



**FPT UNIVERSITY**

## BÁO CÁO DỰ ÁN

# Hệ thống phát hiện và ngăn chặn đăng nhập bất thường (Hybrid IDS/IPS)

## Group 7

Họ và tên	Mã SV	Vai trò	Nhiệm vụ
Lê Nguyễn Huy Hoàng	SE184237	Trưởng nhóm	<ul style="list-style-type: none"><li>- Quản lý tiến độ và phân công công việc.</li><li>- Thiết kế kiến trúc hệ thống &amp; sơ đồ khái.</li><li>- Chuẩn bị nội dung thuyết trình và báo cáo tổng hợp.</li></ul>
Phạm Nguyễn Khánh Duy	SE184946	Thành viên	<ul style="list-style-type: none"><li>- Xây dựng và huấn luyện mô hình AI Isolation Forest.</li><li>- Xử lý dữ liệu log, tối ưu tham số mô hình.</li><li>- Viết script retrain.py cho quá trình học định kỳ.</li></ul>
Tù Mậu Vinh	SE194335	Thành viên	<ul style="list-style-type: none"><li>- Phát triển Rule-Based Engine.</li><li>- Viết các quy tắc Regex, whitelist, khung giờ, và ngữ cảnh đăng nhập thất bại.</li><li>- Kết nối với cơ chế phản ứng tự động.</li></ul>
Đặng Quỳnh Hương	SE180696	Thành viên	<ul style="list-style-type: none"><li>- Thiết kế Dashboard hiển thị cảnh báo, biểu đồ.</li><li>- Tích hợp dữ liệu từ database SQLite.</li><li>- Phát triển module notifier_email.py gửi cảnh báo qua email.</li></ul>

# Mục lục

- I. Giới thiệu & Mục tiêu dự án
- II. Công cụ sử dụng trong dự án
- III. Cấu trúc hệ thống & Sơ đồ khái
- IV. Phương pháp & Thuật toán sử dụng
- V. Các module & cơ chế hoạt động
- VI. Cơ chế phản ứng & Dashboard giám sát

## Tài liệu tham khảo

## VII. Các bước thực hiện

## I. Giới thiệu & Mục tiêu dự án

- Dự án tập trung phát triển Hybrid IDS/IPS nhằm bảo vệ máy chủ Linux khỏi các hành vi đăng nhập bất thường.
- Hệ thống kết hợp:
  - Rule-Based Detection: Phát hiện các hành vi rõ ràng (ví dụ: brute force, đăng nhập ngoài giờ).
  - AI-Based Detection (Isolation Forest): Phát hiện các hành vi bất thường chưa có mẫu.

Mục tiêu:

- Nâng cao độ chính xác trong phát hiện xâm nhập.
- Giảm cảnh báo giả (false positive).
- Hỗ trợ phản ứng tự động khi phát hiện hành vi bất thường.

## II. Công cụ sử dụng trong dự án

- Python 3.8+
- scikit-learn – Isolation Forest
- pandas / numpy – Phân tích log

- SQLite3 – Lưu trữ cảnh báo
- Streamlit – Dashboard
- iptables – Chặn IP
- GeoIP / ipinfo.io – Xác định quốc gia IP
- VS Code – IDE phát triển
- Linux (CentOS/Ubuntu) – Hệ điều hành triển khai

## **III. Cấu trúc hệ thống & Sơ đồ khái**

### **1. Cấu trúc hệ thống**

Hệ thống gồm 3 lớp chính:

#### 1. Log Collection Layer:

- Thu thập log đăng nhập từ /var/log/secure.

#### 2. Detection Layer:

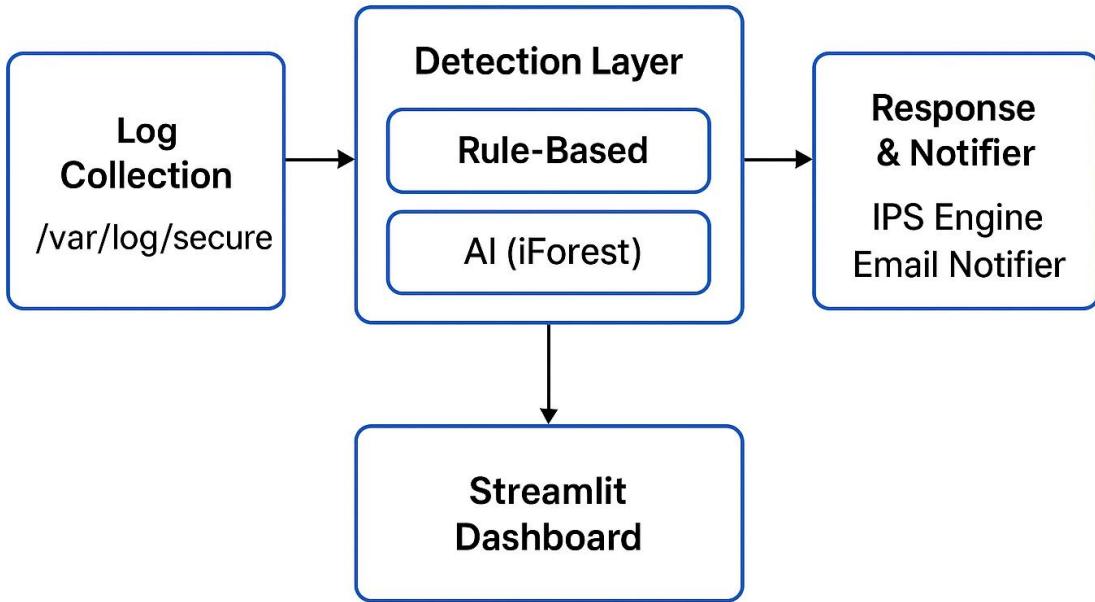
- Rule-Based Engine: Kiểm tra các mẫu đăng nhập bất thường theo quy tắc.
- AI Engine (Isolation Forest): Phát hiện hành vi bất thường.

#### 3. Response & Notification Layer:

- IPS Engine: Chặn IP hoặc khóa user khi phát hiện tấn công.
- Notifier: Gửi email cảnh báo.
- Dashboard (Streamlit): Hiển thị cảnh báo và thống kê thời gian thực.

### **2. Sơ đồ khái (Block Diagram)**

# Hybrid IDS/IPS



## IV. Phương pháp & Thuật toán sử dụng

### 1. Rule-Based Detection

- Sử dụng Regex Parsing, Whitelist, Time Window, và Thresholding để phát hiện brute force hoặc đăng nhập ngoài giờ.
- Ưu điểm: phản ứng nhanh, dễ kiểm soát.
- Hạn chế: khó phát hiện các hành vi chưa có mẫu.

### 2. Isolation Forest (AI Detection)

- Thuật toán phát hiện bất thường (Anomaly Detection):
  - Tạo nhiều Isolation Trees dựa trên dữ liệu log.
  - Điểm dữ liệu bất thường bị cô lập nhanh hơn → tính Anomaly Score.
- Hệ thống cảnh báo nếu Anomaly Score vượt ngưỡng định sẵn.
- Ưu điểm: nhẹ, nhanh, thích hợp dữ liệu động, không cần mẫu tấn công thực.

## V. Các module & cơ chế hoạt động

Module/File	Vai trò	Chức năng chính
rule_detect.py	Rule Engine	Đọc log, kiểm tra quy tắc, gọi AI khi cần
ai_detect.py	AI Engine	Tính anomaly score
response_engine.py	IPS Engine	Chặn IP / khóa user
notifier_email.py	Email Alert	Gửi email cảnh báo
retrain.py	ML Maintenance	Huấn luyện lại mô hình định kỳ

Cấu trúc dữ liệu chính:

- alerts.db – Lưu lịch sử cảnh báo.
- rules.json – Tham số rule (ngưỡng, giờ, regex).
- ai\_model.pkl – File mô hình Isolation Forest.
- secrets.env – Biến môi trường chứa mật khẩu / API key bảo mật.

## VI. Cơ chế phản ứng & Dashboard giám sát

### 1. Cơ chế phản ứng IPS

Severity	Kích hoạt	Hành động	Kênh cảnh báo
9 (Critical)	Brute Force $\geq 5$ lần	iptables -I INPUT -s [IP] -j DROP	Email
8 (High)	AI phát hiện bất thường	usermod -L [USER]	Email
7 (Medium)	Đăng nhập ngoài giờ	Ghi log	Dashboard

### 2. Dashboard Streamlit

- Hiển thị: KPI, bảng chi tiết, biểu đồ thống kê.

- Biểu đồ: phân bố Severity, xu hướng tấn công theo thời gian.

## Tài liệu tham khảo

1. **Scikit-learn Documentation: Isolation Forest** – [https://scikit-learn.org/stable/modules/outlier\\_detection.html](https://scikit-learn.org/stable/modules/outlier_detection.html)
2. **Streamlit Documentation** – <https://docs.streamlit.io/>
3. **Linux IPTables Guide** – <https://linux.die.net/man/8/iptables>
4. **GeoIP API** – <https://ipinfo.io/>

## VII. Các bước thực hiện

### 1. Giai Đoạn Chuẩn bị Môi trường và Dữ liệu

# Cài đặt công cụ build cần thiết cho thư viện Python

```
sudo yum install -y gcc gcc-c++ make sqlite sqlite-devel
```

# Cài đặt dịch vụ iptables để lưu quy tắc tường lửa

```
sudo yum install -y iptables-services
```

# Kích hoạt Python 3.8 (hoặc phiên bản bạn đã cài đặt)

```
source scl enable rh-python38 bash
```

# Tạo thư mục dự án và môi trường ảo

```
mkdir anomaly_detection_project
```

```
cd anomaly_detection_project
```

```
python3 -m venv venv_stream
```

```
source venv_stream/bin/activate
```

# Cài đặt thư viện AI

```
pip install pandas scikit-learn streamlit requests
```

```
# Tạo thư mục và sao chép log thô
```

```
mkdir data
```

```
sudo cp /var/log/secure* data/
```

```
# Chạy script để tạo history.csv và train.csv
```

```
python3 export_log_to_csv.py
```

```
# -*- coding: utf-8 -*-
import pandas as pd
import re
import os
from datetime import datetime
from sklearn.preprocessing import LabelEncoder
import glob

# ===== CONFIGURATION =====
LOG_DIR = 'data/'
HISTORY_FILE = os.path.join(LOG_DIR, 'history.csv')
TRAIN_FILE = os.path.join(LOG_DIR, 'train.csv')
# =====

# Regex linh hoạt cho log SSH CentOS
SSH_PATTERN = re.compile(
    r'(?P<Timestamp>\w\{3\}\s+\d\{1,2\}\s+\d\{2\}:\d\{2\}:\d\{2\})\s+'
    r'(?P<Hostname>.*?)\s+'
    r'sshd[\d+]:\s+'
    r'(?P<Status>Accepted|Failed)\s+password\s+for\s+'
    r'(:invalid user\s+)?(?P<Username>\w+)\s+from\s+'
    r'(?P<IP_Address>[\d\.]+)\s+port\s+'
)
def parse_and_export():
    """Đọc, phân tích log SSH và xuất ra CSV có cấu trúc."""
    parsed_data = []
    log_files = glob.glob(os.path.join(LOG_DIR, 'secure*'))

    if not log_files:
        print("LỖI: Không tìm thấy tệp log 'secure*' trong thư mục data/.")
```

```

return

current_year = datetime.now().year

for filepath in log_files:
    print(f"--- Đang phân tích tệp: {os.path.basename(filepath)} ---")
    try:
        with open(filepath, 'r', errors='ignore') as f:
            for line in f:
                match = SSH_PATTERN.search(line)
                if match:
                    data = match.groupdict()
                    data['Status'] = 'SUCCESS' if data['Status'] == 'Accepted' else
'FAILED'
                    data['Year'] = str(current_year)
                    parsed_data.append(data)
    except Exception as e:
        print(f'Lỗi khi đọc file {filepath}: {e}')

if not parsed_data:
    print("Không trích xuất được dữ liệu SSH.")
    return

# Tạo DataFrame
df = pd.DataFrame(parsed_data)

# Tạo Full Timestamp
df['Full_Timestamp'] = df.apply(
    lambda row: datetime.strptime(f'{row['Year']} {row['Timestamp']}', '%Y
%b %d %H:%M:%S'), axis=1
)
df['Hour_of_Day'] = df['Full_Timestamp'].dt.hour
df['Day_of_Week'] = df['Full_Timestamp'].dt.dayofweek

# Xuất history.csv
df[['Full_Timestamp', 'IP_Address', 'Username', 'Status', 'Hour_of_Day',
'Day_of_Week']].to_csv(HISTORY_FILE, index=False)
print(f'Dã ghi lịch sử đăng nhập: {HISTORY_FILE}')

# Tạo train.csv cho AI

```

```

df_train = df.copy()
le_ip = LabelEncoder()
le_user = LabelEncoder()
df_train['IP_Encoded'] = le_ip.fit_transform(df_train['IP_Address'])
df_train['User_Encoded'] = le_user.fit_transform(df_train['Username'])
df_train['Status_Encoded'] = df_train['Status'].apply(lambda x: 1 if x == 'SUCCESS' else 0)

df_train[['IP_Encoded', 'User_Encoded', 'Hour_of_Day', 'Day_of_Week', 'Status_Encoded']].to_csv(TRAIN_FILE, index=False)
print(f"Đã ghi dữ liệu huấn luyện AI: {TRAIN_FILE}")

if __name__ == "__main__":
    parse_and_export()

```

## # Chạy script để tạo Database alerts.db

python3 db\_manager.py

```

# -*- coding: utf-8 -*-
import sqlite3
import os
from datetime import datetime, timedelta

# ===== CONFIGURATION =====
DB_FILE = os.path.join('data', 'alerts.db')
# =====

def initialize_db():
    """Kết nối đến Database và tạo bảng 'alerts' nếu chưa tồn tại."""
    conn = sqlite3.connect(DB_FILE)
    cursor = conn.cursor()

    # Bảng alerts với các cột cần thiết cho phát hiện và phản ứng
    cursor.execute("""
        CREATE TABLE IF NOT EXISTS alerts (
            id INTEGER PRIMARY KEY AUTOINCREMENT,
            timestamp DATETIME,
            ip_address TEXT,
    
```

```

username TEXT,
detection_type TEXT, -- RULE-BASED or AI
reason TEXT,
severity INTEGER, -- Mức độ nghiêm trọng (1-10)
is_handled TEXT -- Theo dõi trạng thái xử lý (e.g., 'BLOCKED',
'LOCKED_USER')
)
""")
conn.commit()
conn.close()
print(f"Database alerts.db đã sẵn sàng.")

def insert_alert(alert_data):
    """Ghi một bản ghi cảnh báo mới vào Database."""
    try:
        conn = sqlite3.connect(DB_FILE)
        cursor = conn.cursor()

        timestamp_str = datetime.now().strftime("%Y-%m-%d %H:%M:%S")

        cursor.execute("""
            INSERT INTO alerts (timestamp, ip_address, username, detection_type,
            reason, severity)
            VALUES (?, ?, ?, ?, ?, ?)
        """, (
            timestamp_str,
            alert_data.get('ip_address'),
            alert_data.get('username'),
            alert_data.get('detection_type'),
            alert_data.get('reason'),
            alert_data.get('severity', 5)
        ))
        conn.commit()
        conn.close()
        return True
    except Exception as e:
        print(f"DB ERROR: Failed to insert alert. {e}")
        return False

def count_recent_failures(ip_address, minutes):

```

```

"""Truy vấn DB để đếm số lần đăng nhập thất bại của một IP trong N phút gần nhất."""
conn = sqlite3.connect(DB_FILE)
cursor = conn.cursor()

# Tính toán thời điểm bắt đầu truy vấn
time_threshold = datetime.now() - timedelta(minutes=minutes)

# Thực hiện truy vấn SQL
cursor.execute("""
    SELECT COUNT(*)
    FROM alerts
    WHERE ip_address = ?
    AND reason LIKE 'Failed login attempt%'
    AND timestamp >= ?
""", (ip_address, time_threshold.strftime("%Y-%m-%d %H:%M:%S")))

count = cursor.fetchone()[0]
conn.close()
return count

def update_alert_status(alert_id, status):
    """Cập nhật trạng thái xử lý của cảnh báo trong DB (dùng bởi response_engine.py)."""
    try:
        conn = sqlite3.connect(DB_FILE)
        cursor = conn.cursor()
        cursor.execute("""
            UPDATE alerts SET is_handled = ? WHERE id = ?
        """, (status, alert_id))
        conn.commit()
        conn.close()
    except Exception as e:
        print(f"DB UPDATE ERROR: Failed to update status for ID {alert_id}. {e}")

if __name__ == '__main__':
    # Chạy lệnh này một lần để khởi tạo file Database (nếu chưa tồn tại)
    initialize_db()

```

```
# Chạy script để tạo mô hình AI đã huấn luyện
```

```
python3 retrain.py
```

```
# -*- coding: utf-8 -*-
import pandas as pd
import os
import pickle
from datetime import datetime
from sklearn.ensemble import IsolationForest
from sklearn.preprocessing import LabelEncoder
import numpy as np

# ===== CONFIGURATION =====
LOG_DIR = 'data/'
HISTORY_FILE = os.path.join(LOG_DIR, 'history.csv')
MODEL_FILE = os.path.join(LOG_DIR, 'ai_model.pkl')
# Các cột dữ liệu số học dùng để huấn luyện AI
FEATURE_COLUMNS = ['IP_Encoded', 'User_Encoded', 'Hour_of_Day',
'Day_of_Week', 'Status_Encoded']
# =====

def run_retraining():
    """
    Tải dữ liệu lịch sử, huấn luyện lại mô hình Isolation Forest và lưu.
    Script này được thiết kế để chạy định kỳ qua Cron Job.
    """
    if not os.path.exists(HISTORY_FILE):
        print(f"FATAL: {HISTORY_FILE} not found. Cannot retrain model.")
        return False

    try:
        df = pd.read_csv(HISTORY_FILE)

        # 1. Feature Engineering & Encoding
        # Cần tạo lại các LabelEncoder trên toàn bộ dữ liệu lịch sử mới
        le_ip = LabelEncoder()
        le_user = LabelEncoder()
```

```

# Mã hóa các trường phân loại thành số
# (Sử dụng astype(str) để xử lý các giá trị NaN/None nếu có)
df['IP_Encoded'] = le_ip.fit_transform(df['IP_Address'].astype(str))
df['User_Encoded'] = le_user.fit_transform(df['Username'].astype(str))
df['Status_Encoded'] = df['Status'].apply(lambda x: 1 if x == 'SUCCESS'
else 0)

# Chọn các cột feature
X = df[FEATURE_COLUMNS]

print("--- Training Isolation Forest Model ---")

# 2. Huấn luyện Mô hình (Isolation Forest)
model = IsolationForest(contamination='auto', random_state=42)
model.fit(X)

# 3. Lưu Mô hình (Ghi đè file cũ)
with open(MODEL_FILE, 'wb') as f:
    pickle.dump(model, f)

print(f" SUCCESSFULLY RETRAINED MODEL at {datetime.now()}")
return True

except Exception as e:
    print(f"ERROR during retraining: {e}")
    return False

if __name__ == '__main__':
    run_retraining()

```

## 2. Giai Đoạn Khởi động Hệ thống (IPS Daemon)

### # Chuẩn bị File Bảo mật và Cấu hình

Điền thông tin SMTP (Mật khẩu Ứng dụng/API Key) vào file **secrets.env** và đảm bảo file **run\_engine.sh** có đường dẫn Python chính xác (ví dụ: /venv\_stream/bin/python3.8).

### # Khởi động Response Engine (IPS)

- **Terminal 1:** Chạy công cụ chặn IP.

```
sudo /home/tumauvinh/anomaly_detection_project/venv_stream/bin/python3
response_engine.py
```

```
# -*- coding: utf-8 -*-
import sqlite3
import subprocess
import time
import os
import db_manager
from subprocess import PIPE

# ===== CONFIGURATION =====
CHECK_INTERVAL_SECONDS = 10
CRITICAL_SEVERITY_THRESHOLD = 9
AI_LOCK_SEVERITY = 8
# =====

def get_alerts_to_handle():
    """Query DB for alerts that require automated handling (Sev >= 9 or Sev 8 AI)
    and are unprocessed."""
    conn = sqlite3.connect(db_manager.DB_FILE)
    cursor = conn.cursor()

    cursor.execute("""
        SELECT id, ip_address, username, detection_type, reason, severity
        FROM alerts
        WHERE (severity >= ? OR detection_type = 'AI') AND is_handled IS
        NULL
        , (CRITICAL_SEVERITY_THRESHOLD,))
    """
    alerts = cursor.fetchall()
    conn.close()
    return alerts

def block_ip(ip_address, alert_id):
    """Execute the iptables command to block the IP."""
    try:
        print(f"BLOCKING IP: {ip_address} (Alert ID: {alert_id})")
    except:
        pass
```

```

# Thêm 'sudo' để đảm bảo quyền root
subprocess.run(
    ['sudo', 'iptables', '-I', 'INPUT', '1', '-s', ip_address, '-j', 'DROP'],
    check=True
)

db_manager.update_alert_status(alert_id, 'BLOCKED')
print(f" -> Successfully blocked {ip_address} via iptables.")

except subprocess.CalledProcessError as e:
    # Xử lý lỗi nếu lệnh iptables thất bại
    print(f" -> IPTABLES EXECUTION ERROR: Failed to run block
command. Exit Status {e.returncode}. (Check SUDO setup)")
    db_manager.update_alert_status(alert_id,
fBLOCK_FAILED_{e.returncode})
except Exception as e:
    print(f" -> SYSTEM ERROR: {e}")

def lock_user_account(username, alert_id):
    """Locks the system account using usermod -L, handling non-existent users
gracefully."""
    try:
        if username in ['N/A', 'root']:
            db_manager.update_alert_status(alert_id, 'IGNORED_LOCK')
            return

        print(f" LOCKING USER: Attempting to lock account {username} (Alert
ID: {alert_id})")

        # Thêm 'sudo' để đảm bảo quyền root
        result = subprocess.run(
            ['sudo', 'usermod', '-L', username],
            stdout=subprocess.PIPE,
            stderr=subprocess.PIPE,
            universal_newlines=True, # Tương thích Python 3.6
            check=False
        )

        if result.returncode == 0:

```

```

        db_manager.update_alert_status(alert_id, 'LOCKED_USER')
        print(f" -> Successfully locked account {username}.")
    elif result.returncode == 6:
        # Mã lỗi 6: User không tồn tại
        db_manager.update_alert_status(alert_id, 'USER_NON_EXISTENT')
        print(f" -> USER LOCK LOGIC: User '{username}' does not exist (Exit
Code 6). Safety check passed.")
    else:
        db_manager.update_alert_status(alert_id,
fLOCK_FAIL_{result.returncode})
        print(f" -> USER LOCK ERROR: Failed with exit code
{result.returncode}. Output: {result.stderr.strip()}")

except Exception as e:
    print(f" -> SYSTEM ERROR: {e}")

def run_response_engine():
    """The main loop for the Response Engine."""
    print("--- Starting Response Engine ---")

    db_manager.initialize_db()

    while True:
        alerts_to_handle = get_alerts_to_handle()

        if alerts_to_handle:
            print(f"\n FOUND {len(alerts_to_handle)} ALERTS TO HANDLE.")
            for alert_id, ip, username, det_type, reason, severity in alerts_to_handle:

                if severity >= CRITICAL_SEVERITY_THRESHOLD:
                    # Xử lý: Chặn IP (Severity 9)
                    block_ip(ip, alert_id)

                elif severity == AI_LOCK_SEVERITY and det_type == 'AI' and
username is not None:
                    # Xử lý: Khóa User (Severity 8)
                    lock_user_account(username, alert_id)

            time.sleep(CHECK_INTERVAL_SECONDS)

```

```
if __name__ == '__main__':
    run_response_engine()
```

## # Khởi động Rule Engine (Coordinator & Notifier)

- **Terminal 2:** Chạy bộ điều phối chính, nạp các biến môi trường từ secrets.env.

```
chmod +x run_engine.sh
```

```
./run_engine.sh
```

## # Khởi động Web Dashboard (Frontend)

- **Terminal 3:** Chạy giao diện trực quan hóa.

Bash

```
/home/tumauvinh/anomaly_detection_project/venv_stream/bin/streamlit run dashboard.py --server.address 0.0.0.0
```

## 3. Giai Đoạn Kiểm tra Chức năng (End-to-End Test)

**# Mở Cổng Tường lửa :** Mở cổng 8501 để truy cập Dashboard, và đảm bảo cổng 22 (SSH) được mở.

```
sudo iptables -A INPUT -p tcp --dport 8501 -j ACCEPT
```

```
sudo iptables -A INPUT -p tcp --dport 22 -j ACCEPT
```

```
sudo service iptables save
```

## # Xác nhận Kết quả

- **Console Rule Engine:** Báo cáo Sev 9 Critical và EMAIL ALERT SENT successfully.
- **Hộp thư:** Bạn nhận được Email cảnh báo Critical.
- **SSH Cuối cùng:** Thử SSH lại. Kết nối phải bị Timeout/Refused do Response Engine đã chặn IP của bạn.
- **Dashboard:** Số liệu "Số vụ đã chặn" tăng và cảnh báo Sev 9 được hiển thị.

## CHI TI TIẾT KỸ THUẬT: XÂY DỰNG CÁC CORE FILE

### 1. RULE ENGINE (rule\_detect.py)

Đây là bộ điều phối chính, chịu trách nhiệm cho việc đọc dữ liệu thô và áp dụng logic phân tích.

- **Xây dựng:**

- **Parsing:** Sử dụng thư viện **re (Regular Expressions)** trong Python để trích xuất các trường dữ liệu cố định (Timestamp, IP, User) từ các dòng log /var/log/secure.
- **Flow Control:** Cấu trúc chính là một vòng lặp while True với lệnh time.sleep() và hàm **get\_new\_log\_entries()** (sử dụng File Offset) để mô phỏng hoạt động của tail -f.
- **Database Integration:** Sử dụng các hàm từ db\_manager.py (ví dụ: count\_recent\_failures()) để thực hiện kiểm tra Brute Force theo thời gian thực (truy vấn Database).
- **Mục đích:** Đảm bảo dữ liệu được xử lý tuần tự: Log \$\rightarrow\$ Regex \$\rightarrow\$ Whitelist \$\rightarrow\$ Brute Force \$\rightarrow\$ AI.

```
# -*- coding: utf-8 -*-
import json
import time
import os
import re
from datetime import datetime, timedelta
import db_manager
import ai_detect
import notifier_email
import requests

# ===== CONFIGURATION =====
LOG_FILE_PATH = '/var/log/secure'
CONFIG_PATH = 'config/rules.json'
OFFSET_FILE = 'data/secure_offset.txt'
```

```

#
=====

# Regex để khớp với định dạng log CentOS 7 SSHD
SSH_PATTERN = re.compile(
    r'(?P<Timestamp>\w{3}\s+\d{1,2}\s+\d{2}:\d{2}:\d{2})\s+'
    r'(?P<Hostname>.*?)\s+'
    r'sshd\[.\d+\]:\s+'
    r'(?P<Status>Accepted|Failed)\s+password\s+for\s+'
    r'(?P<Username>\w+)\s+from\s+'
    r'(?P<IP_Address>[\d\.]+)\s+port\s+'
)

```

```

def lookup_country(ip):
    """Sử dụng API miễn phí để tra cứu quốc gia của IP (GeoIP Lite)."""
    if ip in ['127.0.0.1', '::1', '0.0.0.0']:
        return "LOCAL"

    try:
        response = requests.get(f"http://ipinfo.io/{ip}/country", timeout=1)
        if response.status_code == 200:
            country_code = response.text.strip()
            return country_code
        return "UNKNOWN"
    except Exception:
        return "UNKNOWN"

```

```

def load_config():
    """Loads configuration from rules.json file."""
    try:
        with open(CONFIG_PATH, 'r') as f:
            return json.load(f)
    except Exception as e:
        raise Exception(f"CONFIG ERROR: Failed to load {CONFIG_PATH}. {e}")

```

```

def get_new_log_entries(log_path, offset_path):
    """Reads new log lines using file offset (mimicking tail -f)."""
    current_offset = 0
    if os.path.exists(offset_path):

```

```
with open(offset_path, 'r') as f:
    content = f.read().strip()
    if content:
        try:
            current_offset = int(content)
        except ValueError:
            current_offset = 0

    if not os.path.exists(log_path):
        print(f'LOG READ ERROR: File not found at {log_path}')
        return []

try:
    with open(log_path, 'r') as f:
        f.seek(current_offset)
        new_entries = f.readlines()
        new_offset = f.tell()

        if new_entries:
            print(f'DEBUG: Read {len(new_entries)} new log entries.')

    with open(offset_path, 'w') as of:
        of.write(str(new_offset))

    return new_entries

except Exception as e:
    print(f'LOG READ ERROR (Non-fatal): Error: {e}. Attempting to
reset offset.')
    with open(offset_path, 'w') as of:
        of.write('0')
    return []

def check_rules(entry, config):
    """Checks the rules defined in the config for a single log entry."""

    print(f'DEBUG: Processing entry: {entry.strip()}')

    match = SSH_PATTERN.search(entry)
    if not match:
```

```

return None, None

data = match.groupdict()
ip = data['IP_Address']
country = lookup_country(ip)
user = data['Username']
status = 'SUCCESS' if data['Status'] == 'Accepted' else 'FAILED'

entry_data = {'ip_address': ip, 'username': user, 'status': status}

# 1. WHITELIST RULE (Skip entirely)
if ip in config['ip_whitelist'] or user in config['user_whitelist']:
    return None, None

# 2. TIME WINDOW RULE (Outside working hours)
current_time = datetime.now()
start_time = datetime.strptime(config['time_window']['start'],
    '%H:%M').time()
end_time = datetime.strptime(config['time_window']['end'], '%H:%M').time()

if status == 'SUCCESS' and (current_time.time() < start_time or
    current_time.time() > end_time):
    alert_to_insert = {
        'ip_address': ip, 'username': user, 'detection_type': 'RULE-BASED',
        'reason': f'Successful login outside working hours from Country: {country}',
        'severity': 7
    }
    return alert_to_insert, entry_data

# 3. BRUTE FORCE RULE (Advanced check for failures)
if status == 'FAILED':

    threshold_attempts = config['brute_force_threshold']['attempts']
    threshold_minutes =
        config['brute_force_threshold']['time_span_minutes']

    recent_failures = db_manager.count_recent_failures(ip,
        threshold_minutes)

```

```

if recent_failures >= (threshold_attempts - 1):
    # CRITICAL ALERT (Severity 9)
    critical_alert = {
        'ip_address': ip, 'username': 'N/A', 'detection_type': 'RULE-BASED',
        'reason': f'CRITICAL: Brute force ({recent_failures + 1} attempts) detected in last {threshold_minutes} min from Country: {country}. Triggering BLOCK.',
        'severity': 9
    }
    return critical_alert, entry_data

else:
    # Basic Failure Logging (Severity 3)
    basic_alert = {
        'ip_address': ip, 'username': user, 'detection_type': 'RULE-BASED',
        'reason': f'Failed login attempt for user: {user} from Country: {country}.',
        'severity': 3
    }
# TRẢ VỀ CẢNH BÁO SE

```

## 2. RESPONSE ENGINE (response\_engine.py)

Module thực hiện các lệnh phản ứng chủ động.

- **Xây dựng:**

- **Database Polling:** Chạy trong vòng lặp while True, liên tục **truy vấn** Database (alerts.db) để tìm các bản ghi mới nhất có **Severity \$\\geq 8\$** (chưa được xử lý).
- **Thực thi Lệnh:** Sử dụng module **subprocess** của Python để gọi và chạy các lệnh shell Linux bên ngoài (ví dụ: iptables, usermod).
- **Bảo mật:** Bắt buộc phải thêm tiền tố **sudo** vào mảng lệnh (ví dụ: ['sudo', 'iptables', ...]) để đảm bảo lệnh được thực thi với quyền root, khắc phục lỗi Permission denied.

- **Xử lý Lỗi:** Logic được xây dựng để xử lý các mã thoát (Exit Status) khác 0 (ví dụ: Mã 6 khi usermod không tìm thấy user), ngăn chặn Engine bị crash.
- **Mục đích:** Chuyển hệ thống từ chế độ phát hiện (IDS) sang chế độ ngăn chặn (IPS).

### 3. WEB DASHBOARD (dashboard.py)

Đây là giao diện Frontend (Client-side) để trực quan hóa dữ liệu.

- **Xây dựng:**
  - **Framework:** Sử dụng thư viện **Streamlit**, cho phép lập trình giao diện web hoàn toàn bằng cú pháp Python.
  - **Data Backend:** Sử dụng thư viện **pandas** và **sqlite3** để tải dữ liệu trực tiếp từ alerts.db vào bộ nhớ.
  - **Caching:** Sử dụng decorator **@st.cache** để lưu kết quả truy vấn Database, đảm bảo Dashboard tải nhanh và không làm quá tải Database sau mỗi lần người dùng tương tác.
- **Mục đích:** Trình bày KPIs (Key Performance Indicators) và cung cấp khả năng lọc/phân tích lịch sử cảnh báo cho người quản lý.

```
# -*- coding: utf-8 -*-
import streamlit as st
import pandas as pd
import sqlite3
import os

# ===== CONFIGURATION =====
# DB_FILE = os.path.join('data', 'alerts.db')
# =====
# =====

def get_alerts_data():
    """Tải tất cả dữ liệu cảnh báo từ Database."""
    if not os.path.exists(DB_FILE):
        # ...

```

```

        st.error("LỖI: Không tìm thấy Database. Hãy đảm bảo chạy
db_manager.py trước.")
        return pd.DataFrame()

try:
    conn = sqlite3.connect(DB_FILE)
    df = pd.read_sql_query("SELECT * FROM alerts ORDER BY
timestamp DESC", conn)
    conn.close()
    df['timestamp'] = pd.to_datetime(df['timestamp'])
    return df
except Exception as e:
    st.error(f'LỖI DB: Không thể đọc dữ liệu cảnh báo. {e}')
    return pd.DataFrame()

# Đã sửa lỗi: Dùng st.cache để tương thích với phiên bản 1.10.0
@st.cache(ttl=10, allow_output_mutation=True)
def load_data():
    """Hàm wrapper có cache để tải dữ liệu, giúp làm mới nhanh hơn."""
    return get_alerts_data()

def run_dashboard():
    st.set_page_config(layout="wide")
    st.title("SSH Anomaly Detection Dashboard (SIEM Lite)")

    if st.button('Làm mới Dữ liệu', key='refresh'):
        st.legacy_caching.clear_cache()

    df = load_data()

    if df.empty:
        st.warning("Database trống hoặc có lỗi khi tải dữ liệu.")
        return

    # --- 1. HIỂN THỊ CHỈ SỐ (KPIs) ---
    st.header("1. Tóm Tắt Hoạt động")

    col1, col2, col3, col4 = st.columns(4)

    df_display = df.copy()

```

```

df_display['is_handled'] = df_display['is_handled'].fillna('PENDING')

blocked_count = df_display[df_display['is_handled'] == 'BLOCKED'].shape[0]
ai_count = df_display[df_display['detection_type'] == 'AI'].shape[0]
critical_count = df_display[df_display['severity'] >= 9].shape[0]

col1.metric("Tổng Cảnh báo (Total)", df.shape[0])
col2.metric("Số vụ đã chặn (Blocked)", blocked_count)
col3.metric("Phát hiện AI (Severity 8)", ai_count)
col4.metric("Cảnh báo Nguy hiểm (Sev 9+)", critical_count)

st.markdown("---")

# --- 2. BIỂU ĐỒ PHÂN BỐ CẢNH BÁO ---
st.header("2. Phân Tích Dữ Liệu")

# Biểu đồ 1: Phân bố theo Mức độ Nghiêm trọng
st.subheader("Phân bố theo Mức độ Nghiêm trọng (Severity)")
severity_counts = df['severity'].value_counts().reset_index()
severity_counts.columns = ['Severity', 'Count']
st.bar_chart(severity_counts.set_index('Severity'))

# Biểu đồ 2: Hoạt động theo Giờ trong Ngày
st.subheader("Hoạt động theo Giờ trong Ngày")
df['hour'] = df['timestamp'].dt.hour
hourly_alerts = df.groupby('hour').size().reset_index(name='Count')
st.line_chart(hourly_alerts.set_index('hour'))

st.markdown("")

# --- 3. BẢNG CẢNH BÁO CHI TIẾT ---
st.header("3. Bảng Cảnh Báo Chi Tiết")

# Chuẩn bị dữ liệu hiển thị và lọc
df_display['is_handled'] = df_display['is_handled'].fillna('PENDING')

# Thêm bộ lọc
cols_filter = st.columns(2)

```

```
status_options = ['Tất cả'] + list(df_display['is_handled'].unique())
status_filter = cols_filter[0].selectbox('Lọc theo Trạng thái Xử lý:', status_options)

type_options = ['Tất cả'] + list(df_display['detection_type'].unique())
type_filter = cols_filter[1].selectbox('Lọc theo Loại Phát hiện:', type_options)

# Áp dụng bộ lọc
if status_filter != 'Tất cả':
    df_display = df_display[df_display['is_handled'] == status_filter]
if type_filter != 'Tất cả':
    df_display = df_display[df_display['detection_type'] == type_filter]

# Hiển thị bảng (Đã sửa lỗi use_container_width)
st.dataframe(df_display.drop(columns=['id']), height=500)

if __name__ == '__main__':
    run_dashboard()
```