졸업자격실험보고서

웹 로그 기반 LSTM - Logistic Regression를 이용한 APT 공격 탐지

APT Attack Detection Using a Web Log - Based LSTM - Logistic Regression

지도교수 장 유 진

동국대학교 WISE 캠퍼스 글로컬에너지대학

전자·정보통신공학과

오 세 창

2 0 2 5

웹 로그 기반 LSTM - Logistic Regression를 이용한 APT 공격 탐지

APT Attack Detection Using a Web Log - Based LSTM - Logistic Regression

오 세 창

지도교수 장 유 진

본 보고서를 졸업자격 실험보고서로 제출함
2025 년 12 월 10 일

오 세 창의 졸업자격 실험보고 통과를 인준함.
2025 년 12 월 10 일

주 심 장 유 진 (인)

부 심 강 법 주 (인)

부 심 반 상 우 (인)

동국대학교 WISE 캠퍼스 글로컬에너지대학
전자·정보통신공학과 졸업자격실험보고서

요 약 서

본 연구에서는 WAF 로그의 payload 문자열에 대한 정교한 의미 분석과 시계열 기반 행위 분석을 결합하기 위하여, BERT 임베딩과 LSTM 시퀀스 학습을 통합한 이중 단계(hybrid) 탐지 구조를 설계하였다. 1단계에서는 BERT 기반 문맥 임베딩을 입력으로 하여 Logistic Regression 분류기를 통해 탐지명(method)을 추정하고, 2단계에서는 요청 간 시간 간격(Δt)을 포함한 시퀀스 데이터를 LSTM에 입력하여 최종 공격 여부를 결정하도록 구성하였다. 학습에는 C-TAS에서 제공하는 침해사고 시나리오 빅데이터 중 랜섬웨어 설치 시나리오 및 10종의 비정상 행위 로그를 활용하여 모델의 실효성을 검증하였다.

Logistic Regression 기반 탐지명 분류기는 정확도 0.99, Macro-F1 0.72를 기록하였으며, SQL Injection, Information Leak 등 주요 공격 유형에서 우수한 분류 성능을 보였다. 이어서 LSTM 기반 공격 여부 분류 모델은 정확도 0.99997, Macro-F1 0.99978의 매우 높은 성능을 달성하였고, 전체 1,180건의 공격 로그를 모두 정확히 탐지하였으며 정상 로그 40,907건 중 오탐은 단 1건에 불과하였다. 이는 탐지율 100%, 오탐률 0.002% 미만으로, 정상 및 공격 로그를 거의 완벽하게 구분할 수 있음을 입증한다.

또한 본 연구는 행위 기반 탐지 방식의 특성상 기존 룰 기반 보안 시스템이 식별하기 어려운 새로운 형태의 공격 패턴도 효과적으로 탐지할 수 있음을 보여주었다. 특히 BERT의 문맥 이해 능력과 LSTM의 시계열 의존 관계 학습을 결합함으로써, 단일 로그만으로는 파악하기 어려운 연속적·단계적 공격 흐름을 정밀하게 포착할 수 있었다. 이러한 결과는 제안한 이중 단계 하이브리드 모델이 실환경에서 발생 가능한 다양한 공격 시나리오에 대해 높은 신뢰도로 대응할 수 있음을 시사한다.

목차

그림 목차

표 목차

I. 서 론

최근 지능형 지속 공격(Advanced Persistent Threat, APT)에 의한 사이버 공격 사례가 전 세계적으로 증가하고 있다.[1] APT 공격은 국가 기반 조직이나 주요 인프라를 대상으로 장기적이고 은밀한 침투를 시도하며, 기존 보안 체계로는 사전 탐지나 차단이 어렵다.[2]

현재 APT 방어의 핵심은 CERT(Computer Emergency Response Team)를 중심으로 한 지속적인 모니터링 및 로그 기반 탐지 체계이다. 그러나 이러한 방식은 구조적·운영적 한계를 지닌다.

정적 룰 기반 탐지 방식은 새로운 형태의 사이버 공격에 효과적으로 대응하지 못한다. 룰 기반 시스템은 미리 정의된 공격 패턴(signature)에 의존하기 때문에, 공격자가 기법을 변형하거나 알려지지 않은 공격 벡터를 사용하면 탐지할 수 없다.

이에 따라 다양한 로그 데이터를 통합 분석하여 공격의 행위적 패턴을 식별하는 행위 기반(behavior-based) 탐지 기법 연구가 활발히 진행되고 있다. 행위 기반 탐지는 네트워크 및 시스템에서 발생하는 일련의 이벤트 흐름을 분석하여 정상 사용자와 다른 비정상적 행위나 공격 단계 간 연관성을 파악한다. 이를 통해 기존 공격 패턴에 존재하지 않는 공격까지 탐지할 수 있다.

머신러닝과 딥러닝 모델을 활용하면 대규모 보안 로그로부터 특징(feature)을 자동으로 학습하고, 정상과 공격 패턴을 스스로 구분할 수 있다.

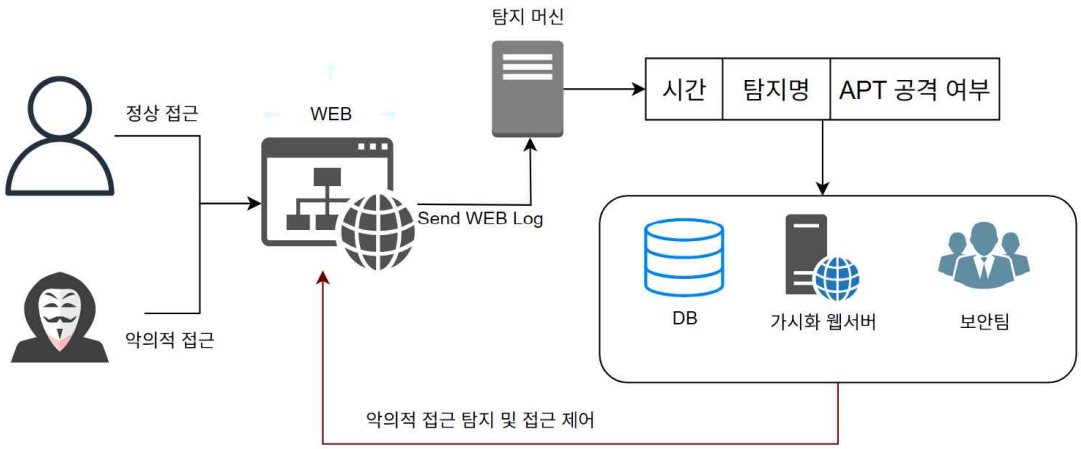
본 연구에서는 WAF 로그를 AI로 분석하는 탐지 시스템을 제안한다.

II. 본 론

2-1. 시스템 구성

본 탐지 모델은 웹 방화벽(WAF) 로그의 payload 문자열을 입력으로 하여, Transformer 기반 문맥 임베딩과 시퀀스 학습을 결합한 이중 단계 하이브리드 탐지 구조를 가진다. 1단계에서는 BERT 임베딩을 이용해 공격 탐지명을(method) 추정하고, 2단계에서는 해당 결과와 시간 간격(Δt)을 포함한 시퀀스 데이터를 LSTM에 입력하여, 최종 공격 여부(label_attack)를 예측한다.

2-2 시스템 구성도



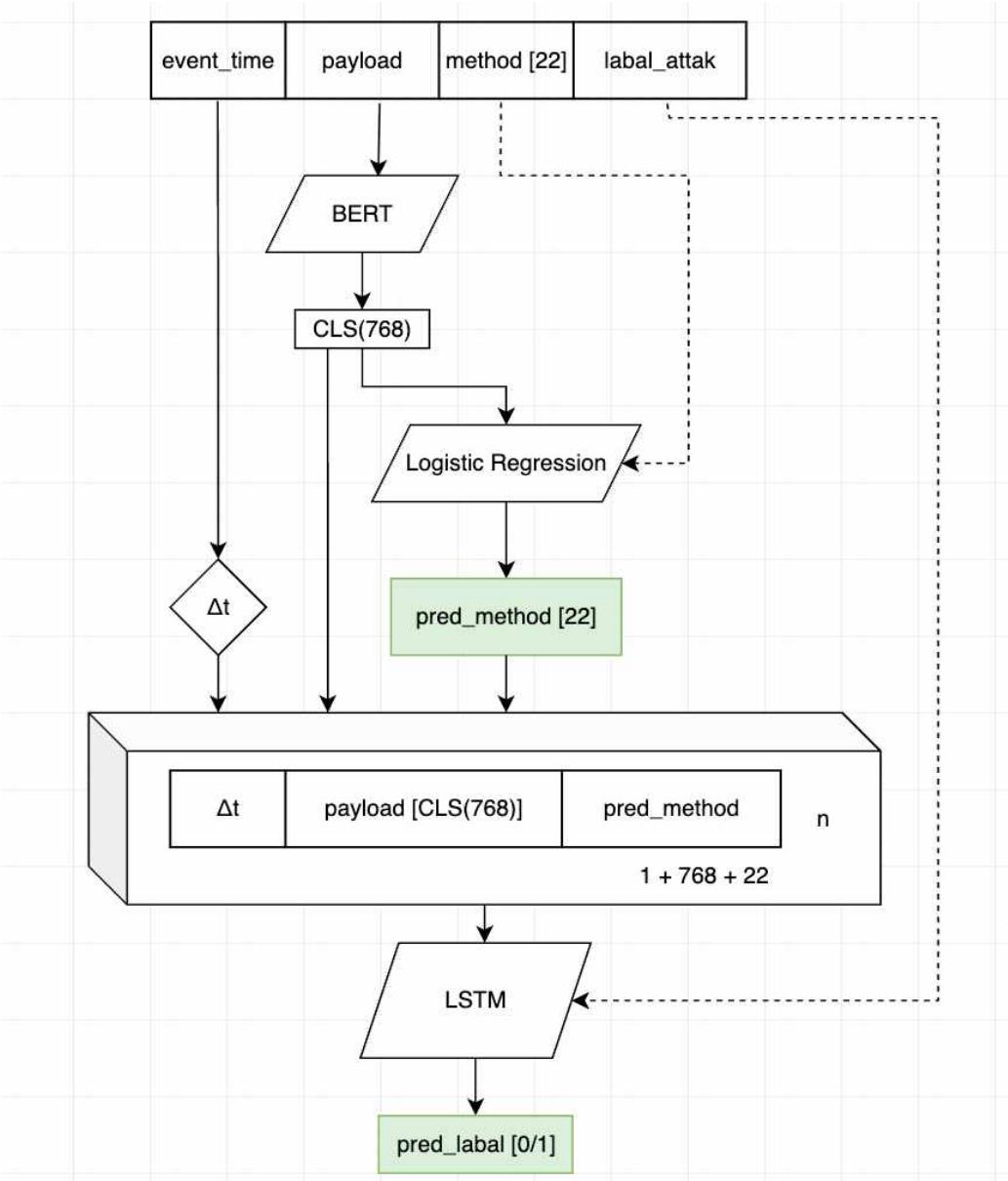
<그림-1> 시스템 구성도

<그림-1>은 제안한 시스템의 전체 구성도를 개략적으로 나타낸 것이다.

본 시스템은 크게 피해 서버, 분석 서버, 가시화 서버의 세 부분으로 구성된다. 피해 서버의 취약한 웹 서버에서 발생하는 access.log를 분석 서버로 전송하면, 분석 서버는 해당 로그를 기반으로 공격 탐지명(method)과 APT 공격 여부를 추론한다. 이후 추론된 결과와 관련 페이로드 정보는 Splunk와 같은 가시화 시스템으로

전달되며, 사용자는 이를 통해 공격 탐지 결과를 직관적으로 확인하고 보안 대응 판단을 수행할 수 있다.

2-3 시스템 파이프라인



<그림-2> 시스템 파이프라인

<그림-2>은 시스템 파이프라인를 간단하게 그림으로 표현한 것이다. 시스템은 크게 BERT 임베딩 모듈, Logistic Regression 기반 method 추론기, Δt(시간 간격) 계산기, LSTM 기반 공격 분류기로 구성된다.

2-4. 하드웨어 구성

<표-1>은 학습에 사용된 하드웨어에 대해 구체적으로 정리한 표이다.

<표-1> 하드웨어 사양

항목	사양
CPU	Intel Core i9-13900KF (24 cores)
GPU	NVIDIA RTX 4090 (24GB VRAM)
Memory	64GB DDR5
OS	Ubuntu 24.04 LTS
Python Version	3.12.3
Deep Learning Framework	PyTorch 2.3.1
Transformer Library	HuggingFace Transformers 4.43.1
ML Library	scikit-learn 1.5.0

2-5. 데이터셋 구성

본 연구에서 사용한 데이터셋은 C-TAS에서 제공한 침해사고 시나리오 빅데이터를 기반으로 구성되었다. 구체적으로 “[WAF] A사 랜섬웨어 설치 시나리오”와 “[WAF] 정상 행위에 포함되지 않는 10종의 비정상 행위”를 병합하여 입력 자료를 생성하였으며, 실험의 공정성과 데이터 누수를 방지하기 위해 중복 레코드를 제거하였다.

2-6. 입력 데이터 구조

<표-2>는 학습에 사용된 모델의 입력에 대해 구체적으로 정리한 표이다. 모델의 입력 데이터는 웹 방화벽 로그의 주요 필드로 구성된다.

<표-2> 입력 필드

필드명	설명
event_time	로그 발생 시각 (이전 이벤트와의 Δt 계산에 사용)
payload	요청의 URI 및 파라미터 문자열 (텍스트 임베딩 입력)
method	공격 탐지명(예: SQL Injection, XSS 등, 학습 시 정답 라벨)
label_attack	공격 여부 레이블 (0: 정상, 1: 공격, 학습 시 정답 라벨)

Ⅲ. 모델 구성 단계별 설명

3-1. 입력 데이터 구조

입력된 payload 문자열을 BERT(bert-base-uncased) 모델에 입력하여 문장의 문맥적 의미를 표현하는 CLS 벡터(768차원)를 추출한다. 이 CLS 임베딩은 페이로드의 명령 구조, 키워드, 문법 패턴 등을 반영한 의미 벡터로, 이후 분류기(Logistic Regression 및 LSTM)의 입력으로 사용된다.

3-2. Logistic Regression 기반 Method 분류기

BERT의 CLS 벡터를 입력받아 탐지명(method)을 예측한다. Logistic Regression은 선형 분류기로, BERT 임베딩의 의미 공간에서 공격 유형별 결정 경계를 학습한다. 학습 단계에서는 method 레이블을 정답으로 사용하고, 예측 단계에서는 pred_method (22차원 one-hot 벡터)를 생성한다.

3-3. 시간 특징(Δt) 계산기

두 로그 간 시간 간격을 계산한다. 계산된 Δt는 log1p(Δt) 변환 후 정규화되어, 요청 간 행동 주기 및 자동화 공격 패턴을 반영하는 정량적 특징으로 사용된다.

3-4. LSTM 입력 벡터 구성

<표-3>에서 세 가지 주요 특징(Δt_norm, CLS(768), pred_method(22))을 결합하여 최종 입력 벡터를 구성한다.

<표-3> 입력 필드 차원

구성 요소	차원
Δt_norm	1
CLS	768
pred_method	22
총 입력 차원	791 (= 1 + 768 + 22)

이러한 벡터가 시간 순서대로 누적되어 시퀀스 형태(길이 n)로 LSTM에 입력된다. 즉, 한 세션 내에서 발생한 연속 로그를 순서대로 분석하여 공격 행위의 전후 맥락을 파악한다.

3-5. LSTM 기반 공격 분류기

입력 시퀀스 [Δt_norm + CLS(768) + pred_method(22)]를 기반으로 공격 여부(label_attack)를 최종 예측한다. LSTM은 시계열 순서에 따라 로그 간 의존 관계(temporal dependency)를 학습하며, 단일 로그 분석으로는 놓치기 쉬운 행위 기반(behavioral) 공격 패턴을 포착한다.

Ⅳ. 실험 결과

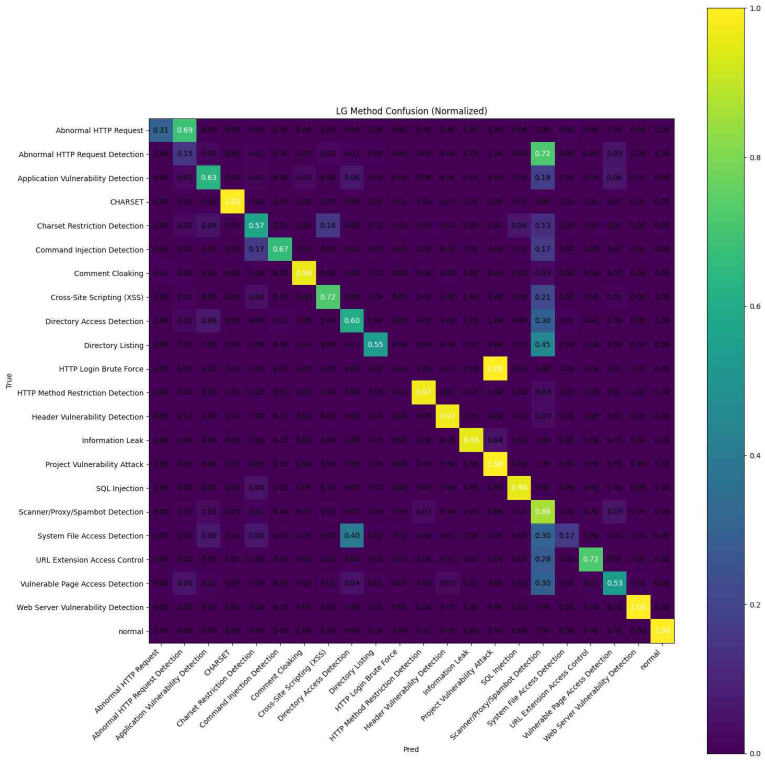
```
==== Method Classification (LG) ====
Accuracy   : 0.967646
Macro-F1   : 0.669689

                                precision    recall  f1-score   support

                Abnormal HTTP Request     0.9922    0.3055    0.4672       838
    Abnormal HTTP Request Detection     0.1057    0.1540    0.1253       461
Application Vulnerability Detection     0.5306    0.6341    0.5778        82
                            CHARSET     1.0000    1.0000    1.0000         6
       Charset Restriction Detection     0.8402    0.5749    0.6827       247
         Command Injection Detection     0.8571    0.6667    0.7500        18
                   Comment Cloaking     0.9912    0.9573    0.9739       234
          Cross-Site Scripting (XSS)     0.5094    0.7200    0.5967        75
         Directory Access Detection     0.5357    0.6000    0.5660       100
                   Directory Listing     0.6667    0.5455    0.6000        11
                HTTP Login Brute Force     0.0000    0.0000    0.0000         3
  HTTP Method Restriction Detection     0.7083    0.9714    0.8193        35
    Header Vulnerability Detection     0.8500    0.9714    0.9067        35
                     Information Leak     1.0000    0.9565    0.9778        92
        Project Vulnerability Attack     0.9894    1.0000    0.9947       654
                       SQL Injection     0.8730    0.9565    0.9129       115
    Scanner/Proxy/Spambot Detection     0.4205    0.8616    0.5652       448
       System File Access Detection     0.7000    0.1707    0.2745        82
       URL Extension Access Control     0.8667    0.7222    0.7879        36
  Vulnerable Page Access Detection     0.6447    0.5326    0.5833       184
Web Server Vulnerability Detection     0.4000    1.0000    0.5714         2
                            normal     1.0000    1.0000    1.0000     40410

                          accuracy                         0.9676     44168
                         macro avg     0.7037    0.6955    0.6697     44168
                      weighted avg     0.9777    0.9676    0.9679     44168
```

<그림-3> 성능 리포트



<그림-4> 혼동 행렬

<그림-3>은 Logistic Regression 기반 탐지명(method) 분류기의 성능 결과이다. <그림-3> 의 분류 리포트에서 확인할 수 있듯이, 전체 정확도는 0.99이며 Macro-F1은 0.72로 나타났다. Project Vulnerability Attack(F1=0.99), SQL Injection(F1=0.98), Information Leak(F1=0.99)과 같이 명확한 문법적·패턴적 특징을 가진 공격 유형에서는 높은 F1-score가 측정되었다. 반면, 데이터의 양이 부족하거나 표현적 유사도가 높은 System File Access Detection(F1=0.43), Command Injection Detection(F1=0.52) 등의 일부 클래스에서는 상대적으로 낮은 분류 성능이 확인되었다.

<그림-4>의 정규화된 혼동행렬에서도 이러한 경향이 확인되며, 대부분의 클래스가 대각선 영역에 높은 비율로 분포하여 안정적인 분류가 이루어졌음을 보여준다. 다만 서로 구조적으로 유사한 입력을 가진 일부 공격 유형 간에는 혼동이 발생하는 모습도 관찰된다. 전체적으로 정상(normal) 클래스는 Precision, Recall, F1-score 모두 1.00으로 매우 안정적으로 분류되며, 다양한 공격 유형에 대해서도 평균적으로 우수한 탐지 성능을 달성하였다.

```
==== Validation ====
Acc=0.999976, Macro-F1=0.999782

              precision    recall  f1-score   support

      attack       1.00      1.00      1.00      1180
      normal       1.00      1.00      1.00     40907

    accuracy                           1.00     42087
   macro avg       1.00      1.00      1.00     42087
weighted avg       1.00      1.00      1.00     42087

Confusion Matrix:
[[ 1180     0]
 [    1 40906]]
```

<그림-5> 성능 리포트

<그림-5>는 LSTM 기반 공격 여부(label_attack) 분류 모델의 검증 결과를 나타낸

다. 모델의 전체 정확도(Accuracy)는 0.99997, Macro-F1은 0.99978로 측정되어 전반적으로 매우 높은 탐지 성능을 보였다. Confusion Matrix를 기준으로 분석한 결과, 공격(attack) 1,180건은 모두 정확하게 탐지되었으며, 정상(normal) 40,907건 중 단 1건만이 공격으로 잘못 분류되었다. 이는 공격 탐지율(Recall) 100%를 달성함과 동시에, 오탐률(False Positive Rate)이 0.002% 미만으로 매우 낮음을 의미한다. 이러한 결과는 제안한 LSTM 기반 모델이 정상과 공격을 거의 완벽하게 구분할 수 있는 수준의 판별 성능을 갖추었음을 보여준다.

Ⅴ. 결론

본 연구에서는 웹 방화벽(WAF) 로그의 APT 공격을 효과적으로 분석하기 위해, Transformer 기반 임베딩(DeBERTa-v3), 로지스틱 회귀(Logistic Regression), 그리고 시계열 분류(LSTM)를 결합한 하이브리드 탐지 모델을 제안하였다. 먼저 WAF 로그의 payload를 언어모델로 임베딩하여 의미적·문맥적 특징을 강화하였으며, 이후 K-Means 군집화를 활용해 유사한 공격 행위를 잠재적 탐지명(method)으로 구조화하였다. 마지막 단계에서는 로그 간의 시간적 흐름과 순서를 고려한 LSTM 분류기를 적용하여 공격 여부를 정밀하게 판단하였다.

제안한 모델은 두 가지 측면에서 높은 성능을 보였다. 첫째, Logistic Regression 기반 탐지명 분류에서는 정확도 0.99, Macro-F1 0.72를 달성하며 주요 공격 유형을 안정적으로 구분하였다. 비록 일부 데이터가 적거나 표현이 유사한 클래스에서 낮은 성능을 보였으나, 전반적으로 정상 트래픽과 주요 공격 유형을 명확하게 판별하는 성능을 확인하였다. 둘째, LSTM 기반 공격 여부(label_attack) 분류에서는 정확도 0.99997, Macro-F1 0.99978을 기록하며, 공격 1,180건을 모두 탐지하고 정상 40,907건 중 단 1건만을 잘못 분류하는 등 매우 높은 수준의 신뢰성과 안정성을 보여주었다.

이러한 실험 결과는 제안한 하이브리드 접근이 단일 모델 기반 탐지 기법의 한계를 극복하며, 비정형 로그의 의미적 특징과 시계열적 맥락을 효과적으로 결합하는 데 유용함을 입증한다. 특히, 단계적 APT 공격이나 다단계 웹 공격과 같이 로그 간 연관성이 중요한 공격 시나리오에서 높은 활용 가능성을 확인하였다.

향후 연구에서는 IDS, 시스템 로그, 엔드포인트 로그 등 다양한 형태의 보안 로그를 통합한 멀티모달 분석 구조로 확장하고, 대규모 실환경 데이터셋 기반 검증과 실시간 탐지 환경 적용 가능성 분석을 통해 실서비스 수준의 안정성과 성능을 확보하는 연구를 진행하고자 한다.

참 고 문 헌

[1] advanced persistent threat (APT)

https://en.wikipedia.org/wiki/Advanced_persistent_threat 2025.11

[2] Trellix The Cyberthreat Report 2025.4

부록 1. 소스코드

```
# -*- coding: utf-8 -*-
"""
Hard-coded pipeline (clean):
payload -> BERT(CLS) -> (LR) method 추정
        -> [CLS + onehot(method) + Δt] 시퀀스 구성
        -> LSTM 으로 label_attack 분류
Artifacts:
  - cls_train.npy, cls_val.npy
  - Xtr_seq.npy, Ytr_seq.npy, Xvl_seq.npy, Yvl_seq.npy
  - method_report.txt, method_clf.joblib
  - lstm_label_attack.pt, meta.pkl
"""

from __future__ import annotations

import os, warnings
from pathlib import Path
from typing import List, Tuple

import numpy as np
import pandas as pd
from tqdm import tqdm

warnings.filterwarnings("ignore")

# =========================
# 전역 설정 (경로/칼럼/HP)
# =========================
TRAIN_CSV = "mnt/data/merged_WAF_noimal_attak_train_noleak.csv"
VAL_CSV   = "mnt/data/merged_WAF_noimal_attak_val_noleak.csv"
ART_DIR           =     Path("1109/apt_artifacts");     ART_DIR.mkdir(parents=True,
exist_ok=True)

PAYLOAD_COL = "payload"
METHOD_COL  = "method"
TIME_COL    = "event_time"
LABEL_COL   = "label_attack"

SEQ_LEN       = 10
```

```
SEQ_STRIDE    = 1

BERT_MODEL    = "bert-base-uncased"
MAX_LEN       = 64
EMB_BATCH     = 64

H_LSTM        = 256
LSTM_LAYERS   = 1
DROPOUT       = 0.1
EPOCHS_LSTM   = 15
BATCH_LSTM    = 128
LR_LSTM       = 1e-3
WEIGHT_DECAY = 1e-4

# (선택) BLAS 쓰레딩 과점유/행 방지
os.environ.setdefault("OMP_NUM_THREADS", "1")
os.environ.setdefault("MKL_NUM_THREADS", "1")

# =========================
# Torch / HF 로드
# =========================
import torch
import torch.nn as nn
from torch.utils.data import Dataset, DataLoader
from transformers import AutoTokenizer, AutoModel

DEVICE = "cuda" if torch.cuda.is_available() else "cpu"
print(f"[INFO] Device: {DEVICE}", flush=True)



# =========================
# 유틸 함수
# =========================
def load_csv(path: str) -> pd.DataFrame:
    """고정 경로 CSV 로드 (존재 확인 포함)."""
    p = Path(path)
    if not p.exists():
        raise FileNotFoundError(f"CSV not found: {p}")
    print(f"[LOAD] {p}", flush=True)
    return pd.read_csv(p)
```

```
def parse_time_series(series: pd.Series) -> pd.Series:
    """epoch ms/s 또는 문자열 타임스탬프를 UTC Timestamp로 파싱."""
    def _parse(x):
        if pd.isna(x): return pd.NaT
        try:
            v = float(x)
            if v > 1e12:  # ms
                return pd.to_datetime(int(v), unit="ms", utc=True)
            if v > 1e9:   # s
                return pd.to_datetime(int(v), unit="s", utc=True)
        except Exception:
            pass
        return pd.to_datetime(str(x), errors="coerce", utc=True)
    return series.apply(_parse)


def add_time_features_fixed(df: pd.DataFrame) -> pd.DataFrame:
    """Δt 특징 생성: 정렬→diff(sec)→log1p→표준화→_dt_norm."""
    df = df.copy()
    df["_ts"] = parse_time_series(df[TIME_COL])
    df.sort_values("_ts", inplace=True, kind="mergesort")  # 안정 정렬
    ts = (df["_ts"].astype("int64") // 10**9).astype("float64")
    ts = ts.fillna(ts.median() if not np.isnan(ts.median()) else 0.0)
    dts = ts.diff().fillna(0.0).clip(lower=0.0).astype(np.float32)

    dt_log = np.log1p(dts.values)
    mean, std = float(dt_log.mean()), float(dt_log.std())
    dt_norm = (dt_log - mean) / (std if std > 0 else 1.0)
    df["_dt_norm"] = dt_norm.astype(np.float32)
    return df


def compute_cls_embeddings(texts: List[str],
                           tokenizer: AutoTokenizer,
                           bert: AutoModel) -> np.ndarray:
    """BERT CLS 임베딩 계산."""
    vecs = []
    bert.eval()
    with torch.no_grad():
        for i in tqdm(range(0, len(texts), EMB_BATCH), desc="BERT CLS 임베딩"):
```

```
            batch = texts[i:i+EMB_BATCH]
            enc = tokenizer(batch, padding=True, truncation=True,
                            max_length=MAX_LEN, return_tensors="pt")
            enc = {k: v.to(DEVICE) for k, v in enc.items()}
            out = bert(**enc)
            cls = out.last_hidden_state[:, 0, :]  # [B, 768]
            vecs.append(cls.detach().cpu().numpy())
    return np.concatenate(vecs, axis=0).astype(np.float32)


def onehot(idx: np.ndarray, K: int) -> np.ndarray:
    oh = np.zeros((len(idx), K), dtype=np.float32)
    oh[np.arange(len(idx)), idx] = 1.0
    return oh


def build_sequences(X: np.ndarray, y: np.ndarray,
                    seq_len: int, stride: int) -> Tuple[np.ndarray, np.ndarray]:
    """슬라이딩 윈도우로 [N,T,D], [N] 구성 (라벨은 윈도우 마지막 시점)."""
    Xs, Ys = [], []
    N = len(X)
    if N < seq_len:
        return np.zeros((0, seq_len, X.shape[1]), dtype=np.float32), np.zeros((0,),
dtype=np.int64)
    for start in range(0, N - seq_len + 1, stride):
        end = start + seq_len
        Xs.append(X[start:end])
        Ys.append(y[end - 1])
    return np.stack(Xs), np.array(Ys, dtype=np.int64)


class SeqDataset(Dataset):
    def __init__(self, X: np.ndarray, y: np.ndarray):
        self.X = torch.from_numpy(X)
        self.y = torch.from_numpy(y).long()
    def __len__(self): return self.X.shape[0]
    def __getitem__(self, i): return self.X[i], self.y[i]


class LSTMClassifier(nn.Module):
    def __init__(self, input_dim: int, hidden: int, num_layers: int,
```

```
                    num_classes: int, dropout: float):
        super().__init__()
        self.proj = nn.Linear(input_dim, hidden)
        self.lstm = nn.LSTM(hidden, hidden, num_layers=num_layers,
batch_first=True,
                            dropout=(dropout if num_layers > 1 else 0.0))
        self.head = nn.Sequential(
            nn.LayerNorm(hidden),
            nn.Dropout(dropout),
            nn.Linear(hidden, num_classes)
        )
    def forward(self, x: torch.Tensor) -> torch.Tensor:
        z = self.proj(x.float())
        _, (hn, _) = self.lstm(z)
        return self.head(hn[-1])


# =========================
# 파이프라인 본문
# =========================
def main() -> None:
    # 1) CSV 로드
    print("[STEP] CSV 로드 중...", flush=True)
    train_df = load_csv(TRAIN_CSV)
    val_df   = load_csv(VAL_CSV)
    print(f"[DONE] Train={len(train_df)}, Val={len(val_df)} rows", flush=True)

    # 필수 칼럼 확인
    req_cols = {PAYLOAD_COL, METHOD_COL, TIME_COL, LABEL_COL}
    for name, df in [("train", train_df), ("val", val_df)]:
        miss = sorted(list(req_cols - set(df.columns)))
        if miss:
            raise ValueError(f"{name} CSV에서 누락된 칼럼: {miss}")

    # 2) 시간 특징
    print("[STEP] 시간 파싱 및 Δt 계산...", flush=True)
    train_df = add_time_features_fixed(train_df)
    val_df   = add_time_features_fixed(val_df)
    print("[DONE] 시간 특징 생성 완료", flush=True)

    # 3) BERT 임베딩
```

```
    print(f"[STEP] BERT 모델 로드: {BERT_MODEL}", flush=True)
    tokenizer = AutoTokenizer.from_pretrained(BERT_MODEL, use_fast=True)
    bert = AutoModel.from_pretrained(BERT_MODEL).to(DEVICE)
    print("[DONE] BERT 로드 완료", flush=True)

    print("[STEP] Train BERT 임베딩 계산...", flush=True)
    cls_tr = compute_cls_embeddings(train_df[PAYLOAD_COL].astype(str).tolist(),
tokenizer, bert)
    print(f"[DONE] Train 임베딩 shape: {cls_tr.shape}", flush=True)

    print("[STEP] Val BERT 임베딩 계산...", flush=True)
    cls_vl = compute_cls_embeddings(val_df[PAYLOAD_COL].astype(str).tolist(),
tokenizer, bert)
    print(f"[DONE] Val 임베딩 shape: {cls_vl.shape}", flush=True)

    np.save(ART_DIR / "cls_train.npy", cls_tr)
    np.save(ART_DIR / "cls_val.npy",   cls_vl)

    # 4) method 분류 (LR만 사용)
    print("[STEP] Method 분류 (LR) 학습...", flush=True)
    from sklearn.preprocessing import LabelEncoder
    from sklearn.linear_model import LogisticRegression
    from sklearn.metrics import f1_score, classification_report
    import joblib

    le_method = LabelEncoder()
    ytr_m = le_method.fit_transform(train_df[METHOD_COL].astype(str))
    yvl_m = le_method.transform(val_df[METHOD_COL].astype(str))

    lr = LogisticRegression(max_iter=1000)
    lr.fit(cls_tr, ytr_m)

    pred_tr_m = lr.predict(cls_tr)
    pred_vl_m = lr.predict(cls_vl)
    f1_lr = f1_score(yvl_m, pred_vl_m, average="macro")
    best_name, best_clf = "LR", lr

    print(f"[DONE] LR Macro-F1={f1_lr:.4f}", flush=True)

    # 리포트/클래시파이어 저장
    report = []
```

```
    report.append("==== Validation (method) ====")
    report.append(f"LR Macro-F1={f1_lr:.6f}\n")
    report.append("-- LR report --")
    report.append(classification_report(yvl_m, pred_vl_m,
target_names=le_method.classes_))
    (ART_DIR / "method_report.txt").write_text("\n".join(report), encoding="utf-8")
    print(f"[SAVE] {ART_DIR/'method_report.txt'}", flush=True)

    joblib.dump({"name": best_name, "model": best_clf, "classes_":
le_method.classes_},
                ART_DIR / "method_clf.joblib")
    print(f"[SAVE] {ART_DIR/'method_clf.joblib'}", flush=True)

    # 5) LSTM 입력 구성
    print("[STEP] LSTM 입력 구성...", flush=True)
    from sklearn.preprocessing import LabelEncoder as _LabelEncoder

    le_attack = _LabelEncoder()
    ytr_a = le_attack.fit_transform(train_df[LABEL_COL].astype(str))
    yvl_a = le_attack.transform(val_df[LABEL_COL].astype(str))
    C_attack = len(le_attack.classes_)

    K_method = len(le_method.classes_)
    oh_tr = onehot(pred_tr_m, K_method)
    oh_vl = onehot(pred_vl_m, K_method)

    xtr_full = np.concatenate([cls_tr, oh_tr,
train_df["_dt_norm"].values.reshape(-1, 1)], axis=1)
    xvl_full = np.concatenate([cls_vl, oh_vl, val_df["_dt_norm"].values.reshape(-1,
1)], axis=1)
    print(f"[INFO] Input dim={xtr_full.shape[1]}, K_method={K_method},
C_attack={C_attack}", flush=True)

    Xtr_seq, Ytr_seq = build_sequences(xtr_full, ytr_a, SEQ_LEN, SEQ_STRIDE)
    Xvl_seq, Yvl_seq = build_sequences(xvl_full, yvl_a, SEQ_LEN, SEQ_STRIDE)
    print(f"[DONE] TrainSeq={Xtr_seq.shape}, ValSeq={Xvl_seq.shape}", flush=True)

    np.save(ART_DIR / "Xtr_seq.npy", Xtr_seq)
    np.save(ART_DIR / "Ytr_seq.npy", Ytr_seq)
    np.save(ART_DIR / "Xvl_seq.npy", Xvl_seq)
    np.save(ART_DIR / "Yvl_seq.npy", Yvl_seq)
```

```
    print(f"[SAVE] X/Y seq saved to {ART_DIR}", flush=True)

    # 6) LSTM 학습
    print("[STEP] LSTM 학습 시작...", flush=True)
    train_loader = DataLoader(SeqDataset(Xtr_seq, Ytr_seq), batch_size=BATCH_LSTM, shuffle=True)
    val_loader = DataLoader(SeqDataset(Xvl_seq, Yvl_seq), batch_size=BATCH_LSTM, shuffle=False)

    model = LSTMClassifier(xtr_full.shape[1], H_LSTM, LSTM_LAYERS, C_attack, DROPOUT).to(DEVICE)
    optim = torch.optim.AdamW(model.parameters(), lr=LR_LSTM, weight_decay=WEIGHT_DECAY)
    crit  = nn.CrossEntropyLoss()

    from sklearn.metrics import accuracy_score, f1_score

    def eval_on(loader: DataLoader) -> Tuple[float, float]:
        model.eval()
        ys, ps = [], []
        with torch.no_grad():
            for xb, yb in loader:
                xb, yb = xb.to(DEVICE), yb.to(DEVICE)
                pred = model(xb).argmax(1)
                ys.append(yb.cpu().numpy()); ps.append(pred.cpu().numpy())
        y = np.concatenate(ys); p = np.concatenate(ps)
        acc = accuracy_score(y, p)
        f1m = f1_score(y, p, average="macro")
        return acc, f1m

    best_f1, best_state = -1.0, None
    for epoch in range(1, EPOCHS_LSTM + 1):
        model.train(); total = 0.0
        for xb, yb in train_loader:
            xb, yb = xb.to(DEVICE), yb.to(DEVICE)
            optim.zero_grad()
            loss = crit(model(xb), yb)
            loss.backward(); optim.step()
            total += float(loss.item()) * xb.size(0)
        train_loss = total / max(1, len(train_loader.dataset))
        acc, f1m = eval_on(val_loader)
```

```
        print(f"[EPOCH {epoch:02d}] loss={train_loss:.4f} | val_acc={{acc:.4f}} | val_f1={{f1m:.4f}}".format(acc=acc, f1m=f1m), flush=True)
        if f1m > best_f1:
            best_f1 = f1m
            best_state = {k: v.detach().cpu().clone() for k, v in model.state_dict().items()}

    if best_state is not None:
        model.load_state_dict({k: v.to(DEVICE) for k, v in best_state.items()})
    print("[DONE] 학습 완료", flush=True)

    # 7) 저장(모델/메타)
    torch.save(model.state_dict(), ART_DIR / "lstm_label_attack.pt")
    print(f"[SAVE] 모델 저장: {ART_DIR/'lstm_label_attack.pt'}", flush=True)

    meta = {
        "label_attack_classes": np.array(le_attack.classes_).tolist(),
        "method_classes":       np.array(le_method.classes_).tolist(),
        "best_method_model":    "LR",
        "const": {
            "PAYLOAD_COL": PAYLOAD_COL,
            "METHOD_COL":  METHOD_COL,
            "TIME_COL":    TIME_COL,
            "LABEL_COL":   LABEL_COL,
            "SEQ_LEN": SEQ_LEN,
            "SEQ_STRIDE": SEQ_STRIDE,
            "BERT_MODEL": BERT_MODEL,
            "MAX_LEN": MAX_LEN
        }
    }
    import joblib
    joblib.dump(meta, ART_DIR / "meta.pkl")
    print(f"[SAVE] {ART_DIR/'meta.pkl'}", flush=True)

    print("\n=== DONE ===", flush=True)
    print(f"Artifacts -> {ART_DIR}", flush=True)


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