Towards Real-Time Network Intrusion Detection With Image-Based Sequential Packets Representation 논문 GPU 서버 실습

스마트보안전공 한보윤

목차

I. 실습 개요

본 실습은 기존 논문에서 제안한 CNN 기반 네트워크 침입 탐지(NIDS) 방법론을 직접 구현하여 보는 데 목적이 있다. 논문에서는 네트워크 침입 탐지 시스템(NIDS)의 성능과 탐지 효율성을 높이기 위해, 순차적 패킷 데이터를 이미지 형태로 표현한 후 CNN 모델을 이용하여 공격을 탐지하는 새로운 방법을 제안하였다. 이 방법의 핵심은 각 패킷의 헤더 정보, 페이로드, 그리고 패킷 간의 시간차를 시각적으로 표현하여 네트워크 공격 특유의 미세한 패턴을 포착할 수 있다는 데 있다.

논문에서 제시한 방법은 크게 다음의 세 가지 주요 요소로 구성된다.

1. Packet Parser : 원본 PCAP 형식의 네트워크 데이터를 처리하여, 각 패킷의 헤더 정보, 페이로드 및 패킷 간 시간 차이를 포함한 주요 특징을 추출한다.

2. Image Builder : 패킷 파서에서 추출한 데이터를 시각적 이미지로 변환하여 네트워크 공격의 특징적인 패턴을 표현한다.

3. Network Intrusion Detector(NID) : 생성된 이미지 데이터를 입력으로 사용하여 CNN 모델을 학습시키고, 이를 통해 정상적인 트래픽과 악의적인 공격을 효과적으로 구분한다.

실습은 현실 네트워크 환경을 반영한 CICIDS2017 데이터셋을 사용하여 진행되었으며, 논문에서 제시한 방법론이 실제 데이터셋을 이용하여 어떤 과정으로 적용될 수 있는지를 구체적으로 확인하고자 한다.

II. 논문 실습 구상

1) 기존 논문에서는 네트워크 침입 탐지를 과정을 크게 세 가지로 구분하였으며, 각각 패킷 파서(packet parser), 이미지 빌더(image builder), 네트워크 침입 탐지기(Network Intrusion Detector)로 칭하였다.

2) Packet Parser

패킷 파서는 원본 PCAP 데이터를 입력받아 헤더(Header) 정보와 델타 시간(Delta Time), 페이로드(Payload)를 추출하도록 구현하였다. 논문에서 제안한 방법을 바탕으로, dpkt와 scapy 라이브러리를 이용하여 TCP 프로토콜 기반의 패킷 데이터를 파싱하

였다.

각 패킷은 최대 1594바이트 크기의 데이터를 포함하고 있으며, 본 실습에서는 네트워크 환경 및 프로토콜에 의한 편향을 방지하기 위해 ETH(이더넷) 헤더(14바이트), IP 버전(1바이트), 차별화 서비스 필드(1바이트), 프로토콜(1바이트), 소스 및 대상 IP 주소(각 4바이트), 소스 및 대상 포트(각 2바이트), IP 옵션과 TCP 옵션(각 40바이트)을 포함한 총 109바이트를 제거하였다.

이후 최종적으로 사용되는 패킷 헤더 데이터는 25바이트이며, 델타 시간(1바이트), 페이로드(최대 1460바이트)와 함께 사용된다. 페이로드가 1460바이트보다 작은 경우에는 패딩을 적용하여 총 1486바이트로 고정하였다. 또한, 각 패킷에 대해 다음과 같은 7가지 보조 피처(feature)를 함께 추출하였다.

- Source IP
- Destination IP
- Source Port
- Destination Port
- Epoch Time
- Protocol
- Direction

이 과정은 packet_parser.py 파일을 통해 구현되었으며, 각 요일별로 CSV 파일에 7가지 보조 피처를 저장하고, NPZ 파일 형식으로 벡터화된 1486 바이트 데이터를 저장한다.

3) Image builder

이미지 빌더는 패킷 파서에서 생성된 NPZ 파일과 CICIDS2017 데이터셋 내 CSV 파일의 오류를 수정한 Clean CSV 파일을 사용하여 RGB 이미지로 변환하는 과정이다. Clean CSV 파일은 흐름(Flow)이 정상 또는 특정 공격 유형에 속하는지에 대한 라벨 정보를 제공한다.

생성된 이미지의 파일명에는 흐름 정보를 포함시켜 데이터셋을 나눌 때 (Train/Validation/Test) 동일한 흐름의 이미지가 다른 데이터셋으로 분리되지 않도록 하였다.

이미지는 각 흐름 내의 패킷 수에 따라 생성되며, 이미지 한 장에 포함되는 최대 패킷 수는 논문에서 제시한 기준을 따라 15개로 설정하였다. 흐름당 이미지 개수는 다

음과 같은 방식으로 정해진다.

- 흐름 내의 패킷 수가 15개 이하인 경우: 첫 번째 패킷부터 마지막 패킷까지 패킷 하나가 추가될 때마다 이미지를 생성한다.
- 흐름 내의 패킷 수가 15개를 초과하는 경우: 처음 15개 패킷까지만 사용하여 15장의 이미지를 생성한다.

예를 들어,

- 흐름 A에 포함된 패킷 수가 **5개**일 경우
  → 총 5장의 이미지가 생성됨 (패킷 1개를 사용한 이미지, 패킷 2개를 사용한 이미지, ..., 패킷 5개를 사용한 이미지)
- 흐름 B에 포함된 패킷 수가 **17개**일 경우
  → 첫 15개의 패킷만을 사용하여 총 15장의 이미지가 생성됨 (패킷 1개를 사용한 이미지, 패킷 2개를 사용한 이미지, ..., 패킷 15개를 사용한 이미지)

이미지는 요일별 폴더 하위에 라벨 및 패킷 수에 따라 저장되며, 각 이미지는 Forward(fwd) 패킷과 Backward(bwd) 패킷을 RGB 채널을 구분하여 표현하였다. Direction 값이 0이면 Forward 패킷으로 간주하여 R 채널에 매핑하고, Direction 값이 1이면 Backward 패킷으로 간주하여 G 채널에 매핑하였고, 사용하지 않은 B 채널은 0으로 채웠다. 생성된 모든 이미지는 동일한 크기(15×1486)를 갖도록 패딩을 적용하여 일관된 사이즈를 유지하였다.

이미지 생성 후, 학습을 위해 데이터셋을 Train, Validation, Test로 각 70:15:15 비율로 분할하였다.

4) NID

기존 논문에서는 네트워크 트래픽을 정상과 악성으로만 구분하는 이진(Binary Class) 분류 모델을 제안하였다. 그러나 본 실습에서는 공격 유형을 보다 구체적이고 정밀하게 식별하기 위해 다중 클래스(Multiclass) 형태로 모델을 변형하여 구현하였다. 이를 통해 다양한 공격 유형을 보다 세밀하게 탐지할 수 있도록 하였다.

최종 학습 결과는 Confusion Matrix로 시각화하여 모델 성능을 직관적으로 확인할 수 있도록 구현하였다.

III. 코드 소개 및 실행 과정

0. 경로 저장은 다음과 같이 한다.

```
.
├── SPIN_IDS
│   ├── clean_csv
│   └── pcap
├── packet_parser.py
├── match_clean.py
├── image_builder.py
├── compute_image_counts.py
├── view_image.py
├── train_val_test.py
├── bruteforce_train_val_test.py
└── cnn_multiclass.py
```

모든 python 코드의 실행은 'python3 [파일명]' 형식을 사용하여 실행했다.

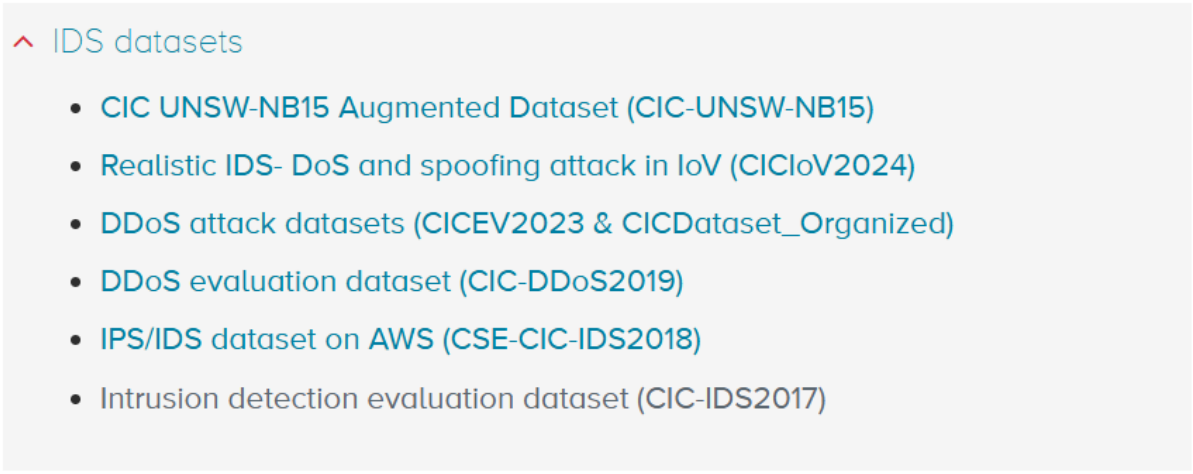
1. Dataset 준비

학습에 필요한 데이터셋을 다운받는다. CICIDS2017 데이터를 사용했다.

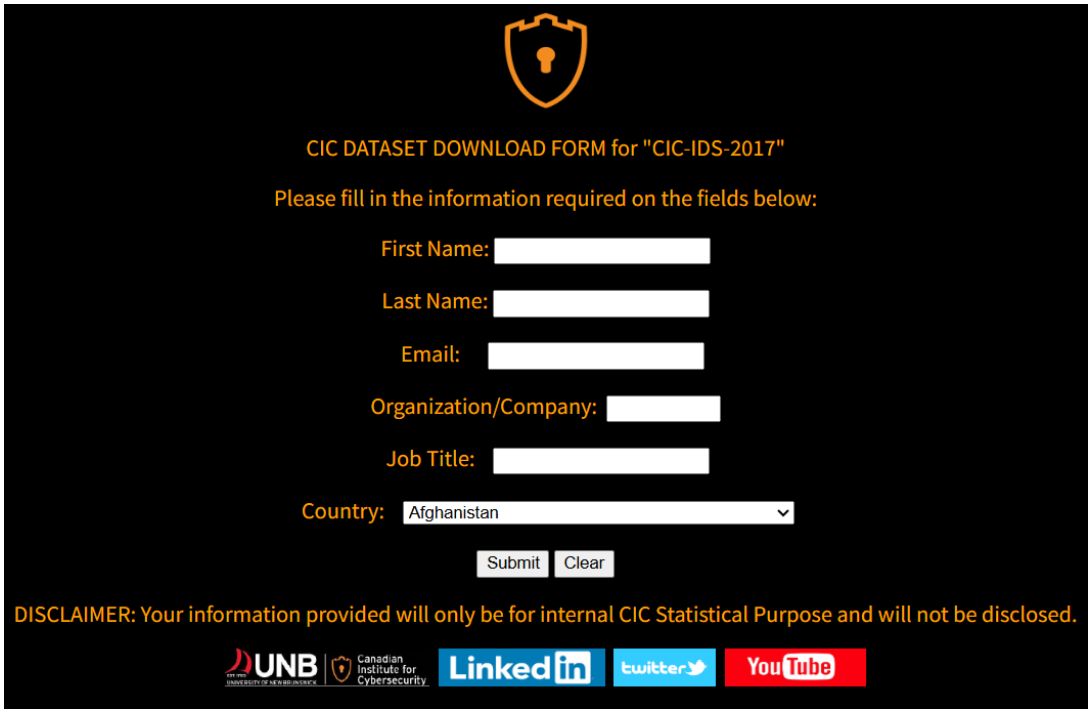
1) 아래 링크를 클릭하여 사이트에 들어간다.

Datasets | Research | Canadian Institute for Cybersecurity | UNB

2) Available datasets 중 IDS datasets 항목의 CICIDS2017을 클릭한다.



3) 페이지의 가장 아래로 내려가 Download this dataset을 클릭하고 이어 뜨는 폼 작성 후 제출하여 데이터셋을 다운로드 한다.



4) 다운로드가 끝난 후 파일을 열어보면 아래와 같이 구성되어 있다.

MachineLearningCSV	2025-05-18 오후 11:21	파일 폴더	
GeneratedLabelledFlows	2025-05-18 오후 11:22	파일 폴더	
Thursday-WorkingHours.pcap	2025-05-17 오후 4:04	Wireshark capture...	8,107,911KB
Friday-WorkingHours.pcap	2025-05-17 오후 4:07	Wireshark capture...	8,632,138KB
Tuesday-WorkingHours.pcap	2025-05-17 오후 4:19	Wireshark capture...	10,789,340...
Monday-WorkingHours.pcap	2025-05-17 오후 4:19	Wireshark capture...	10,568,855...
Wednesday-workingHours.pcap	2025-05-18 오후 9:21	Wireshark capture...	13,106,240...

이 중 pcap 파일 5개만 사용한다.

5) Clean csv 파일 다운로드

논문에서는 CICIDS2017 데이터셋의 CSV 파일의 오류를 발견한 아래 연구를 참조하였다.

G. Engelen, V. Rimmer, and W. Joosen, "Troubleshooting an intrusion detection dataset: The CICIDS2017 case study," in Proc. IEEE Secur. Privacy Workshops, 2021, pp. 7–12.

1)~4)번의 과정을 거쳐 다운로드한 데이터셋에 포함된 CSV 파일의 추출 방식에 오류가 있다는 내용의 연구이다.

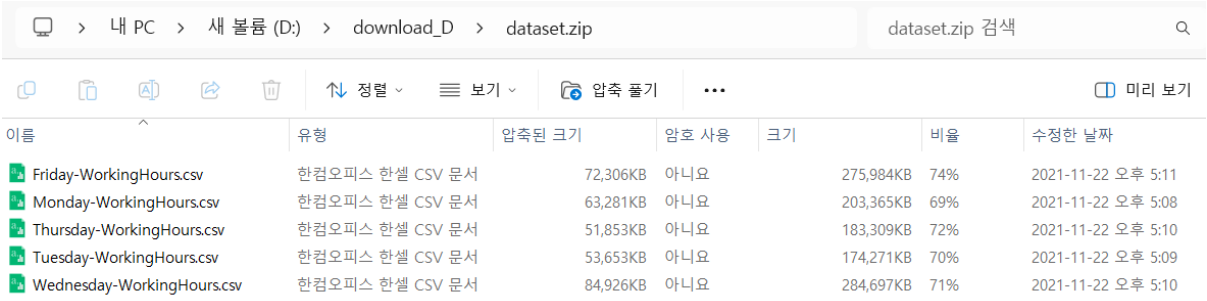
이에 해당 연구에서 제시한 정제된 새 CSV파일은 https://intrusion-detection.distrinet-research.be/WTMC2021/tools_datasets.html 에서 다운로드 받을 수 있다.

Improved CICIDS2017 dataset for flow-based network intrusion detection

The latest version of our improved version of the CICIDS2017 flow-based dataset can be downloaded here.

20/10/2021 UPDATE: *Dataset files reuploaded (fixed error in Idle Time features). Note that the fixed version of the CICFlowMeter tool is not affected.*
22/10/2021 UPDATE: *CICFlowMeter fixed as per this GitHub pull request (Fwd and Bwd Bulk features affected). Dataset CSV files regenerated and reuploaded.*
24/11/2021 UPDATE: *CICFlowMeter fixed as per this and this GitHub pull request (Down/Up ratio fixed and several features related to flow length affected). Dataset CSV files regenerated and reuploaded.*

Improved CICIDS2017 dataset for flow-based network intrusion detection 문단의 링크 중 'here'을 눌러 dataset.zip을 다운로드 받을 수 있다.

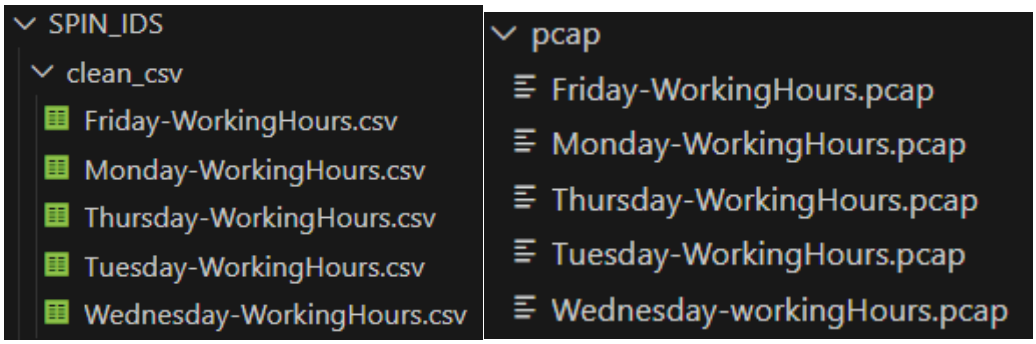


다운로드가 완료되면 사진과 같이 5개의 csv 파일이 있는 것을 확인할 수 있다.

6) 서버와 연결된 vscode로 다운받은 데이터셋들을 업로드한다.

SPIN_IDS폴더를 두고 필요한 데이터셋들을 SPIN_IDS 하위 폴더에 각각 묶어 업로드했다.

csv 파일들과 pcap 파일들을 묶는 폴더 이름은 가시성을 위해 각각 clean_csv와 pcap으로 설정했다.



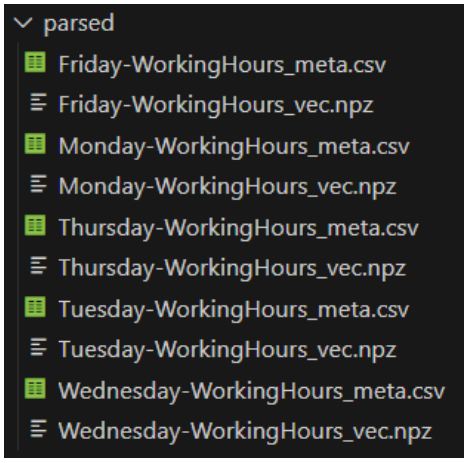
2. Packet_parser.py

1) Raw packet(원본 pcap)에서 필요한 데이터를 꺼내 csv와 npz 파일에 저장하는 코드이다.

```
import os, glob
import dpkt
import numpy as np
import pandas as pd
from scapy.all import Ether, IP, TCP

PCAP_DIR  = "SPIN_IDS/pcap"
PARSE_OUT = "SPIN_IDS/parsed"
N         = 1486
HEADER_LEN= 25
DT_IDX    = HEADER_LEN

os.makedirs(PARSE_OUT, exist_ok=True)

def parse_pcap(pcap_path):
    packets = []
    prev_ts = None
    with open(pcap_path, 'rb') as f:
        try:
            reader = dpkt.pcap.Reader(f)
        except (ValueError, dpkt.dpkt.NeedData):
            f.seek(0)
            reader = dpkt.pcapng.Reader(f)
        for ts, buf in reader:
            try:
                eth = Ether(buf)
            except Exception:
                continue
            if not eth.haslayer(IP) or not eth.haslayer(TCP):
                continue
            ip  = eth[IP]
            tcp = eth[TCP]
            ts = float(ts)
            eth_pkt = dpkt.ethernet.Ethernet(buf)
            ip_pkt  = eth_pkt.data
            tcp_pkt = ip_pkt.data
            hdr_len = ip_pkt.hl*4 + tcp_pkt.off*4
            hdr = buf[14:14+hdr_len]
            payload = bytes(tcp.payload) or b''

            V = np.zeros(N, dtype=np.uint8)
            hl = min(len(hdr), HEADER_LEN)
```

```
45.            V[:hl] = np.frombuffer(hdr, dtype=np.uint8, count=hl)
46.            dt = 0 if prev_ts is None else min(int((ts-prev_ts)*1000),255)
47.            prev_ts = ts
48.            V[DT_IDX] = dt
49.            start = HEADER_LEN+1
50.            L = min(len(payload), N-start)
51.            if L>0:
52.                V[start:start+L] = np.frombuffer(payload[:L],
    dtype=np.uint8)
53.
54.            src, dst = ip.src, ip.dst
55.            sp, dp   = tcp.sport, tcp.dport
56.            dr       = 0 if (src,sp)<=(dst,dp) else 1
57.            packets.append((V, (src, dst, sp, dp, 6, ts, dr)))
58.    return packets
59.
60.if __name__ == '__main__':
61.    for pcap in sorted(glob.glob(os.path.join(PCAP_DIR, "*.pcap*"))):
62.        day = os.path.splitext(os.path.basename(pcap))[0]
63.        meta_csv = os.path.join(PARSE_OUT, f"{day}_meta.csv")
64.        vec_npz  = os.path.join(PARSE_OUT, f"{day}_vec.npz")
65.        print(f"▶ Parsing {day}...")
66.        packets = parse_pcap(pcap)
67.
68.        df = pd.DataFrame([{
69.            'src_ip':   a[0], 'dst_ip': a[1],
70.            'src_port': a[2], 'dst_port': a[3],
71.            'proto':    a[4], 'timestamp': a[5], 'direction': a[6]
72.        } for _, a in packets])
73.        df.to_csv(meta_csv, index=False)
74.
75.        Vs = np.stack([v for v,_ in packets])
76.        np.savez_compressed(vec_npz, V=Vs)
77.        print(f"  ✓ {len(packets)} packets → {meta_csv}, {vec_npz}")
```

2) 실행 결과 : SPIN_IDS 폴더 하위에 사진과 같이 저장된다.



SPIN_IDS > parsed > Friday-WorkingHours_meta.csv

```
1  src_ip,dst_ip,src_port,dst_port,proto,timestamp,direction
2  192.168.10.50,192.168.10.3,56108,3268,6,1499428790.315195,1
3  192.168.10.50,192.168.10.3,56108,3268,6,1499428790.3152,1
4  192.168.10.3,192.168.10.50,3268,56108,6,1499428790.315459,0
5  192.168.10.3,192.168.10.50,3268,56108,6,1499428790.315463,0
6  192.168.10.50,192.168.10.3,56108,3268,6,1499428790.315541,1
7  192.168.10.50,192.168.10.3,56108,3268,6,1499428790.315545,1
```

csv 파일에 저장된 데이터 예시이다.

Friday-WorkingHours_meta.csv 파일 상단의 데이터를 캡처하였다.

3. Match_clean.py (필수로 진행해야하는 과정이 아닌, 단순 튜플 5개만으로 흐름을 그룹화했을 때 나타나는 오류를 발견하기 위한 과정이다.)

1) 튜플 5개(src_ip, dst_ip, src_port, dst_port, proto)만으로 clean csv와 parsed_meta csv를 매칭하고 flow별로 그룹화해 패킷 개수와 라벨을 저장하는 코드이다.

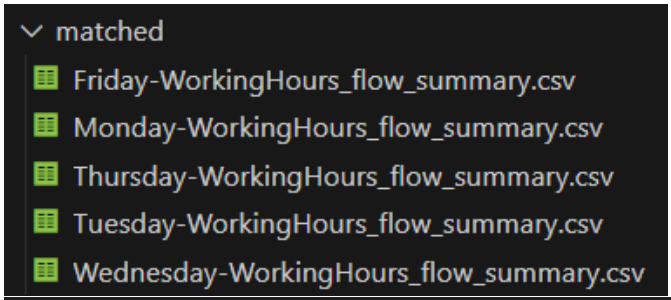
```python
import os
import glob
import pandas as pd
from tqdm import tqdm

PARSE_DIR = "SPIN_IDS/parsed"
CLEAN_DIR = "SPIN_IDS/clean_csv"
OUT_DIR   = "matched"
os.makedirs(OUT_DIR, exist_ok=True)

def load_labels(clean_csv_path):
    df = pd.read_csv(clean_csv_path)

    src = next(c for c in df.columns if 'src ip' in c.lower())
    dst = next(c for c in df.columns if 'dst ip' in c.lower())
    sport = next(c for c in df.columns if 'src port' in c.lower())
    dport = next(c for c in df.columns if 'dst port' in c.lower())
    proto = next((c for c in df.columns if 'protocol' in c.lower()), None)
```

```
    lbl = next(c for c in df.columns if 'label' in c.lower())

    df['flow_key'] = list(zip(
        df[src], df[dst],
        df[sport].astype(int), df[dport].astype(int),
        df[proto].astype(int) if proto else 6
    ))
    label_map = {
        k: v for k, v in zip(df['flow_key'], df[lbl])
    }

    rev_map = {(dst_ip, src_ip, dpt, spt, pr): lab
               for (src_ip, dst_ip, spt, dpt, pr), lab in label_map.items()}
    label_map.update(rev_map)
    return label_map

if __name__ == '__main__':
    clean_files = sorted(glob.glob(os.path.join(CLEAN_DIR, '*.csv')))
    for clean_csv in clean_files:
        day = os.path.basename(clean_csv).replace('.csv','')
        print(f"Processing {day} flows…")

        label_map = load_labels(clean_csv)

        parsed_file = os.path.join(PARSE_DIR, f"{day}_meta.csv")
        if not os.path.exists(parsed_file):
            parsed_file = glob.glob(os.path.join(PARSE_DIR, f"{day}*_meta.csv"))
            if parsed_file:
                parsed_file = parsed_file[0]
            else:
                print(f"  parsed_meta not found for {day}")
                continue
        parsed = pd.read_csv(parsed_file)

        parsed['flow_key5'] = list(zip(
            parsed.src_ip, parsed.dst_ip,
            parsed.src_port, parsed.dst_port,
            parsed.proto
        ))

        flow_groups = parsed.groupby('flow_key5').size().reset_index(name='packet_count')

        flow_groups['label'] = flow_groups['flow_key5'].map(label_map).fillna('Unknown')

        out_csv = os.path.join(OUT_DIR, f"{day}_flow_summary.csv")
        flow_groups.to_csv(out_csv, index=False)
        print(f"  Saved flow summary: {out_csv}  ({len(flow_groups)} flows)")

```

2) 출력 결과

```
Processing Friday-WorkingHours flows…
  Saved flow summary: matched/Friday-WorkingHours_flow_summary.csv  (587699 flows)
Processing Monday-WorkingHours flows…
  Saved flow summary: matched/Monday-WorkingHours_flow_summary.csv  (262917 flows)
Processing Thursday-WorkingHours flows…
  Saved flow summary: matched/Thursday-WorkingHours_flow_summary.csv  (287874 flows)
Processing Tuesday-WorkingHours flows…
  Saved flow summary: matched/Tuesday-WorkingHours_flow_summary.csv  (216049 flows)
Processing Wednesday-WorkingHours flows…
  Saved flow summary: matched/Wednesday-WorkingHours_flow_summary.csv  (234947 flows)
```

3) 실행 결과 : 요일별 csv 파일이 저장된다.



matched > Friday-WorkingHours_flow_summary.csv

```
1  flow_key5,packet_count,label
2  "('1.1.70.73', '192.168.10.15', 80, 52756, 6)",5,BENIGN
3  "('1.1.70.73', '192.168.10.15', 80, 52760, 6)",4,BENIGN
4  "('1.1.70.73', '192.168.10.15', 80, 52761, 6)",4,BENIGN
5  "('1.1.70.73', '192.168.10.15', 80, 52762, 6)",4,BENIGN
6  "('1.1.70.73', '192.168.10.15', 80, 52763, 6)",4,BENIGN
7  "('1.1.70.73', '192.168.10.15', 80, 52764, 6)",4,BENIGN
8  "('1.1.70.73', '192.168.10.15', 80, 52765, 6)",4,BENIGN
```

4) 위 과정으로 csv 파일에 저장된 데이터를 통해, 튜플-5 기준만으로 라벨을 할당할 시에 공격 라벨이 잘못 할당되는 경우가 있는 것을 확인하였다.

   아래는 그 예시이다.

① Wednesday **clean_csv 파일**에 저장된 라벨 -> slowloris, Hulk 두 종류의 라벨 저장

172.16.0.1-192.168.10.50-53816-80-6,172.16.0.1,53816,192.168.10.50,80,6,05/07/2017 **02:48:54 PM, DoS slowloris**

172.16.0.1-192.168.10.50-53816-80-6,172.16.0.1,53816,192.168.10.50,80,6,05/07/2017 **04:00:38 PM**, **DoS Hulk**

② Wednesday **matched csv 파일**에 저장된 라벨 -> Hulk 한 가지 라벨만 저장

"(**'172.16.0.1', '192.168.10.50', 53816, 80, 6**)",118,**DoS Hulk**

문제점 : 다른 요일 혹은 같은 요일일 때도 다른 시간대에 같은 주소를 사용하여 다른 종류의 공격/정상 패킷을 보내는 경우가 있어 흐름을 정리할 때 혼동되는 경우가 있었다.

해결책 : src ip, dst ip, src port, dst port, protocol의 5-튜플을 먼저 수집하고 그 튜플에 해당하는 라벨을 찾아 붙이는 것이 아니라, 순서를 바꾸어 먼저 라벨을 확인하고나서 5-튜플에 대한 정보를 수집하여 같은 5-튜플이더라도 라벨이 다르면 다른 흐름으로 수집되도록 수정하였다. (수정된 코드는 Image_builder.py 코드에 포함되어있다.)

이처럼 다른 시간 대에 같은 5-튜플을 사용하는 경우를 고려하여, 같은 5-튜플이더라도 라벨이 다르면 다른 흐름으로 처리하는 로직을 image_builder.py에 추가하였다.

4. Image_builder.py (3번 코드의 오류를 개선하고, 더불어 이미지까지 생성하는 코드이다.)

1) Parsed의 벡터 파일(.npz)과 clean_csv 파일(.csv)을 통해 이미지를 생성하는 코드이다.

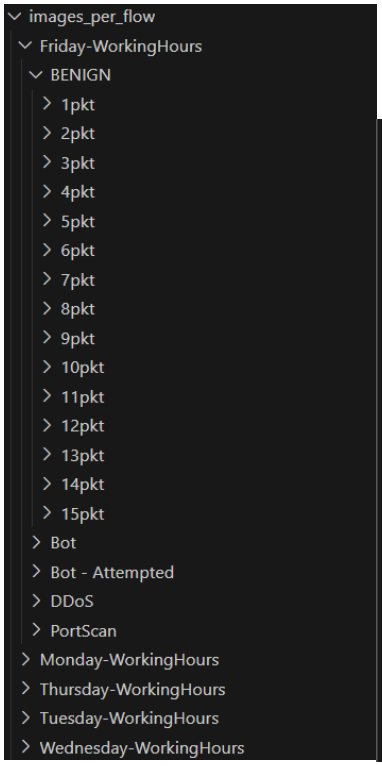
```
import os, glob
import numpy as np
import pandas as pd
from tqdm import tqdm

PARSED_DIR     = "SPIN_IDS/parsed"
SUMMARY_DIR    = "SPIN_IDS/clean_csv"
IMAGE_ROOT     = "SPIN_IDS/images_per_flow"
META_SUFFIX    = "_meta.csv"
VEC_SUFFIX     = "_vec.npz"
SUMMARY_SUFFIX = ".csv"

P = 15

def canonical_flow_id(src_ip, dst_ip, src_port, dst_port, proto):

    a, b = (src_ip, src_port), (dst_ip, dst_port)
    if a <= b:
        return (src_ip, dst_ip, src_port, dst_port, proto)
    else:
        return (dst_ip, src_ip, dst_port, src_port, proto)

def parse_flow_key(key_str):

    src, dst, sp, dp, pr = key_str.split('-')
    return canonical_flow_id(src, dst, int(sp), int(dp), int(pr))

def build_images_for_day(parsed_meta_csv, vec_npz, summary_csv, base_out):

    df_meta = pd.read_csv(parsed_meta_csv)
    df_meta['packet_idx'] = df_meta.index
    df_meta['flow_id'] = df_meta.apply(
        lambda r: canonical_flow_id(
            r['src_ip'], r['dst_ip'],
            r['src_port'], r['dst_port'],
            r['proto']
        ),
        axis=1
    )

    data = np.load(vec_npz)
    V_all = data['V']
    width = V_all.shape[1]

    df_sum = pd.read_csv(summary_csv)
    df_sum['flow_id'] = df_sum['Flow ID'].apply(parse_flow_key)

    df_labels = df_sum[['flow_id','Label']].drop_duplicates()

    df = pd.merge(df_meta, df_labels, on='flow_id', how='inner')
    grouped = df.groupby(['flow_id','Label'])

    day = os.path.basename(parsed_meta_csv).replace(META_SUFFIX, '')
```

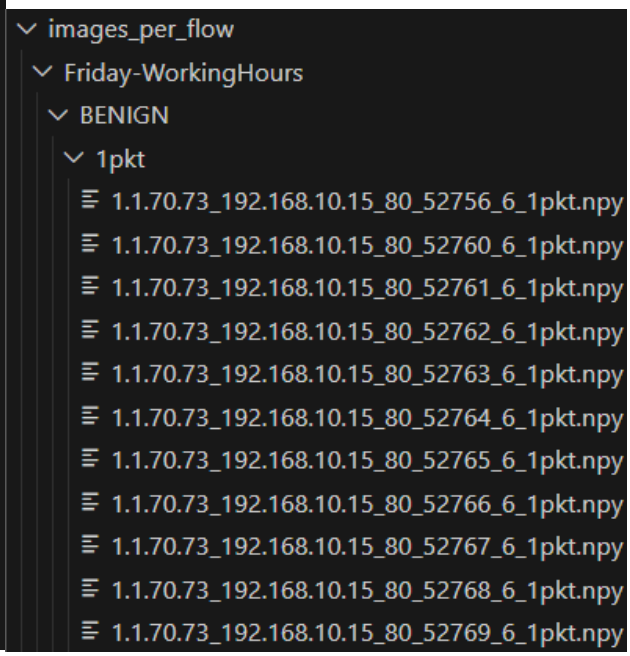
```
    for (flow_id, label), grp in tqdm(grouped, desc=f"Build {day}"):

        grp = grp.sort_values('timestamp')
        idxs = grp['packet_idx'].to_numpy()
        Vf   = V_all[idxs]
        dirs = grp['direction'].to_numpy()

        max_k = min(P, len(Vf))

        for k in range(1, max_k+1):
            img = np.zeros((P, width, 3), dtype=np.uint8)
            for j in range(k):
                ch = 0 if dirs[j]==0 else 1
                img[j, :, ch] = Vf[j]

            out_dir = os.path.join(base_out, day, str(label), f"{k}pkt")
            os.makedirs(out_dir, exist_ok=True)

            fid_str = "_".join(map(str, flow_id))
            fname   = f"{fid_str}_{k}pkt.npy"
            path    = os.path.join(out_dir, fname)

            np.save(path, img)

def main():
    os.makedirs(IMAGE_ROOT, exist_ok=True)
    summary_paths = sorted(glob.glob(os.path.join(SUMMARY_DIR, f"*{SUMMARY_SUFFIX}")))
    for summary_csv in summary_paths:
        day = os.path.basename(summary_csv).replace(SUMMARY_SUFFIX, '')
        pm = os.path.join(PARSED_DIR, day + META_SUFFIX)
        vp = os.path.join(PARSED_DIR, day + VEC_SUFFIX)
        if os.path.exists(pm) and os.path.exists(vp):
            build_images_for_day(pm, vp, summary_csv, IMAGE_ROOT)
        else:
            tqdm.write(f"[SKIP] missing parsed/vec for {day}")
    print("All days processed!")

if __name__ == "__main__":
    main()

```

2) 출력 결과

```
Build Friday-WorkingHours: 100%| | 294499/294499 [10:41<00:00, 458.74it/s]
Build Monday-WorkingHours: 100%| | 131667/131667 [08:09<00:00, 268.97it/s]
Build Thursday-WorkingHours: 100%| | 159663/159663 [06:57<00:00, 382.60it/s]
Build Tuesday-WorkingHours: 100%| | 108179/108179 [06:43<00:00, 267.82it/s]
Build Wednesday-WorkingHours: 100%| | 134473/134473 [08:15<00:00, 271.48it/s]
All days processed!
```

3) 실행 결과 : SPIN_IDS 하위 폴더에 사진과 같이 저장된다.





저장된 데이터 중 일부만 띄워 캡처하였다.

5. Compute_image_counts.py

1) 4번 코드에 의해 생성된 이미지 중, 각 요일에 대한 라벨별로 패킷 개수에 따른 이미지가 몇 장씩 있는지 출력하여 csv 파일로 저장하는 코드이다.

```
1) import os
2) import glob
3) import pandas as pd
4) from tqdm import tqdm
5)
6) IMAGE_ROOT = "SPIN_IDS/images_per_flow"
7) OUTPUT_CSV = "image_counts_summary.csv"
8)
9) records = []
10)
11)for day in sorted(os.listdir(IMAGE_ROOT)):
12)    day_dir = os.path.join(IMAGE_ROOT, day)
13)    if not os.path.isdir(day_dir):
14)        continue
15)
16)    for label in sorted(os.listdir(day_dir)):
```

```
        label_dir = os.path.join(day_dir, label)
        if not os.path.isdir(label_dir):
            continue

        for pkt_dir in sorted(os.listdir(label_dir)):

            if not pkt_dir.endswith("pkt"):
                continue

            full_pkt_dir = os.path.join(label_dir, pkt_dir)

            npy_files = glob.glob(os.path.join(full_pkt_dir, "*.npy"))
            count = len(npy_files)

            records.append({
                "day": day,
                "label": label,
                "packet_count": pkt_dir.replace("pkt", ""),
                "num_images": count
            })

df = pd.DataFrame(records,
   columns=["day","label","packet_count","num_images"])
print(df.to_string(index=False))

df.to_csv(OUTPUT_CSV, index=False)
print(f"\nSummary written to {OUTPUT_CSV}")
```

2) 출력 결과

상단 데이터 일부를 캡처하였다.

day	label	packet_count	num_images
Friday-WorkingHours	BENIGN	10	78662
Friday-WorkingHours	BENIGN	11	74728
Friday-WorkingHours	BENIGN	12	72419
Friday-WorkingHours	BENIGN	13	71232
Friday-WorkingHours	BENIGN	14	69750
Friday-WorkingHours	BENIGN	15	67502
Friday-WorkingHours	BENIGN	1	89410
Friday-WorkingHours	BENIGN	2	89410
Friday-WorkingHours	BENIGN	3	88418
Friday-WorkingHours	BENIGN	4	88011
Friday-WorkingHours	BENIGN	5	87964
Friday-WorkingHours	BENIGN	6	87803
Friday-WorkingHours	BENIGN	7	81490
Friday-WorkingHours	BENIGN	8	80254
Friday-WorkingHours	BENIGN	9	79283
Friday-WorkingHours	Bot	10	124
Friday-WorkingHours	Bot	11	124
Friday-WorkingHours	Bot	12	54
Friday-WorkingHours	Bot	13	54
Friday-WorkingHours	Bot	14	53
Friday-WorkingHours	Bot	15	53
Friday-WorkingHours	Bot	1	738
Friday-WorkingHours	Bot	2	738
Friday-WorkingHours	Bot	3	738
Friday-WorkingHours	Bot	4	738
Friday-WorkingHours	Bot	5	738
Friday-WorkingHours	Bot	6	738
Friday-WorkingHours	Bot	7	738
Friday-WorkingHours	Bot	8	738
Friday-WorkingHours	Bot	9	738
Friday-WorkingHours	Bot - Attempted	1	490
Friday-WorkingHours	Bot - Attempted	2	490
Friday-WorkingHours	Bot - Attempted	3	490
Friday-WorkingHours	Bot - Attempted	4	490
Friday-WorkingHours	Bot - Attempted	5	490
Friday-WorkingHours	Bot - Attempted	6	490

3) 실행 결과

상단 데이터 일부를 캡처하였다.

image_counts_summary.csv

```
1   day,label,packet_count,num_images
2   Friday-WorkingHours,BENIGN,10,78662
3   Friday-WorkingHours,BENIGN,11,74728
4   Friday-WorkingHours,BENIGN,12,72419
5   Friday-WorkingHours,BENIGN,13,71232
6   Friday-WorkingHours,BENIGN,14,69750
7   Friday-WorkingHours,BENIGN,15,67502
8   Friday-WorkingHours,BENIGN,1,89410
9   Friday-WorkingHours,BENIGN,2,89410
10  Friday-WorkingHours,BENIGN,3,88418
11  Friday-WorkingHours,BENIGN,4,88011
12  Friday-WorkingHours,BENIGN,5,87964
13  Friday-WorkingHours,BENIGN,6,87803
14  Friday-WorkingHours,BENIGN,7,81490
15  Friday-WorkingHours,BENIGN,8,80254
16  Friday-WorkingHours,BENIGN,9,79283
```

6. View_image.py

1) 생성된 이미지 파일을 출력하는 코드이다.

이미지가 정상적으로 생성되었는지 확인하기 위해 실행한다.

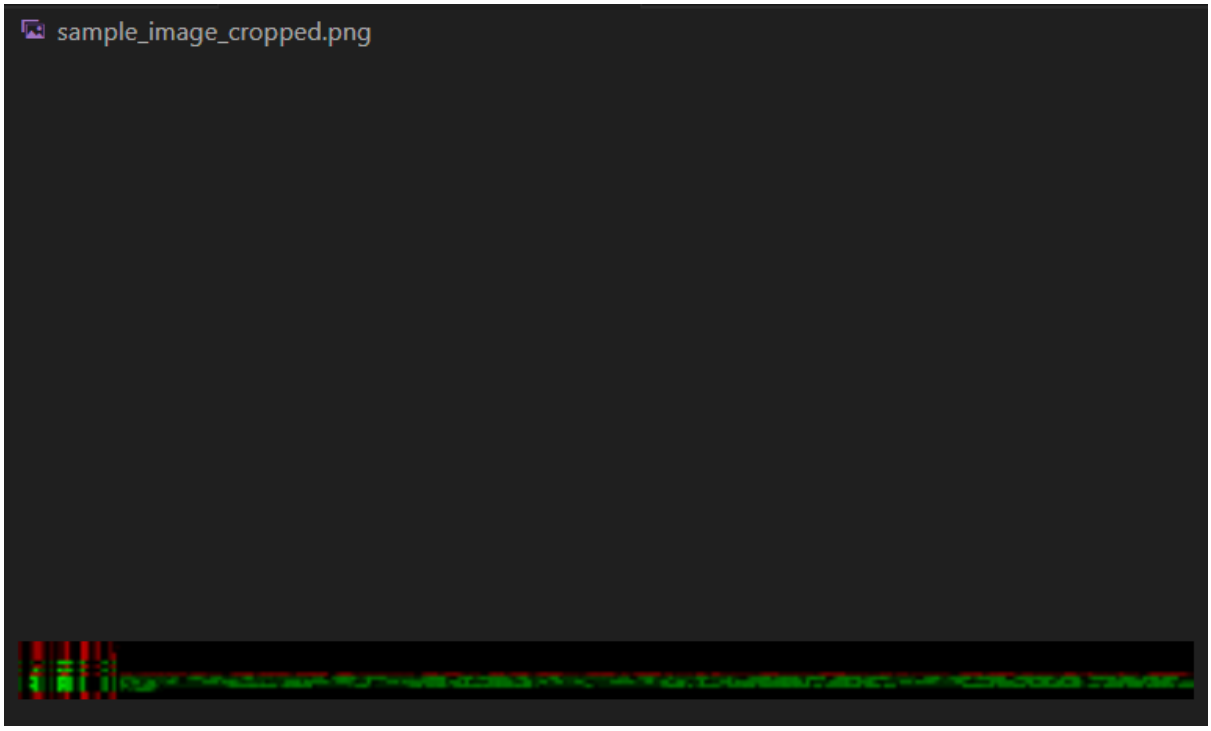
실행 방법 : **5번 라인의 file_path**를 수정하여 별도의 경로 지정을 통해 원하는 이미지를 생성한다.

```
1) import numpy as np
2) import matplotlib.pyplot as plt
3) import os
4)
5) file_path = "SPIN_IDS/images_per_flow/Wednesday-WorkingHours/DoS
   Hulk/15pkt/172.16.0.1_192.168.10.50_32768_80_6_15pkt.npy"
6) save_path = "sample_image_cropped.png"
7)
8) ext = os.path.splitext(file_path)[1].lower()
9) if ext == '.npz':
10)    archive = np.load(file_path)
11)    images = archive['images']
12)elif ext == '.npy':
13)    arr = np.load(file_path)
14)    images = np.expand_dims(arr, 0)
15)else:
16)    raise ValueError(f"Unsupported file type: {ext}")
17)
18)print(f"총 이미지 수: {len(images)}, 이미지 shape: {images[0].shape}")
19)
20)img = images[0]
21)H, W, C = img.shape
22)crop_w = 300
23)img_cropped = img[:, :crop_w, :] if W > crop_w else img
24)
25)plt.imsave(save_path, img_cropped)
26)print(f"이미지 저장 완료: {save_path}")
27)
28)plt.figure(figsize=(10, 2))
29)plt.imshow(img_cropped)
30)plt.title("Flow Image (Cropped to First 300 columns)")
31)plt.axis('off')
32)plt.tight_layout()
33)plt.show()
```

2) 출력 결과

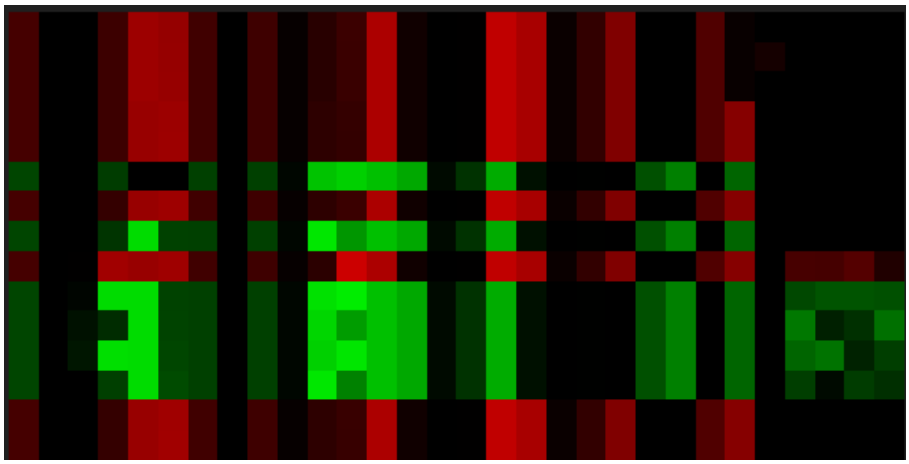
```
총 이미지 수: 1, 이미지 shape: (15, 1486, 3)
이미지 저장 완료: sample_image_cropped.png
```

3) 실행 결과 : sample_image_cropped 이름의 이미지 파일 1개가 저장된다.



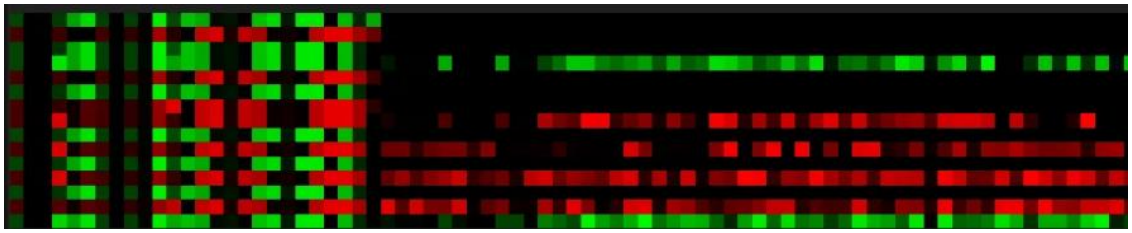
4) 가로 길이가 길어 잘 보이지 않을 경우, **22번 라인의 crop_w** 값을 변경해 간단하게 볼 수 있다.

아래 사진은 가로 길이를 30으로 설정하여 출력한 DoS Hulk의 결과이다.

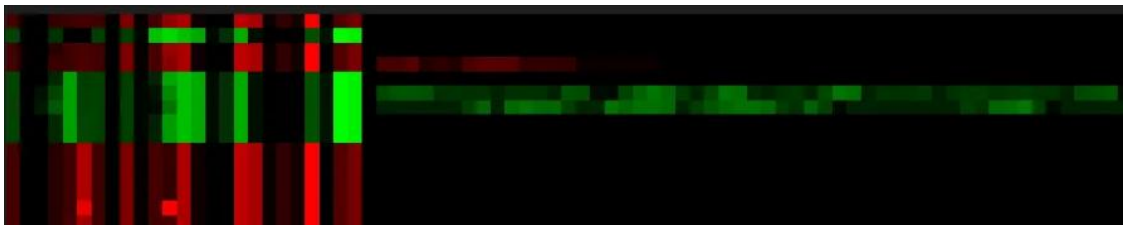


5) 라벨별 이미지 예시 (일부 라벨에 대한 이미지만 추출)

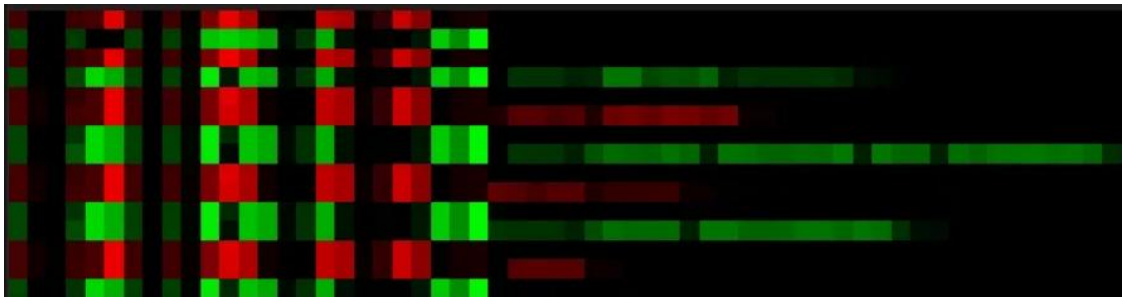
① BENIGN (정상)



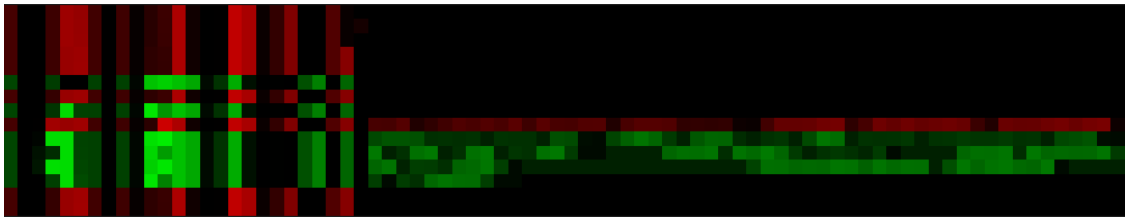
② DDoS



③ FTP-Patator



④ DoS GolenEye



7. Train_val_test.py

1) image_per_flow의 이미지(.npy)를 스캔하여 각 레이블별 flow_id 100개를 샘플링하여 70:15:15 비율로 train:val:test 세트로 분할하는 코드이다.

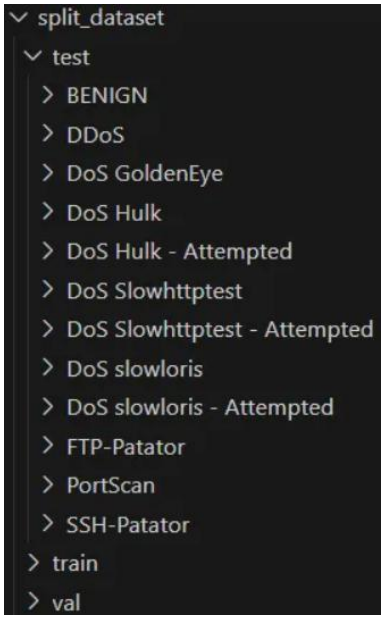
```python
import os
import glob
import random
import shutil

IMAGE_ROOT  = "SPIN_IDS/images_per_flow"
OUTPUT_ROOT = "SPIN_IDS/split_dataset"
P           = 15
FLOWS_PER_LABEL = 100
SPLIT_RATIOS = {"train": 0.70, "val": 0.15, "test": 0.15}

random.seed(42)

def build_file_map(image_root):
    file_map = {}
    for day in os.listdir(image_root):
        day_dir = os.path.join(image_root, day)
        if not os.path.isdir(day_dir): continue
        for label in os.listdir(day_dir):
            lbl_dir = os.path.join(day_dir, label)
            if not os.path.isdir(lbl_dir): continue
            for k_dir in os.listdir(lbl_dir):
                if not k_dir.endswith("pkt"): continue
                try:
                    k = int(k_dir.replace("pkt",""))
                except:
                    continue
                pkt_dir = os.path.join(lbl_dir, k_dir)
                for fn in os.listdir(pkt_dir):
                    if not fn.endswith(".npy"):
                        continue

                    suffix = f"_{k}pkt.npy"
                    if not fn.endswith(suffix):
                        continue
                    flow_id = fn[:-len(suffix)]
                    path = os.path.join(pkt_dir, fn)
                    file_map.setdefault(label, {})\
                            .setdefault(flow_id, {})[k] = path
    return file_map

def sample_and_split_flows(flows, seed=42):
    random.seed(seed)
    sampled = random.sample(flows, FLOWS_PER_LABEL)
    random.shuffle(sampled)
    n = FLOWS_PER_LABEL
    n_train = int(n * SPLIT_RATIOS["train"])
    n_val   = int(n * SPLIT_RATIOS["val"])

    n_test  = n - n_train - n_val
    return {
        "train": sampled[:n_train],
        "val":   sampled[n_train:n_train+n_val],
        "test":  sampled[n_train+n_val:]
    }

def main():
    os.makedirs(OUTPUT_ROOT, exist_ok=True)

    print("Scanning images...")
    file_map = build_file_map(IMAGE_ROOT)
```

```
62.
63.     for label, flows_dict in file_map.items():
64.         flows15 = [fid for fid, km in flows_dict.items() if 15 in km]
65.         if len(flows15) < FLOWS_PER_LABEL:
66.             print(f"[SKIP] '{label}' has only {len(flows15)} flows with 15pkt
(<{FLOWS_PER_LABEL})")
67.             continue
68.
69.         print(f"[SAMPLE] Label={label}: {len(flows15)} → 샘플링 {FLOWS_PER_LABEL}")
70.         splits = sample_and_split_flows(flows15)
71.
72.         for split_name, flow_list in splits.items():
73.             for fid in flow_list:
74.                 for k in range(1, P+1):
75.
76.                     src_path = flows_dict[fid].get(k)
77.                     if not src_path:
78.                         continue
79.                     dst_dir = os.path.join(OUTPUT_ROOT, split_name, label, f"{k}pkt")
80.                     os.makedirs(dst_dir, exist_ok=True)
81.                     dst_path = os.path.join(dst_dir, os.path.basename(src_path))
82.                     shutil.copy(src_path, dst_path)
83.
84.         print(f"[DONE] '{label}' split:  train={len(splits['train'])},
val={len(splits['val'])}, test={len(splits['test'])}")
85.
86.     print("All labels processed.")
87.
88. if __name__ == "__main__":
89.     main()
```

2) 출력 결과

```
Scanning images...
[SKIP] 'Bot - Attempted' has only 0 flows with 15pkt (<100)
[SKIP] 'Bot' has only 53 flows with 15pkt (<100)
[SAMPLE] Label=BENIGN: 401158 → 샘플링 100
[DONE] 'BENIGN' split:  train=70, val=15, test=15
[SAMPLE] Label=DDoS: 20522 → 샘플링 100
[DONE] 'DDoS' split:  train=70, val=15, test=15
[SAMPLE] Label=PortScan: 263 → 샘플링 100
[DONE] 'PortScan' split:  train=70, val=15, test=15
[SKIP] 'FTP-Patator - Attempted' has only 11 flows with 15pkt (<100)
[SKIP] 'SSH-Patator - Attempted' has only 8 flows with 15pkt (<100)
[SAMPLE] Label=SSH-Patator: 2960 → 샘플링 100
[DONE] 'SSH-Patator' split:  train=70, val=15, test=15
[SAMPLE] Label=FTP-Patator: 3972 → 샘플링 100
[DONE] 'FTP-Patator' split:  train=70, val=15, test=15
[SKIP] 'Web Attack - XSS - Attempted' has only 3 flows with 15pkt (<100)
[SKIP] 'Web Attack - XSS' has only 24 flows with 15pkt (<100)
[SKIP] 'Infiltration' has only 4 flows with 15pkt (<100)
[SKIP] 'Web Attack - Brute Force' has only 81 flows with 15pkt (<100)
[SKIP] 'Web Attack - Sql Injection' has only 0 flows with 15pkt (<100)
[SKIP] 'Web Attack - Brute Force - Attempted' has only 6 flows with 15pkt (<100)
[SKIP] 'Infiltration - Attempted' has only 2 flows with 15pkt (<100)
[SKIP] 'Heartbleed' has only 1 flows with 15pkt (<100)
[SAMPLE] Label=DoS Hulk: 14108 → 샘플링 100
[DONE] 'DoS Hulk' split:  train=70, val=15, test=15
[SAMPLE] Label=DoS slowloris: 2235 → 샘플링 100
[DONE] 'DoS slowloris' split:  train=70, val=15, test=15
[SAMPLE] Label=DoS slowloris - Attempted: 1730 → 샘플링 100
[DONE] 'DoS slowloris - Attempted' split:  train=70, val=15, test=15
[SAMPLE] Label=DoS Slowhttptest: 1407 → 샘플링 100
[DONE] 'DoS Slowhttptest' split:  train=70, val=15, test=15
[SAMPLE] Label=DoS Hulk - Attempted: 443 → 샘플링 100
[DONE] 'DoS Hulk - Attempted' split:  train=70, val=15, test=15
[SKIP] 'DoS GoldenEye - Attempted' has only 80 flows with 15pkt (<100)
[SAMPLE] Label=DoS Slowhttptest - Attempted: 3367 → 샘플링 100
[DONE] 'DoS Slowhttptest - Attempted' split:  train=70, val=15, test=15
[SAMPLE] Label=DoS GoldenEye: 7494 → 샘플링 100
[DONE] 'DoS GoldenEye' split:  train=70, val=15, test=15
All labels processed.
```

3) 실행 결과 : SPIN_IDS의 하위 폴더에 사진처럼 저장된다.



8. Bruteforce_train_val_test.py

1) bruteforce의 모든 pkt 당 이미지 개수는 100장이 되지 않아 위 분할 과정에서는 생략되었지만 15pkt의 이미지 개수는 81개로, 학습하는 데에 충분하다고 판단하여 해당 라벨에 대해서도 별도의 분할 과정을 거쳤다.

아래는 bruteforce에 대해서만 분할된 데이터셋을 생성하는 코드이다.

```python
import os, glob, random, shutil, math

IMAGE_ROOT   = "SPIN_IDS/images_per_flow"
OUTPUT_ROOT  = "SPIN_IDS/split_bruteforce"
LABEL        = "Web Attack - Brute Force"
MAX_FLOWS    = 100
P            = 15
RATIO_TRAIN  = 0.70
RATIO_VAL    = 0.15
RATIO_TEST   = 0.15

random.seed(42)

def build_flow_map():
    """
    flow_id → { k: filepath, ... }
    """
    fmap = {}
    pattern = os.path.join(IMAGE_ROOT, "*", LABEL, "*pkt")
    for pkt_dir in glob.glob(pattern):
        k = int(os.path.basename(pkt_dir).replace("pkt", ""))
        for npy in glob.glob(os.path.join(pkt_dir, "*.npy")):
            fid = os.path.basename(npy).rsplit(f"_{k}pkt.npy", 1)[0]
            fmap.setdefault(fid, {})[k] = npy
    return fmap

def select_flows(fmap):
    selected = []
    for k in range(P, 0, -1):

        candidates = [fid for fid, km in fmap.items() if k in km and fid not in selected]
        random.shuffle(candidates)
        need = MAX_FLOWS - len(selected)
        if need <= 0:
            break

        selected += candidates[:need]
    return selected

def split_flows(flows):
    N = len(flows)
    n_train = int(math.floor(N * RATIO_TRAIN + 0.5))
    n_val   = int(math.floor(N * RATIO_VAL   + 0.5))
    n_test  = N - n_train - n_val

    random.shuffle(flows)
    return {
        "train": flows[:n_train],
        "val":   flows[n_train:n_train+n_val],
        "test":  flows[n_train+n_val:]
    }

def main():
    os.makedirs(OUTPUT_ROOT, exist_ok=True)

    fmap     = build_flow_map()
```

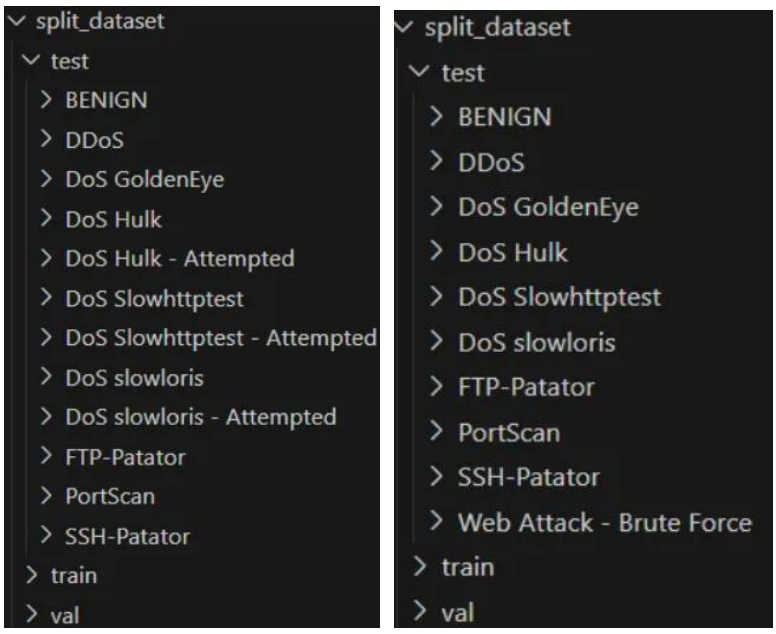
```
    flows    = select_flows(fmap)
    total    = len(flows)
    if total == 0:
        print("[ERROR] 후보 흐름이 하나도 없습니다.")
        return
    print(f"[INFO] 총 확보된 흐름: {total}개 (최대 {MAX_FLOWS}개)")

    splits = split_flows(flows)
    for split, flist in splits.items():
        print(f"  ▶ {split}: {len(flist)} flows")
        for fid in flist:
            for k in range(1, P+1):
                src = fmap[fid].get(k)
                if not src:
                    continue
                dst_dir = os.path.join(OUTPUT_ROOT, split, LABEL, f"{k}pkt")
                os.makedirs(dst_dir, exist_ok=True)
                shutil.copy(src, os.path.join(dst_dir, os.path.basename(src)))

    print("분할 및 복사 완료!")

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

2) 실행 결과 : SPIN_IDS 하위 폴더에 split_bruteforce 폴더가 생성된다.

실행 결과로 나온 데이터를 직접 다른 split_dataset들과 합쳐주어 하나의 폴더에 담기게 한다. (드래그하여 이동시킨다.)

3) 데이터 정리 : 학습에 사용하지 않는 * - Attempted 데이터들은 직접 삭제해주었다. (선택 후 delete)

(왼) – 정리 전(Attempted 데이터들이 섞인 모습) / (오) – 정리 후(학습에 사용할 최종 데이터셋)



9. Cnn_multiclass.py

1) 분할한 학습용 데이터를 이용해 각 라벨의 이미지를 학습시키는 코드이다. 학습 결과도 터미널에 출력되고 이미지 데이터로 저장된다.

가장 좋은 학습 모델도 .pth 파일로 별도 저장된다.

```python
import os, glob, random
import numpy as np
import torch
from torch import nn, optim
from torch.utils.data import Dataset, DataLoader, WeightedRandomSampler
from sklearn.metrics import classification_report, confusion_matrix
from tqdm import tqdm
import matplotlib.pyplot as plt

os.environ["CUDA_VISIBLE_DEVICES"] = "1"
device = torch.device("cuda" if torch.cuda.is_available() else "cpu")
print(f"▶ Using device: {device}")

DATA_ROOT   = "SPIN_IDS/split_dataset"
INPUT_SHAPE = (15, 1486)
BATCH_SIZE  = 128
LR          = 1e-3
EPOCHS      = 100
PATIENCE    = 10

label_list = [
    "BENIGN", "DDoS", "DoS GoldenEye", "DoS Hulk", "DoS Slowhttptest",
    "DoS slowloris", "FTP-Patator", "PortScan", "SSH-Patator",
    "Web Attack - Brute Force"
]
label_map = {lbl:i for i,lbl in enumerate(label_list)}
NUM_CLASSES = len(label_list)

class FlowImageDataset(Dataset):
    def __init__(self, split):
        super().__init__()
        self.samples = []
        split_dir = os.path.join(DATA_ROOT, split)
        for cls in os.listdir(split_dir):
            cls_dir = os.path.join(split_dir, cls)
            if not os.path.isdir(cls_dir): continue
            lbl = label_map.get(cls, None)
            if lbl is None: continue
            for pkt in range(1, 16):
                pkt_dir = os.path.join(cls_dir, f"{pkt}pkt")
                if not os.path.isdir(pkt_dir): continue
                for fn in glob.glob(os.path.join(pkt_dir, "*.npy")):
                    self.samples.append((fn, lbl))

    def __len__(self):
        return len(self.samples)

    def __getitem__(self, idx):
        fn, lbl = self.samples[idx]
        arr = np.load(fn).astype(np.float32) / 255.0
        x   = torch.from_numpy(arr).permute(2,0,1)
        return x, lbl

train_ds = FlowImageDataset("train")
cnts = np.zeros(NUM_CLASSES, dtype=int)
for _,lbl in train_ds.samples:
    cnts[lbl] += 1
total = cnts.sum()
```

```
class_weights = [ total/c for c in cnts ]
print("Class counts:", cnts)
print("Class weights:", class_weights)
weight_tensor = torch.tensor(class_weights, dtype=torch.float32).to(device)

sample_weights = [ class_weights[lbl] for _,lbl in train_ds.samples ]
sampler = WeightedRandomSampler(
    weights=sample_weights,
    num_samples=len(sample_weights),
    replacement=True
)

train_loader = DataLoader(
    train_ds, batch_size=BATCH_SIZE, sampler=sampler,
    num_workers=0, pin_memory=False
)
val_loader = DataLoader(
    FlowImageDataset("val"), batch_size=BATCH_SIZE, shuffle=False,
    num_workers=0, pin_memory=False
)
test_loader = DataLoader(
    FlowImageDataset("test"), batch_size=BATCH_SIZE, shuffle=False,
    num_workers=0, pin_memory=False
)

class FocalLoss(nn.Module):
    def __init__(self, gamma=2.0, weight=None, reduction='mean'):
        super().__init__()
        self.gamma    = gamma
        self.reduction= reduction
        if weight is not None:
            self.register_buffer('weight', weight)
        else:
            self.weight = None

    def forward(self, logits, targets):
        ce = nn.functional.cross_entropy(
            logits, targets,
            weight=self.weight,
            reduction='none'
        )
        pt = torch.exp(-ce)
        loss = (1 - pt)**self.gamma * ce
        return loss.mean() if self.reduction=='mean' else loss.sum()

class IntrusionCNN(nn.Module):
    def __init__(self, num_classes):
        super().__init__()
        self.features = nn.Sequential(
            nn.Conv2d(3,   96, 5, padding=2), nn.ReLU(inplace=True),
            nn.MaxPool2d((1,2), padding=(0,1)), nn.BatchNorm2d(96), nn.Dropout(0.2),

            nn.Conv2d(96, 128, 5, padding=2), nn.ReLU(inplace=True),
            nn.MaxPool2d((1,2), padding=(0,1)),

            nn.Conv2d(128,192, 3, padding=1), nn.ReLU(inplace=True),
            nn.MaxPool2d((1,2), padding=(0,1)),

            nn.Conv2d(192,384, 4, padding=1), nn.ReLU(inplace=True),
            nn.MaxPool2d((1,2), padding=(0,1)),
        )
        with torch.no_grad():
            dummy = torch.zeros(1,3,*INPUT_SHAPE)
            feat  = self.features(dummy)
            flat_dim = feat.view(1,-1).size(1)

        self.classifier = nn.Sequential(
            nn.Flatten(),
```

```
            nn.Dropout(0.2),
            nn.Linear(flat_dim, 64), nn.ReLU(inplace=True),
            nn.Linear(64, num_classes)
        )

        for m in self.modules():
            if isinstance(m, (nn.Conv2d, nn.Linear)):
                nn.init.xavier_normal_(m.weight)
                if m.bias is not None:
                    nn.init.zeros_(m.bias)

    def forward(self, x):
        x = self.features(x)
        return self.classifier(x)

model = IntrusionCNN(NUM_CLASSES).to(device)
print("▶ Total params:", sum(p.numel() for p in model.parameters()))

criterion = FocalLoss(gamma=2.0, weight=weight_tensor)
optimizer = optim.RMSprop(model.parameters(), lr=LR, weight_decay=1e-4)
scheduler = optim.lr_scheduler.ReduceLROnPlateau(
    optimizer, mode='max', factor=0.5, patience=3
)

best_val_acc = 0.0
no_improve   = 0

train_losses, train_accs = [], []
val_losses,   val_accs   = [], []

def run_epoch(loader, train=True):
    if train:
        model.train()
    else:
        model.eval()

    total_loss, correct = 0.0, 0
    loop = tqdm(loader, desc="Train" if train else " Eval")
    with torch.set_grad_enabled(train):
        for X,y in loop:
            X, y = X.to(device), y.to(device)
            logits = model(X)
            loss   = criterion(logits, y)
            if train:
                optimizer.zero_grad()
                loss.backward()
                optimizer.step()
            total_loss += loss.item()*X.size(0)
            correct    += (logits.argmax(1)==y).sum().item()
    return total_loss/len(loader.dataset), correct/len(loader.dataset)

for ep in range(1, EPOCHS+1):
    tr_loss, tr_acc = run_epoch(train_loader, train=True)
    vl_loss, vl_acc = run_epoch(val_loader,   train=False)

    train_losses.append(tr_loss); train_accs.append(tr_acc)
    val_losses.append(vl_loss);   val_accs.append(vl_acc)

    scheduler.step(vl_acc)
    print(f"Ep{ep:02d}: train_loss={tr_loss:.4f} acc={tr_acc:.4f} | val_loss={vl_loss:.4f}
acc={vl_acc:.4f}")

    if vl_acc > best_val_acc + 1e-4:
        best_val_acc = vl_acc
        no_improve   = 0
        torch.save(model.state_dict(), "best_model.pth")
        print(" → New best, model saved")
    else:
```

```
        no_improve += 1
        if no_improve >= PATIENCE:
            print(f"!! EarlyStopping after {ep} epochs")
            break

model.load_state_dict(torch.load("best_model.pth"))
te_loss, te_acc = run_epoch(test_loader, train=False)
print(f"▶ Test loss={te_loss:.4f}, acc={te_acc:.4f}\n")

y_true, y_pred = [], []
model.eval()
with torch.no_grad():
    for X,y in test_loader:
        X = X.to(device)
        logits = model(X)
        y_pred.extend(logits.argmax(1).cpu().tolist())
        y_true.extend(y.tolist())

print("Classification Report:\n")
print(classification_report(y_true, y_pred, target_names=label_list, digits=4))

cm = confusion_matrix(y_true, y_pred)
fig,ax = plt.subplots(figsize=(10,8))
im = ax.imshow(cm, cmap="Blues")
fig.colorbar(im, ax=ax)
ax.set(
    xticks=np.arange(NUM_CLASSES), yticks=np.arange(NUM_CLASSES),
    xticklabels=label_list, yticklabels=label_list,
    xlabel="Predicted", ylabel="Actual", title="Confusion Matrix"
)
plt.setp(ax.get_xticklabels(), rotation=45, ha="right")
thresh = cm.max()/2.
for i in range(NUM_CLASSES):
    for j in range(NUM_CLASSES):
        ax.text(j,i,f"{cm[i,j]:d}",
                ha="center", va="center",
                color="white" if cm[i,j]>thresh else "black")
plt.tight_layout()
plt.savefig("confusion_matrix.png")
print("▶ Confusion matrix saved as 'confusion_matrix.png'")

epochs = np.arange(1, len(train_losses)+1)
plt.figure(figsize=(8,4))
plt.plot(epochs, train_losses, label="Train Loss")
plt.plot(epochs,   val_losses, label="Val Loss")
plt.xlabel("Epoch"); plt.ylabel("Loss"); plt.legend(); plt.title("Loss Curve")
plt.tight_layout(); plt.savefig("loss_curve.png")

plt.figure(figsize=(8,4))
plt.plot(epochs, train_accs, label="Train Acc")
plt.plot(epochs,   val_accs, label="Val Acc")
plt.xlabel("Epoch"); plt.ylabel("Accuracy"); plt.legend(); plt.title("Accuracy Curve")
plt.tight_layout(); plt.savefig("acc_curve.png")

print("▶ Training curves saved: loss_curve.png, acc_curve.png")
```

GPU 사용 설정 후 데이터셋을 이용하여 학습하도록 하였다.

아래는 코드의 중요 부분 설명이다.

- Focal Loss 함수를 이용해 잘못 분류되는 샘플에 더 집중되도록 설정하였고, 4개의 convolution + ReLU + pooling 블록으로 CNN 모델을 정의하였다.

- run_epoch 함수로 한 epoch의 학습/검증을 처리하며 ReduceLROnPlateau 스케줄러로

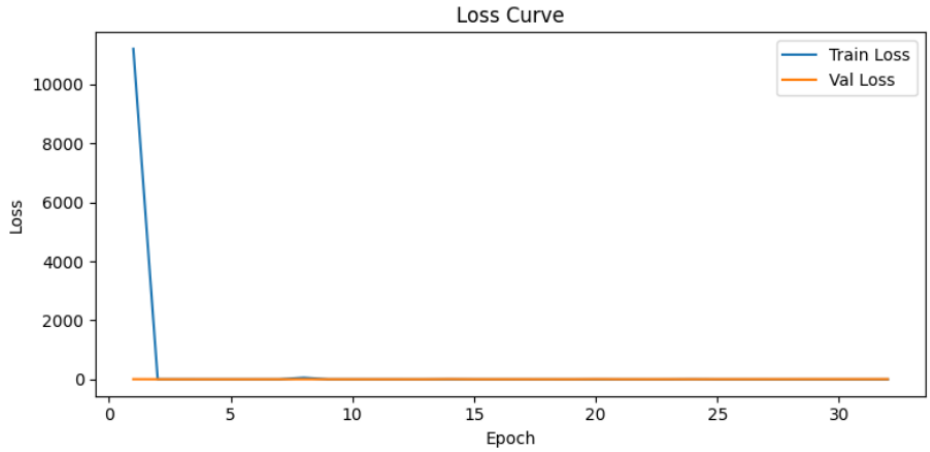
검증 정확도 향상이 멈추면 학습률이 반감되게 하였고, epoch 연속 개선이 없으면 early stopping 하도록 설정했다. (코드에선 10으로 설정)

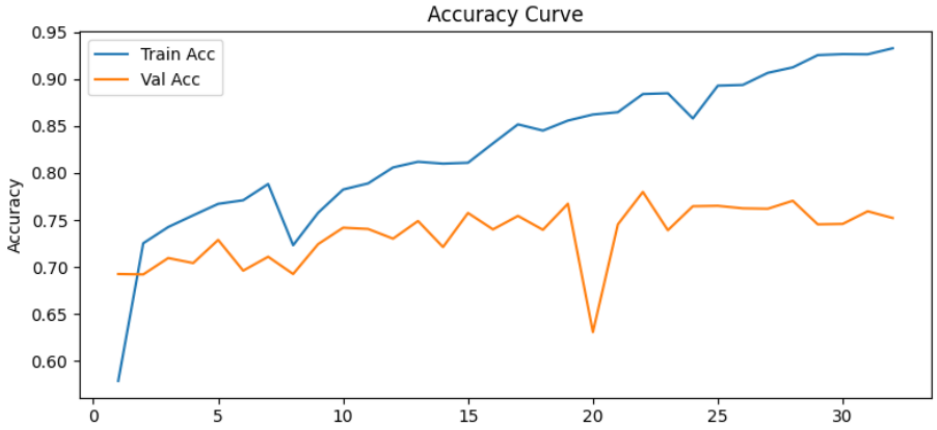
- 테스트는 저장된 최적의 모델을 불러와 test 손실/정확도를 계산하고 classification_report와 confusion matrix를 저장하게 하였다.

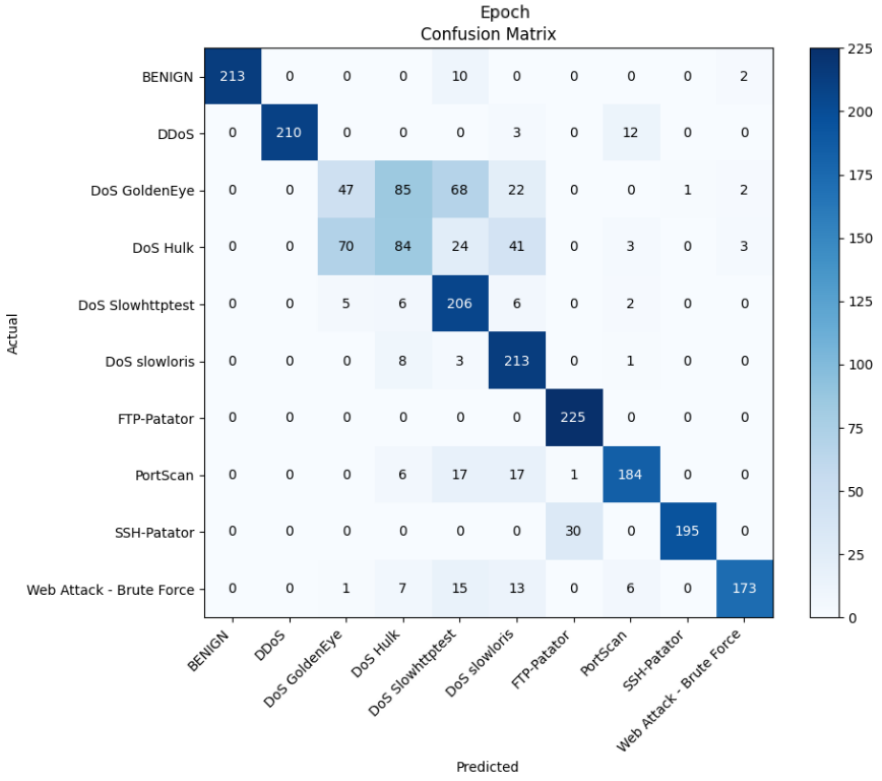
2) 출력 결과

```
▶Test loss=8.0641, acc=0.7812

Classification Report:

                        precision    recall  f1-score   support

                BENIGN     1.0000    0.9467    0.9726       225
                 DDoS      1.0000    0.9333    0.9655       225
        DoS GoldenEye      0.3821    0.2089    0.2701       225
             DoS Hulk      0.4286    0.3733    0.3990       225
       DoS Slowhttptest    0.6006    0.9156    0.7254       225
          DoS slowloris    0.6762    0.9467    0.7889       225
            FTP-Patator    0.8789    1.0000    0.9356       225
               PortScan    0.8846    0.8178    0.8499       225
            SSH-Patator    0.9949    0.8667    0.9264       225
Web Attack - Brute Force    0.9611    0.8047    0.8759       215

               accuracy                         0.7812      2240
              macro avg    0.7807    0.7814    0.7709      2240
           weighted avg    0.7799    0.7812    0.7705      2240

▶Confusion matrix saved as 'confusion_matrix.png'
▶Training curves saved: loss_curve.png, acc_curve.png
```

3) 실행 결과

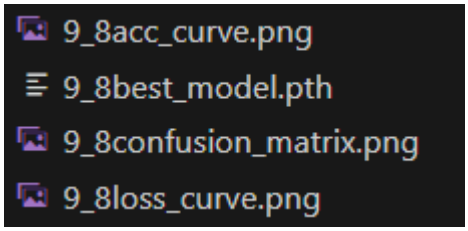






4) 디렉토리에 저장되는 결과는 아래와 같다.

*9_8 이름은 임의로 지정하였다.



IV. 결과 분석

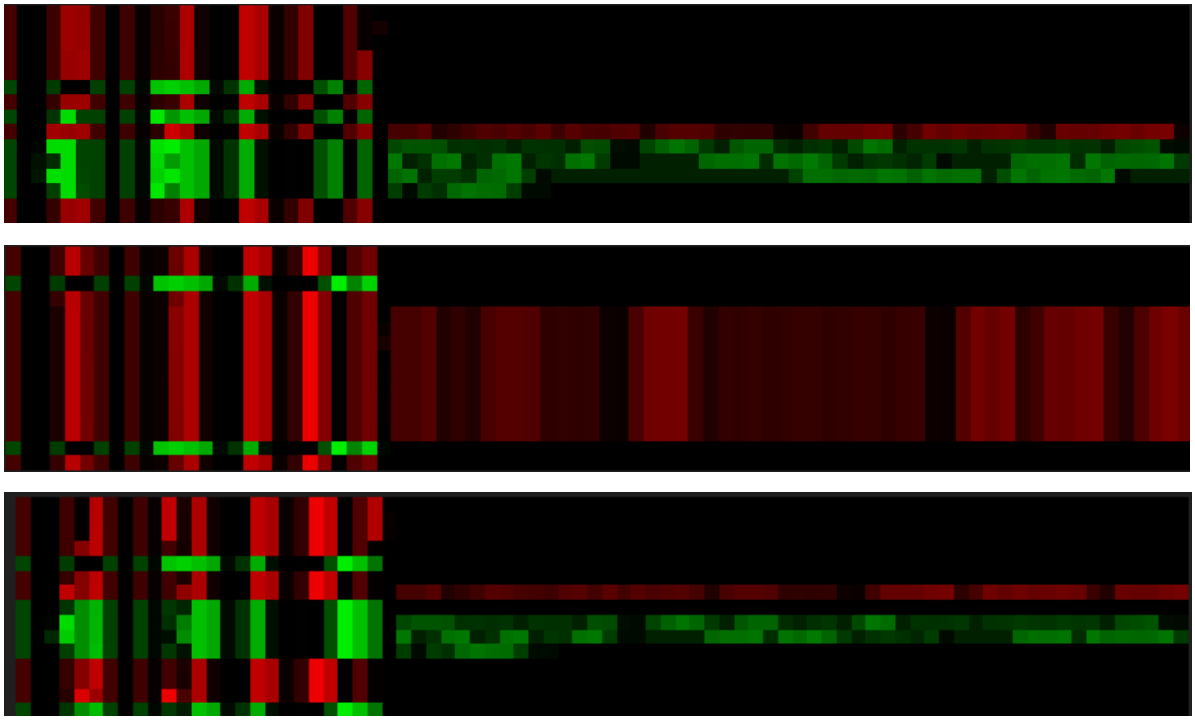
Multiclass 모델을 통해 학습한 결과는 DoS 공격들을 헷갈리는 양상이 심했다.

DoS 공격을 제외한 다른 공격들은 대체로 높은 정확도로 결과가 나왔다.

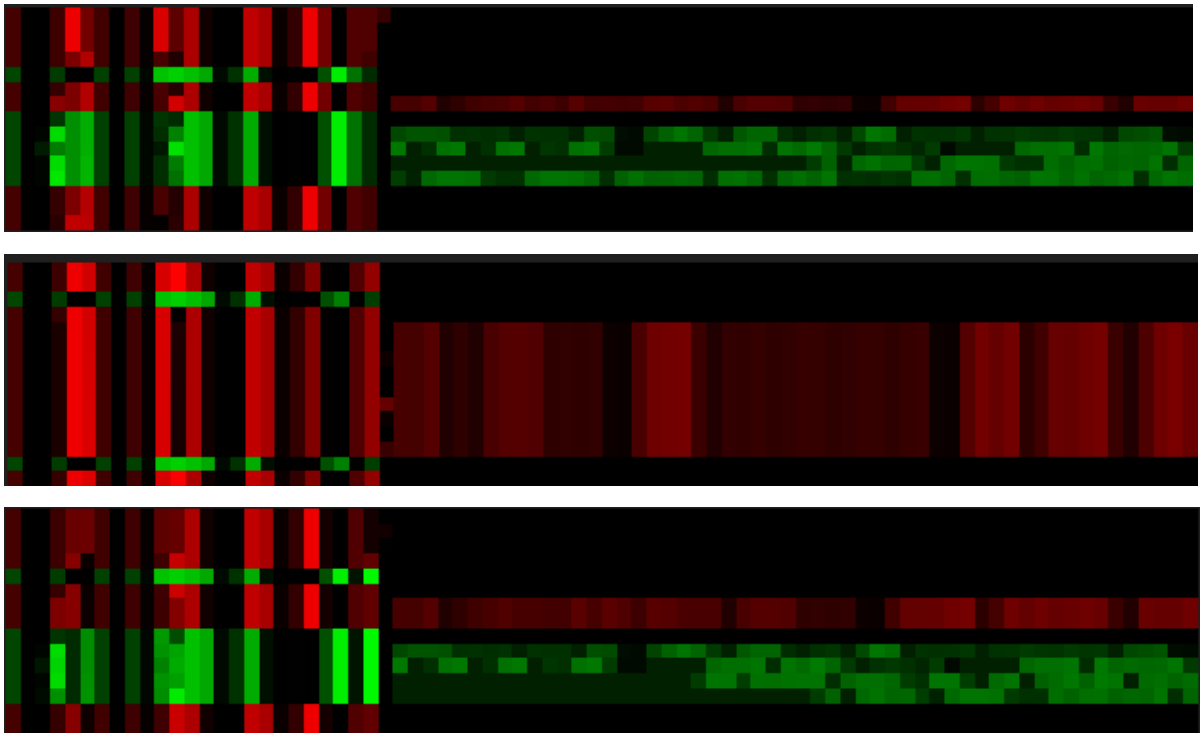
DoS 공격의 이미지를 몇 개 꺼내보았을 때, 비슷하게 생긴 이미지들이 많았다.

아래는 특히 가장 혼잡도가 강했던 DoS Hulk와 DoS GoldenEye의 이미지 예시이다.

DoS Hulk



DoS GoldenEye



같은 데이터셋으로 논문 방식의 Binaryclass 모델로도 한 번 학습시켜보았을 땐, 정상/악성 탐지에서 높은 정확도의 결과가 나왔다.

```
▶ Test loss=0.0317, acc=0.9996
```

이는 정상/공격 두 가지 측면에서 공격을 탐지하는 것은 적합하지만, 각각의 라벨들에 대해 어떤 라벨의 공격인지 판단하는 것에는 아직 한계점이 있다는 것을 알 수 있다.