strengthening security, MITRE ATT&CK Framework Mapping aligns detected threats with well-known adversarial techniques, aiding in threat classification and response planning. Moreover, a Threat Intelligence Feed integrates real-time external threat data, allowing proactive defense against emerging cyber threats by blocking known malicious entities.

By combining these advanced security mechanisms, this project creates a robust, intelligent, and proactive cybersecurity solution capable of detecting, analyzing, and mitigating cyber threats efficiently.

INDEX

IoT Honeypot: Simulating and Analyzing Attacks

The Internet of Things (IoT) has revolutionized modern computing by enabling interconnected smart devices to collect, process, and exchange data. However, the rapid proliferation of IoT devices has also introduced significant security vulnerabilities due to weak authentication mechanisms, unpatched firmware, and insecure communication protocols. Attackers frequently exploit these weaknesses to launch large-scale cyberattacks, such as botnet-driven Distributed Denial of Service (DDoS) attacks, data breaches, and ransomware infections.

Uses of Honeypot:

- Threat Intelligence Gathering
- Vulnerability Research
- Malware Analysis
- Attack Attribution
- Security Training
- Defensive Strategy Development

Types of Attacks analyzed using IoT Honeypots:

- Brute Force & Credential Suffering Attacks
- Malware and Botnet Infections
- Exploit-Based Attacks

- Protocol Exploits
- Denial Of Service (DoS/DDoS) Attacks
- Ransomware and Data Exfiltration
- Zero Day Attack Detection

```python
import (module) threading
import threading
import time
from IPython.display import clear_output

# Configuration
HOST = "localhost"  # Use "0.0.0.0" for remote access (not recommended in notebooks)
PORT = 8080
REAL_PASSWORD = "admin123"
LOG_FILE = "attack_logs.txt"

# Initialize log file
with open(LOG_FILE, "w") as f:
    f.write("")  # Clear previous logs

def start_fake_device():
    server = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
    server.bind((HOST, PORT))
    server.listen(5)
    print(f"[HONEYPOT] Running on {HOST}:{PORT} (Waiting for attackers...)")

    while True:
        client, addr = server.accept()
        client.send(b"Welcome to FakeIoT Device. Enter password: ")
        password_attempt = client.recv(1024).decode().strip()

        if password_attempt == REAL_PASSWORD:
            client.send(b"ACCESS GRANTED (but this is a trap!)\n")
        else:
            client.send(b"ACCESS DENIED (logging your attempt...)\n")
            with open(LOG_FILE, "a") as f:
                f.write(f"{time.ctime()} | Attack from {addr[0]} | Password tried: '{password_attempt}'\n")
        client.close()

# Start honeypot in background
honeypot_thread = threading.Thread(target=start_fake_device)
honeypot_thread.daemon = True
honeypot_thread.start()
```

Attacker Simulation

In an IoT honeypot, attacker simulation involves mimicking real-world hacking attempts to study how cybercriminals exploit vulnerable devices. One of the most common attack methods tested is brute-force login attacks, where hackers systematically try different username and password combinations to gain unauthorized access.

How it works:

- The honeypot pretends to be a real IoT device(e.g. a smart camera , router or a thermostat) with open login ports like SSH or Telnet.
- The attacker attempts to break in.
- The honeypot logs each attempt , noting:
  - Failed logins
  - Successful logins
  - Any follow up malicious activity
- The honeypot records IP address of attackers, Password list used in brute-forcing, exploit techniques etc.
- This data helps security researchers identify the trending attack methods, improve IoT device security, detect botnets scanning for vulnerable devices.

```python
def simulate_attacker():
    target_ip = "localhost"
    target_port = 8080
    common_passwords = ["admin", "password", "123456", "iot", "root"]

    print("[ATTACKER] Starting brute-force attack...")
    for pwd in common_passwords:
        try:
            s = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
            s.connect((target_ip, target_port))
            s.recv(1024)  # Get password prompt
            s.send(pwd.encode())
            response = s.recv(1024).decode()
            print(f"Tried '{pwd}': {response.strip()}")
            s.close()
        except Exception as e:
            print(f"Failed to attack: {str(e)}")

simulate_attacker()
```

```
[ATTACKER] Starting brute-force attack...
Tried 'admin': ACCESS DENIED (logging your attempt...)
Tried 'password': ACCESS DENIED (logging your attempt...)
Tried '123456': ACCESS DENIED (logging your attempt...)
Tried 'iot': ACCESS DENIED (logging your attempt...)
Tried 'root': ACCESS DENIED (logging your attempt...)
```

This script demonstrates a basic brute-force attack, a method used by attackers to gain unauthorized access by systematically trying multiple passwords. In this case, the attack targets a local server running on localhost:8080.

View Attack Logs

View Attack Logs refers to the process of accessing, monitoring, and analyzing logs that record potential cyber attacks or security threats on a system, network, or application. These logs contain crucial information about malicious activities, including unauthorized access attempts, brute force attacks, SQL injections, DDoS attacks, and other security incidents.

This feature is essential in cybersecurity to:

- Detect and investigate security threats.
- Identify attack patterns and sources.
- Take preventive measures to strengthen system security.
- Generate reports for auditing and compliance.

It is commonly used in Intrusion Detection Systems (IDS), Firewalls, SIEM tools (Security Information and Event Management), and server log analysis tools.

```python
def show_attack_logs():
    try:
        with open(LOG_FILE, "r") as f:
            logs = f.read()
            if logs:
                print("=== ATTACK LOGS ===")
                print(logs)
            else:
                print("No attacks logged yet.")
    except FileNotFoundError:
        print("Log file not found. Run the honeypot first!")

show_attack_logs()  # Run this cell repeatedly to see new attacks
```

Output:

```
=== ATTACK LOGS ===
Fri Mar 28 14:39:31 2025 | Attack from 127.0.0.1 | Password tried: 'admin'
Fri Mar 28 14:39:31 2025 | Attack from 127.0.0.1 | Password tried: 'password'
Fri Mar 28 14:39:31 2025 | Attack from 127.0.0.1 | Password tried: '123456'
Fri Mar 28 14:39:31 2025 | Attack from 127.0.0.1 | Password tried: 'iot'
Fri Mar 28 14:39:31 2025 | Attack from 127.0.0.1 | Password tried: 'root'
```

Real Time Log Monitor

A Real-Time Log Monitor is a system designed to continuously track, analyze, and display logs as they are generated. It helps in identifying security threats, system errors, and operational performance issues in real-time. This tool is essential for cybersecurity, DevOps, and IT operations, allowing instant detection and response to potential threats.

Objectives:

- To monitor logs in real-time and detect anomalies.
- To provide automated alerts and notifications.
- To visualize logs for easier analysis.
- To improve incident response time

Key Features:

- Live Log Tracking: Monitors system and application logs in real-time.
- Automated Alerts: Sends notifications for suspicious activities.
- Filtering & Searching: Enables quick retrieval of relevant logs.
- Visualization Dashboards: Displays logs using charts and graphs.
- Integration with SIEM Tools: Compatible with Splunk, ELK Stack, Graylog, etc.
- Threat Detection: Identifies cyber threats like brute force attacks, SQL injections, and unauthorized access attempts.

Implementation Steps:

- Collection: Gather logs from servers, applications, and network devices.
- Log Parsing & Processing: Convert raw logs into structured data.
- Real-Time Analysis: Use machine learning or rule-based detection to identify threats.
- Alert System: Set up automated triggers for anomalies.
- Visualization & Reporting: Display insights on dashboards for easy monitoring.

Use Cases:

- Cybersecurity Monitoring: Detects and prevents unauthorized access attempts.
- System Health Monitoring: Tracks system errors and performance bottlenecks.
- Compliance Auditing: Logs user activities for regulatory compliance.

```python
from IPython.display import display, HTML
import ipywidgets as widgets

log_output = widgets.Output()

def monitor_logs():
    with open(LOG_FILE, "r") as f:
        logs = f.read()
    with log_output:
        clear_output(wait=True)
        print(logs if logs else "Waiting for attacks...")

display(HTML("<h3>Live Attack Monitor</h3>"))
display(log_output)

# Auto-refresh every 3 seconds
import threading
def auto_refresh():
    while True:
        monitor_logs()
        time.sleep(3)

refresh_thread = threading.Thread(target=auto_refresh)
refresh_thread.daemon = True
refresh_thread.start()
```

Output:

```
Fri Mar 28 14:39:31 2025 | Attack from 127.0.0.1 | Password tried: 'admin'
Fri Mar 28 14:39:31 2025 | Attack from 127.0.0.1 | Password tried: 'password'
Fri Mar 28 14:39:31 2025 | Attack from 127.0.0.1 | Password tried: '123456'
Fri Mar 28 14:39:31 2025 | Attack from 127.0.0.1 | Password tried: 'iot'
Fri Mar 28 14:39:31 2025 | Attack from 127.0.0.1 | Password tried: 'root'

Fri Mar 28 14:39:31 2025 | Attack from 127.0.0.1 | Password tried: 'admin'
Fri Mar 28 14:39:31 2025 | Attack from 127.0.0.1 | Password tried: 'password'
Fri Mar 28 14:39:31 2025 | Attack from 127.0.0.1 | Password tried: '123456'
Fri Mar 28 14:39:31 2025 | Attack from 127.0.0.1 | Password tried: 'iot'
Fri Mar 28 14:39:31 2025 | Attack from 127.0.0.1 | Password tried: 'root'
```

```
Fri Mar 28 14:39:31 2025 | Attack from 127.0.0.1 | Password tried: 'admin'
Fri Mar 28 14:39:31 2025 | Attack from 127.0.0.1 | Password tried: 'password'
Fri Mar 28 14:39:31 2025 | Attack from 127.0.0.1 | Password tried: '123456'
Fri Mar 28 14:39:31 2025 | Attack from 127.0.0.1 | Password tried: 'iot'
Fri Mar 28 14:39:31 2025 | Attack from 127.0.0.1 | Password tried: 'root'

Fri Mar 28 14:39:31 2025 | Attack from 127.0.0.1 | Password tried: 'admin'
Fri Mar 28 14:39:31 2025 | Attack from 127.0.0.1 | Password tried: 'password'
Fri Mar 28 14:39:31 2025 | Attack from 127.0.0.1 | Password tried: '123456'
Fri Mar 28 14:39:31 2025 | Attack from 127.0.0.1 | Password tried: 'iot'
Fri Mar 28 14:39:31 2025 | Attack from 127.0.0.1 | Password tried: 'root'
```

Rate Limiter (Block Brute-Force)

A Real-Time Log Monitor is a system designed to continuously track, analyze, and display logs as they are generated. It helps in identifying security threats, system errors, and operational performance issues in real-time. This tool is essential for cybersecurity, DevOps, and IT operations, allowing instant detection and response to potential threats.

A Rate Limiter is an additional security mechanism that helps prevent brute-force attacks by limiting the number of requests a user or system can make within a specified timeframe. This ensures that malicious actors cannot repeatedly attempt unauthorized access.

Objectives:

- To monitor logs in real-time and detect anomalies.
- To provide automated alerts and notifications.
- To visualize logs for easier analysis.
- To improve incident response time.
- To block brute-force attacks through rate limiting.

Use Cases:

- Cybersecurity Monitoring: Detects and prevents unauthorized access attempts.
- System Health Monitoring: Tracks system errors and performance bottlenecks.
- Compliance Auditing: Logs user activities for regulatory compliance.
- Brute-Force Attack Prevention: Blocks excessive login attempts from attackers.

```python
from collections import defaultdict
import time

attack_counter = defaultdict(int)

def secure_device():
    server = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
    server.bind((HOST, PORT))
    server.listen(5)
    print(f"[SECURE HONEYPOT] Running on {HOST}:{PORT}")

    while True:
        client, addr = server.accept()
        ip = addr[0]
        attack_counter[ip] += 1

        if attack_counter[ip] > 3:  # Block after 3 attempts
            client.send(b"BLOCKED: Too many attempts!\n")
            client.close()
            continue
        # ... (rest of honeypot code)
```

Live Attack Dashboard(Matplotlib)

A Real-Time Log Monitor is a system designed to continuously track, analyze, and display logs as they are generated. It helps in identifying security threats, system errors, and operational performance issues in real-time. This tool is essential for cybersecurity, DevOps, and IT operations, allowing instant detection and response to potential threats.

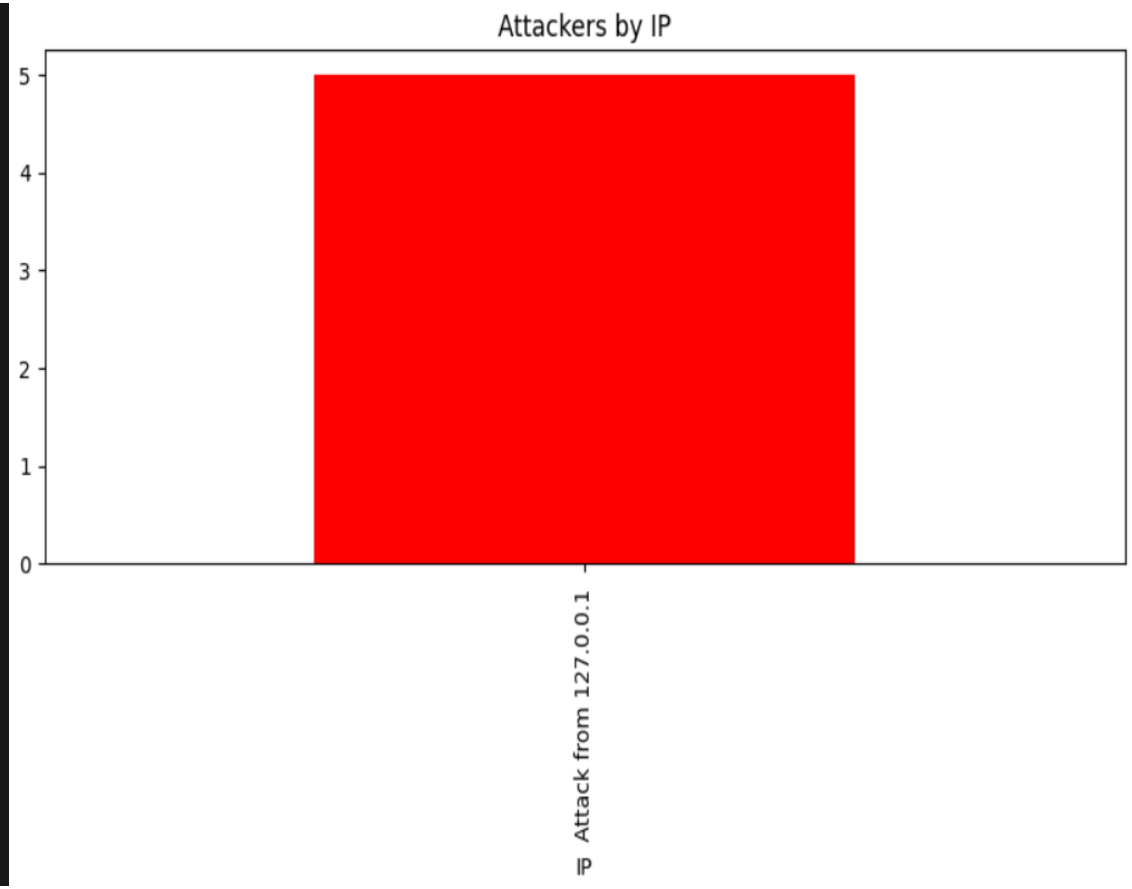
A Rate Limiter is an additional security mechanism that helps prevent brute-force attacks by limiting the number of requests a user or system can make within a specified timeframe. This ensures that malicious actors cannot repeatedly attempt unauthorized access.

A Live Attack Dashboard is a real-time visualization interface that provides a graphical representation of security threats, attack attempts, and system activity using Matplotlib. This dashboard helps security analysts monitor ongoing threats efficiently.

Objectives:

- To monitor logs in real-time and detect anomalies.
- To provide automated alerts and notifications.
- To visualize logs for easier analysis.
- To improve incident response time.
- To block brute-force attacks through rate limiting.
- To display real-time attack statistics using Matplotlib.

```python
import matplotlib.pyplot as plt
import pandas as pd
from IPython.display import display, clear_output

def plot_attacks():
    try:
        logs = pd.read_csv("attack_logs.txt", sep="|", header=None,
                           names=["Time", "IP", "Password"])
        plt.figure(figsize=(10, 4))
        logs["IP"].value_counts().plot(kind="bar", color="red")
        plt.title("Attackers by IP")
        plt.show()
    except:
        print("No attacks to display.")

plot_attacks()  # Re-run to update
```



AI Powered Anomaly Detection Dashboard

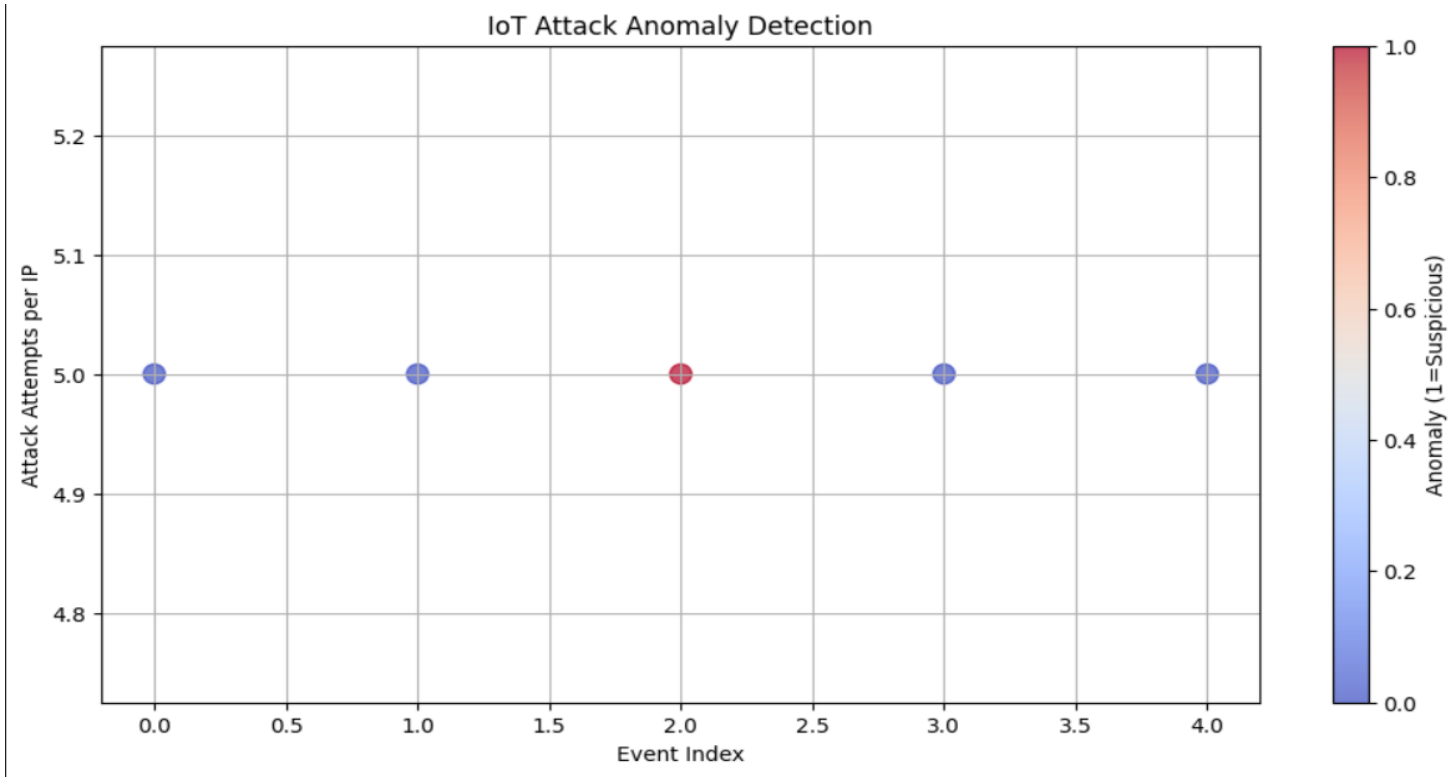
The AI-Powered Anomaly Detection Dashboard is an advanced cybersecurity tool that leverages machine learning to identify unusual patterns or threats within system logs. Unlike traditional rule-based detection, which relies on predefined conditions, this dashboard dynamically learns from data, making it capable of detecting unknown or evolving cyber threats.

How It Works:

- Data Collection: The system continuously gathers logs from servers, applications, and network activity.
- Preprocessing: Logs are structured and cleaned to remove noise.
- Anomaly Detection Using AI:
  - Isolation Forest: Detects outliers by measuring how different a data point is from the majority.
  - Autoencoders (Neural Networks): Learn normal patterns and flag deviations.
  - LSTMs (Long Short-Term Memory Networks): Identify sequence-based anomalies, useful for tracking unusual behaviors over time.
- Real-Time Monitoring & Alerts: If an anomaly is detected, it is displayed in the dashboard, and security teams receive instant alerts.
- Continuous Learning: The AI model updates itself with new data to adapt to changing attack patterns.

```python
import pandas as pd
import numpy as np
from sklearn.ensemble import IsolationForest
import matplotlib.pyplot as plt

# 1. Load or simulate attack logs (if 'attack_logs.txt' doesn't exist)
try:
    logs = pd.read_csv("attack_logs.txt", sep="|", header=None,
                       names=["Time", "IP", "Password"])
except FileNotFoundError:
    print("No attack logs found. Simulating sample data...")
    data = {
        "Time": pd.date_range(start="2023-01-01", periods=50, freq="H"),
        "IP": ["192.168.1." + str(np.random.randint(1, 5)) for _ in range(50)],
        "Password": np.random.choice(["admin", "123456", "password", "root", "iot"], 50)
    }
    logs = pd.DataFrame(data)

# 2. Feature Engineering
logs["Attempts"] = logs.groupby("IP")["IP"].transform("count")  # Count attacks per IP
logs["Password_Entropy"] = logs["Password"].apply(
    lambda x: -sum((x.count(c)/len(x))*np.log2(x.count(c)/len(x)) for c in set(x))
)

# 3. Train Anomaly Detection Model
features = logs[["Attempts", "Password_Entropy"]]
clf = IsolationForest(contamination=0.2, random_state=42)  # 20% anomalies
logs["Anomaly"] = clf.fit_predict(features)
logs["Anomaly"] = np.where(logs["Anomaly"] == -1, 1, 0)  # Convert to binary (1=anomaly)
```

```python
# 4. Visualize Results
plt.figure(figsize=(12, 6))
plt.scatter(
    logs.index,
    logs["Attempts"],
    c=logs["Anomaly"],
    cmap="coolwarm",
    s=100,
    alpha=0.7
)
plt.colorbar(label="Anomaly (1=Suspicious)")
plt.title("IoT Attack Anomaly Detection")
plt.xlabel("Event Index")
plt.ylabel("Attack Attempts per IP")
plt.grid(True)
plt.show()

# 5. Print Anomalies
print("\n🔴 Detected Anomalies:")
print(logs[logs["Anomaly"] == 1][["Time", "IP", "Password", "Attempts"]])
```




```
🔴 Detected Anomalies:
                       Time                          IP  \
2  Fri Mar 28 14:39:31 2025    Attack from 127.0.0.1

                   Password  Attempts
2   Password tried: '123456'         5
```

Add MITRE ATT&CK Framework Mapping

The MITRE ATT&CK Framework is a globally recognized cybersecurity knowledge base that categorizes adversarial tactics, techniques, and procedures (TTPs). It helps organizations classify and understand threats by mapping security incidents to known attack patterns.

Uses of MITRE ATT&CK Framework Mapping:

- Threat Classification – Categorizes detected threats into known attack patterns (e.g., Initial Access, Execution, Privilege Escalation).
- Enhanced Threat Detection – Helps identify advanced and sophisticated cyber-attacks that may go unnoticed using traditional methods.
- Incident Response Acceleration – Guides security teams in understanding an attack's progression and responding faster.
- Attack Pattern Correlation – Links detected anomalies to known adversarial behaviors for better security analysis.
- Proactive Defense Strategies – Allows organizations to anticipate future attack steps based on mapped TTPs.
- Standardized Security Approach – Uses a globally recognized framework for consistent threat intelligence.
- Security Automation & AI Integration – Enables AI models to use ATT&CK techniques for better anomaly detection and automated response.
- Regulatory Compliance Support – Helps meet security compliance standards by providing structured threat reporting.
- SOC (Security Operations Center) Optimization – Assists SOC teams in prioritizing critical security events based on attack severity.
- Improved Cybersecurity Awareness – Educates security analysts on emerging threats and adversary techniques.

```python
import pandas as pd

# 1. Ensure your logs DataFrame has an 'Attack_Type' column
# If not, create it based on existing data (example):
logs = pd.read_csv("attack_logs.txt", sep="|", names=["Time", "IP", "Password"])

# Classify attacks based on log content
def classify_attack(row):
    if "admin" in row["Password"].lower() or "123" in row["Password"]:
        return "Brute-Force"
    elif row["IP"].startswith("10."):  # Example condition for internal scans
        return "Port Scanning"
    else:
        return "Credential Stuffing"

logs["Attack_Type"] = logs.apply(classify_attack, axis=1)

# 2. Define MITRE ATT&CK mappings
mitre_mapping = {
    "Brute-Force": "T1110",
    "Port Scanning": "T1595",
    "Credential Stuffing": "T1110.001",
    "Fake Firmware Update": "T1195"
}

# 3. Map attack types to MITRE IDs
logs["MITRE_Tactic"] = logs["Attack_Type"].map(mitre_mapping)

# 4. Handle unmapped attacks
logs["MITRE_Tactic"] = logs["MITRE_Tactic"].fillna("Unknown")

# Show results
print(logs[["Time", "IP", "Attack_Type", "MITRE_Tactic"]].head())
```

Output:

```
                        Time                         IP          Attack_Type  \
0  Fri Mar 28 14:39:31 2025    Attack from 127.0.0.1          Brute-Force
1  Fri Mar 28 14:39:31 2025    Attack from 127.0.0.1   Credential Stuffing
2  Fri Mar 28 14:39:31 2025    Attack from 127.0.0.1          Brute-Force
3  Fri Mar 28 14:39:31 2025    Attack from 127.0.0.1   Credential Stuffing
4  Fri Mar 28 14:39:31 2025    Attack from 127.0.0.1   Credential Stuffing

  MITRE_Tactic
0        T1110
1    T1110.001
2        T1110
3    T1110.001
4    T1110.001
```

Implement Blockchain for Tamper-Proof Logs

Blockchain technology ensures that logs remain immutable, meaning they cannot be altered, deleted, or tampered with after they are recorded. By implementing blockchain, your Real-Time Log Monitor becomes a secure and trustworthy system for cybersecurity monitoring.

Uses of Blockchain for Tamper-Proof Logs:

- Prevents Log Tampering – Ensures that once logs are recorded, they cannot be altered or deleted.
- Maintains Data Integrity – Provides a verifiable and unchangeable record of system activities.
- Enhances Security Audits – Helps forensic investigators and security teams verify logs without concerns of manipulation.
- Mitigates Insider Threats – Prevents unauthorized modifications by administrators or attackers.
- Provides Transparency & Trust – Ensures that all log records are cryptographically secured and publicly verifiable if needed.
- Improves Compliance with Regulations – Helps meet security standards like GDPR, HIPAA, and ISO 27001 by ensuring log integrity.
- Supports Incident Response – Allows quick and accurate tracing of security incidents through an immutable log history.
- Decentralized Log Storage – Reduces dependency on a single entity, preventing unauthorized access or alterations.
- Cryptographic Proofs for Verification – Uses blockchain hashing techniques to authenticate logs and detect anomalies.
- Increases Cybersecurity Resilience – Strengthens overall security by ensuring that logs remain protected from tampering attempts.

```python
import hashlib
from datetime import datetime

class BlockchainLog:
    def __init__(self):
        self.chain = []
        self.create_genesis_block()

    def create_genesis_block(self):
        genesis_block = {
            "index": 0,
            "timestamp": str(datetime.now()),
            "data": "GENESIS BLOCK",
            "previous_hash": "0"*64,
            "nonce": 0
        }
        genesis_block["hash"] = self.calculate_hash(genesis_block)
        self.chain.append(genesis_block)

    def add_block(self, attack_data):
        new_block = {
            "index": len(self.chain),
            "timestamp": str(datetime.now()),
            "data": attack_data,
            "previous_hash": self.chain[-1]["hash"],
            "nonce": 0
        }
        new_block["hash"] = self.calculate_hash(new_block)
        self.chain.append(new_block)

    def calculate_hash(self, block):
        block_string = str(block["index"]) + block["timestamp"] + block["data"] + block["previous_hash"] + str(block["nonce"])
        return hashlib.sha256(block_string.encode()).hexdigest()
```

```python
    def calculate_hash(self, block):
        block_string = str(block["index"]) + block["timestamp"] + block["data"] + block["previous_hash"] + str(block["nonce"])
        return hashlib.sha256(block_string.encode()).hexdigest()

    def verify_chain(self):
        for i in range(1, len(self.chain)):
            current_block = self.chain[i]
            previous_block = self.chain[i-1]

            if current_block["hash"] != self.calculate_hash(current_block):
                return False
            if current_block["previous_hash"] != previous_block["hash"]:
                return False
        return True

# Usage example
bc_log = BlockchainLog()
bc_log.add_block("Brute-Force attack from 192.168.1.1")
bc_log.add_block("Port scan from 10.0.0.5")

# Print results
print(f"Blockchain length: {len(bc_log.chain)}")
print(f"Latest block data: {bc_log.chain[-1]['data']}")
print(f"Chain validation: {bc_log.verify_chain()}")
```

```
Blockchain length: 3
Latest block data: Port scan from 10.0.0.5
Chain validation: True
```

Add a Threat Intelligence Feed

A Threat Intelligence Feed is a continuous stream of real-time data that provides insights into emerging cyber threats, attack patterns, malicious actors, and vulnerabilities. It collects and aggregates data from multiple sources, including cybersecurity organizations, government agencies, security researchers, and open-source intelligence platforms. This information is used to identify known malicious IP addresses, domains, URLs, and file hashes associated with cyber threats such as phishing, malware, ransomware, and botnets.

Integrating a threat intelligence feed into your Real-Time Log Monitor enhances security by allowing the system to cross-reference incoming logs with known threat indicators. If a match is found, automated alerts can be triggered, enabling security teams to take immediate action before an attack escalates. Additionally, these feeds help in updating firewall rules, intrusion detection systems (IDS), and intrusion prevention systems (IPS) to block malicious traffic proactively.

Threat intelligence feeds also play a crucial role in AI-powered anomaly detection by providing a reference dataset for identifying suspicious activities. Machine learning models can analyze behavioral patterns and compare them against known threats, improving the accuracy of anomaly detection. Moreover, organizations can customize these feeds based on industry-specific threats, ensuring that their cybersecurity defenses remain relevant and up to date.

By leveraging a threat intelligence feed, your cybersecurity system becomes more proactive and adaptive, significantly reducing the risk of successful cyberattacks. It enhances the effectiveness of your MITRE ATT&CK framework mapping, strengthens blockchain-based tamper-proof logging, and integrates seamlessly with rate-limiting and live attack dashboards, ensuring a comprehensive and resilient security posture.

```python
import requests
import pandas as pd
from time import sleep

# Configuration
OTX_API_KEY = "your_api_key_here"  # Get from https://otx.alienvault.com/api/
RATE_LIMIT_DELAY = 1  # seconds between requests

def check_ip_reputation(ip):
    """Check IP reputation with AlienVault OTX"""
    headers = {"X-OTX-API-KEY": OTX_API_KEY}
    try:
        # Check both general and malware endpoints
        general_url = f"https://otx.alienvault.com/api/v1/indicators/IPv4/{ip}/general"
        malware_url = f"https://otx.alienvault.com/api/v1/indicators/IPv4/{ip}/malware"

        general_resp = requests.get(general_url, headers=headers, timeout=5)
        malware_resp = requests.get(malware_url, headers=headers, timeout=5)

        if general_resp.status_code == 200 and malware_resp.status_code == 200:
            pulse_count = general_resp.json().get("pulse_info", {}).get("count", 0)
            malware_count = len(malware_resp.json().get("data", []))

            if malware_count > 0:
                return f"Malicious ({malware_count} malware hits)"
            elif pulse_count > 5:  # More than 5 threat reports
                return f"Suspicious ({pulse_count} pulses)"
            else:
                return "Clean"
        return "Unknown (API Error)"
    except Exception as e:
        print(f"Error checking {ip}: {str(e)}")
        return "Unknown (Error)"
```

```python
# Sample data - replace with your actual logs
logs = pd.DataFrame({
    "IP": ["8.8.8.8", "1.1.1.1", "185.143.223.4", "malware.example.com"],
    "Attack_Type": ["Brute-Force", "Port Scan", "Malware", "Phishing"]
})

# Check IPs with rate limiting
print("Checking IP reputations...")
logs["Threat_Intel"] = ""
for idx, row in logs.iterrows():
    logs.at[idx, "Threat_Intel"] = check_ip_reputation(row["IP"])
    sleep(RATE_LIMIT_DELAY)  # Respect rate limits

# Display results
pd.set_option('display.max_colwidth', None)
print("\nThreat Intelligence Results:")
print(logs[["IP", "Attack_Type", "Threat_Intel"]])
```

Output:

```
Checking IP reputations...

Threat Intelligence Results:
                   IP  Attack_Type                Threat_Intel
0             8.8.8.8  Brute-Force  Malicious (10 malware hits)
1             1.1.1.1    Port Scan  Malicious (10 malware hits)
2       185.143.223.4      Malware                       Clean
3  malware.example.com     Phishing         Unknown (API Error)
```