Computer Security

Intrusion Detection System

김진일 채정영 황우삼

Abstract

본 연구는 Intrusion Detection System(IDS)에 대한 개념 이해 및 분석을 목적으로 하며, IDS 의 장단점, 특징 및 효과 등을 분석하고 실제 IDS를 구현한 어플리케이션을 예시로 다루며 이해를 돕고 있는 상황이다. 기존에 존재하는 애플리케이션 (Tripwire, Suricata)의 자세한 동작 설명을 통해서 해당 IDS와 툴에 대한 이해도를 높이고 IDS가 일상속에서 어떻게 사용되는 지에 대해 연구한다. 하지만 기존에 존재하는 어플리케이션은 오탐율이 중요한 이슈로 남고있다. 본 연구는 이러한 상황에서 IDS의 단점인 패킷의 오탐율 높은 것과 패턴의 변화로 인한 탐지 어려움 등을 해결하기 위해 머신러닝을 이용한 IDS모델을 만들어 보았으며, 이러한 모델들을 통해서 비슷한 Feature를 가진 패킷들을 그룹화를 통해 좀 더 유연한 패킷 탐지를 구현하였다. 이러한 IDS모델을 만들기 위해 Feature 정의 및 train을 통하여 다양한 패킷들 속에서 더 정확한 탐지를 할 수 있는지에 대한 여부를 서버 테스트를 통해서 확인해보았으며, 실제로 Slowloris DoS공격을 감지하는 시뮬레이션을 실행하여 확인해 보았다. DoS 뿐만 아니라 모델을 훈련만 시킨다면 무슨 공격이든 학습된 모델이 이를 감지하고 사용자에게 알릴 수 있을 것이다. 이렇게 만든 IDS모델을 IPS 와 같이 사용을 한다면 더욱 안전을 보장 할 수 있을 것이라고 예측된다.

CONTENTS

Abstract ……..…….. 2
Contents ……..…….. 3
I. Instruction ……..…….. 3
II. Problem Definition ……..…….. 4
1. 기존 IDS 모델 ……..……..…….. 4
1.1 Tripwire ……..……,..... 4
1.2 Suricata ……..……. 6
III. Solutions ……..……... 7
1. 실험설계……..……..…… 7
1.1 시나리오 ……..……..…… 7
1.2 서버 설계 ……..……..…… 7
1.2.1 코드 ……..……..…… 7
1.3 공격 설계 ……..……..……. 9
1.3.1 공격 결과 ……..……..……. 9
1.4 모델 설계……..……..……... 9
1.4.1 cicflowmeter ……..……... 10
1.4.1.1 구조……..……. 10
1.4.1.2 패킷 스니퍼 (Packet Sniffer)……..……..…... 10
1.4.1.3 특징 추출 (Feature Extraction) ……..……..…... 11
1.4.2. 머신러닝을 사용한 IDS (IDS using Machine Learning) …… 11
1.4.2.1 IDS 데이터 셋 (IDS Dataset) …… 11
1.4.2.2. CSV 파일 읽기 및 전처리 (Preprocessing) …… 13
1.4.2.3. 모델 구성 (Model Composition) ……... 14
1.4.2.4. 모델 훈련 및 성능 테스트 (Model Training).....……. 15
1.4.2.5. cicflowmeter에 모델 적용 (Combination of cicflowmeter and IDS model) 16
1.4.2.6 결과 ……..…… 17
IV. Conclusion ……..……..…….. 17
V. Reference ……..……..……..…… 18

I. Introduction

IDS란 시스템이나 네트워크의 비정상적인 변화 혹은 행동을 탐지하는 시스템으로 동작중에 의심스러운 행동이 감지가 될 경우에 로그를 남기거나 해당 관리자에게 경고를 보낸다. IDS의 대표적인 특징으로는 내부 혹은 네트워크의 접속점에 위치하며 방화벽의 부족한 부분을 보강하기 위해 사용되며 의심스러운 행동이 발견 되거나 시스템의 침해여부를 판단하기 위해 사용된다.

IDS는 크게 호스트 기반(HIDS)와 네트워크 기반(NIDS)로 나뉜다. 우선 호스트 기반은 시스템 로그, 네트워크 연결 또는 하드디스크를 검사하여 내부에서 문제가 발생했을 경우에 사용자에게 경고를 한다. 즉 실행 중인 프로세스 관찰이나 시스템 로그 분석 후 감지하는 일을 한다. 또한 서버에 직접 설치하므로 네트워크 환경과 무관하고 호스트 시스템으로 부터 생성되고 수집되는 감사 자료를 침입 탐지에 사용하므로 정확한 탐지 및 침입 방지를 할 수 있고 추가적인 하드웨어가 필요하지 않는다. 하지만 각각의 시스템마다 설치가 필요하며 호스트에 직접 설치하므로 호스트 성능에 의존적이고 리소스 사용으로 인한 서버 부하가 발생하는 단점이 있다.

최근 통신의 발달로 인해서 다양한 장소에서 네트워크를 통하여 많은 일들을 할 수 있다. 그렇기 때문에 네트워크에 대한 보안 또한 아주 중요한 요소로 떠오르고 있다. 네트워크의 공격으로는 분산 서비스 거부 공격(Distributed Denial Of Service)등 이 있다.

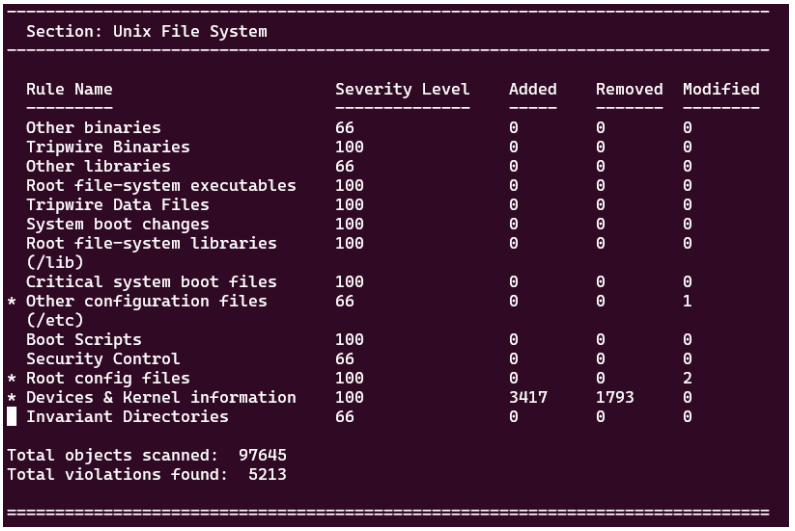
이러한 위험 요소를 초기에 발견 및 해결하고자 하는 방법으로 네트워크 단계에서 미리 공격을 탐지하는 Intrusion Detection System(IDS)에 대해 관심을 가지고 개념 이해 및 분석을 통해 단점들을 보완한 머신러닝을 통한 IDS model을 만들어 보았으며 이러한 실험을 통해서 IDS에 대한 관심을 가질 수 있는 계기를 제공하며 네트워크를 보안하는 한가지의 방법으로서 IDS를 소개한다.

II. Problem Definition

1. 기존 IDS 모델

1.1 Tripwire

호스트 기반의 기능을하는 어플리케이션으로는 Tripwire이 있다. 이것은 서버내의 파일의 무결성 검사를 통해서 파일의 변화를 감지하여 침입의 여부를 판단한다. 무결성 검사 명령은 tripwire -check이며, 변화 상태 업데이트 명령은 tripwire --update 이다.



무결성 검사:$./tripwire -check
변화상태 업데이트: $./tripwire --update

그림 1

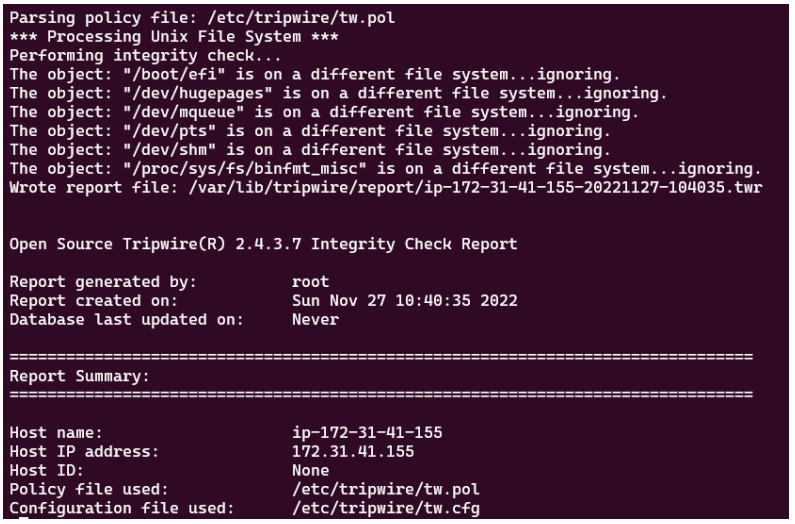


그림 2

그림 1의 명령어를 통해 Tripwire를 사용할 수 있고 사용자는 서버내의 무결성 검사를 하여 파일의 변화를 감지 및 침입여부를 판단한다(그림 2). 만약 서버내의 수정한 사항이 있다면 변화상태를 업데이트 하여 해당 데이터 베이스를 업데이트 한다.

위의 두 사진은 Tripwire를 사용하여 서버내의 무결성 검사를 한 결과로 검사 결과로는 호스트 이름과 아이피 등을 확인할 수 있고 또한 보안 단계에 따라서 파일의 추가, 삭제, 변경등을 요약하여 보여주므로 서버내 파일 변화 감지에 대해

알아보기가 쉽다.

반면, 네트워크 기반은 네트워크를 지나가는 패킷들을 분석하여 의심스러운 패킷들을 경고하는데 이러한 기능은 네트워크를 통한 침투를 감지하게 된다. 네트워크를 통해서 전송되는 정보를 분석하여서 침입 여부를 판단한다. 이를 수행하기 위해서 네트워크 세그먼트 당 하나의 감지기를 설치해야 하지만 설치가 용이하다. 운영체제에 독립적으로 구현 및 관리가 쉽고 네트워크에서 발생하는 여러 유형의 침입을 탐지할 수 있다.

하지만 암호화된 패킷을 분석할 수 없으며 호스트 상에서 수행되는 세부 행위에 대해서 탐지할 수 없다. 또한 오탐율이 높다는 단점을 가지고 있다.

1.2 Suricata

```
tail -f /var/log/suricata/fast.log
```

```
root@ip-172-31-41-155:/home/woosam# tail -f /var/log/suricata/fast.log
12/03/2022-11:56:29.293241  [**] [1:1000004:1] GET [**] [Classification: (null)] [Priority: 3] {TCP} 49.143.
41.161:9758 -> 172.31.41.155:5000
12/03/2022-11:56:29.304236  [**] [1:1000004:1] GET [**] [Classification: (null)] [Priority: 3] {TCP} 49.143.
41.161:9760 -> 172.31.41.155:5000
12/03/2022-11:56:29.319901  [**] [1:1000004:1] GET [**] [Classification: (null)] [Priority: 3] {TCP} 49.143.
41.161:9699 -> 172.31.41.155:5000
12/03/2022-11:56:30.843674  [**] [1:1000005:1] Connect(TCP) to Server [**] [Classification: (null)] [Priorit
y: 3] {TCP} 49.143.41.161:2750 -> 172.31.41.155:22
12/03/2022-11:57:40.225355  [**] [1:1000005:1] Connect(TCP) to Server [**] [Classification: (null)] [Priorit
y: 3] {TCP} 175.212.59.183:61306 -> 172.31.41.155:22
12/03/2022-11:57:40.235895  [**] [1:1000005:1] Connect(TCP) to Server [**] [Classification: (null)] [Priorit
y: 3] {TCP} 218.209.92.114:59138 -> 172.31.41.155:22
12/03/2022-11:59:58.911543  [**] [1:1000005:1] Connect(TCP) to Server [**] [Classification: (null)] [Priorit
y: 3] {TCP} 85.17.24.112:46578 -> 172.31.41.155:22
12/03/2022-11:59:58.926227  [**] [1:1000005:1] Connect(TCP) to Server [**] [Classification: (null)] [Priorit
y: 3] {TCP} 218.209.92.114:59138 -> 172.31.41.155:22
12/03/2022-12:00:04.523109  [**] [1:1000005:1] Connect(TCP) to Server [**] [Classification: (null)] [Priorit
y: 3] {TCP} 45.32.17.205:33797 -> 172.31.41.155:443
12/03/2022-12:00:34.363444  [**] [1:1000005:1] Connect(TCP) to Server [**] [Classification: (null)] [Priorit
y: 3] {TCP} 218.209.92.114:59112 -> 172.31.41.155:22
```

그림 3

해당 사진은 서버내로 들어오는 패킷들 중 TCP 연결로 들어오는 패킷들을 검출하는 것으로 위의 명령어를 통해서 실시간으로 확인 할 수 있는 것을 볼 수있다.

해당 패킷을 검출하기 위한 Rule은 Snort Rule 구조를 따르고 있는데 아래의 사진에서 확인할 수 있다. Snort rule이란 IP 네트워크에서 프로토콜 분석, 콘텐츠 검색/비교 작업등 실시간 트래픽 분석과 패킷 로깅 작업을 수행하여 다양한 공격과 스캔을 탐지하는 방법이다.

Snort Rule 구조

룰 헤더							옵션
Action	Protocol	SrcIP	SrcPort	->	DstIP	DstPort	Option

그림 4

Action은 패킷을 탐지했을 경우 동작을 설정하는 방법으로 alert, log, pass 등이 있다. alert는 경고를 발생시키고 탐지 정보를 로그 파일에 기록하며, log 옵션은 패킷에 대해서 로그를 기록한다. pass 옵션은 패킷을 무시한다. 프로토콜은 TCP, UDP, ICMP, IP설정 시 해당 프로토콜에 관해서 탐지 한다. option에는 탐지 조건이 나오는데 msg 는 alert가 발생하면 msg가 설정된 문장을 로그파일에 기록한다.

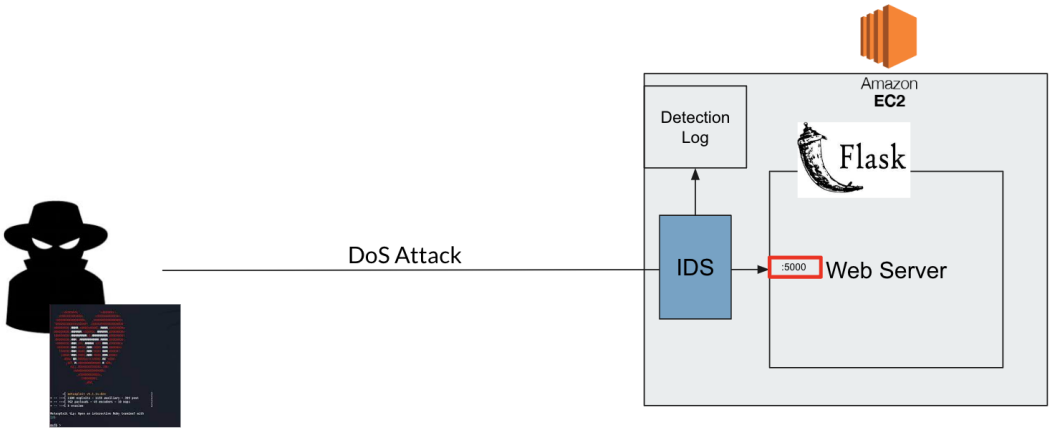
이러한 Snort Rule을 통해서 다양한 패킷을 탐지할 수 있으며 IDS를 구현할 수 있다.

III. Solutions

1. 실험설계

1.1 시나리오

Web Server 를 돌리고 있는 EC2 Instance에 Kali Linux 의 metasploit을 이용하여 Slowloris DoS 공격을 진행한다. 이를 새로운 IDS 모델이 감지하여 사용자에게 알려주어야 한다.



공격 시나리오, 그림 5

1.2 서버 설계

Amazon Web Service 에서 Cloud Computer instance인 EC2인스턴스를 서비스 받아 그 서비스에 Flask 를 이용하여 Web server를 만들었다. 서버 포트는 5000번을 사용하였고 IP:5000 을 사용하여 외부에서 웹서비스를 사용 가능하도록 하였다.

1.2.1 코드

github 주소 : https://github.com/qpalzmm22/ids-network

```python
"""Flask Login Example and instagram fallowing find"""

from flask import Flask, url_for, render_template, request, redirect, session
from flask_sqlalchemy import SQLAlchemy
from instagram import getfollowedby, getname
import sqlite3

conn = sqlite3.connect("login.db")

cur = conn.cursor()
conn.execute('CREATE TABLE IF NOT EXISTS users(id TEXT, password TEXT)')

app = Flask(__name__)
app.config['SQLALCHEMY_DATABASE_URI'] = 'sqlite:///test.db'
db = SQLAlchemy(app)


class User(db.Model):
    """ Create user table"""
    id = db.Column(db.Integer, primary_key=True)
    username = db.Column(db.String(80), unique=True)
    password = db.Column(db.String(80))

    def __init__(self, username, password):
        self.username = username
        self.password = password


@app.route('/', methods=['GET', 'POST'])
def home():
    """ Session control"""
    if not session.get('logged_in'):
        return render_template('index.html')
    else:
        if request.method == 'POST':
            username = getname(request.form['username'])
            return render_template('index.html', data=getfollowedby(username))
        return render_template('index.html')


@app.route('/login', methods=['GET', 'POST'])
def login():
    """Login Form"""
    if request.method == 'GET':
        return render_template('login.html')
    else:
        name = request.form['username']
        passw = request.form['password']
        try:
```

```
50              data = User.query.filter_by(username=name, password=passw).first()
51              if data is not None:
52                  session['logged_in'] = True
53                  return "Login Success" #redirect(url_for('home'))
54              else:
55                  return 'Dont Login'
56          except:
57              return "Dont Login"
58
59  @app.route('/register/', methods=['GET', 'POST'])
60  def register():
61      """Register Form"""
62      if request.method == 'POST':
63          new_user = User(username=request.form['username'], password=request.form['password'])
64          new_user
65          db.session.add(new_user)
66          db.session.commit()
67          return render_template('login.html')
68      return render_template('register.html')
69
70  @app.route("/logout")
71  def logout():
72      """Logout Form"""
73      session['logged_in'] = False
74      return redirect(url_for('home'))
75
76
77  if __name__ == '__main__':
78      app.debug = True
79      with app.app_context() :
80          db.create_all()
81      app.secret_key = "123"
82      app.run(host='0.0.0.0')
```

1.3 공격 설계

공격은 Windows10 환경에서 WSL2 을 사용하여 구동시킨 Kali Linux 에서 진행하였다. metasploit을 사용하여 Slowloris DoS 공격을 진행하였습니다. 150 sockets 을 이용하여 15초의 텀마다 open header http request 를 보내서 서버에서 리소스를 잡아먹도록 하였습니다.

1.3.1 공격 결과

```
msf6 auxiliary(dos/http/slowloris) > set rport 5000
rport => 5000
msf6 auxiliary(dos/http/slowloris) > exploit

[*] Starting server...
[*] Attacking 15.164.165.155 with 150 sockets
[*] Creating sockets...
[*] Sending keep-alive headers... Socket count: 0
[*] Sending keep-alive headers... Socket count: 0
```

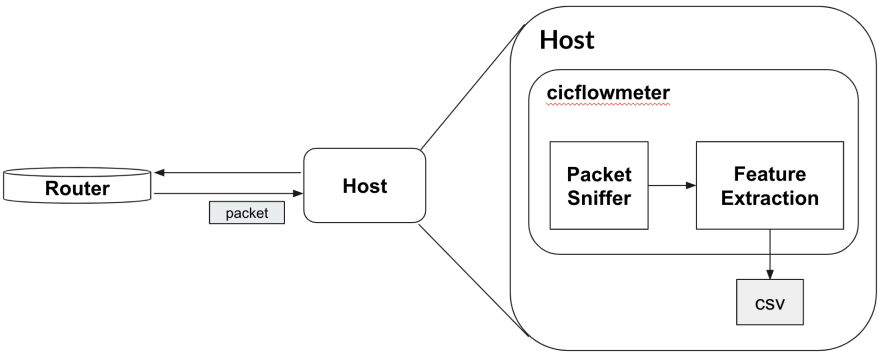
공격 결과, 그림 6

1.4 모델 설계

1.4.1 cicflowmeter

1.4.1.1 구조

cicflowmeter 구조는 Packet Sniffer와 Feature Extraction으로 구성되어 있다. Packet Sniffer가 Host와 연결된 Router에 패킷을 읽어 들이는 역할을 한다. 패킷 정보를 읽은 후, Feature Extraction 과정을 거쳐 CSV 파일에 Feature 데이터를 저장하게 된다.



cicflowmeter 구조, 그림 7

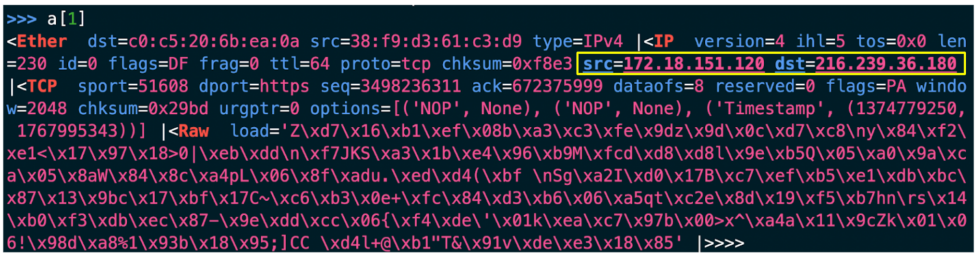
1.4.1.2. 패킷 스니퍼 (Packet Sniffer)

패킷 스니퍼는 Scapy 파이썬 모듈에서 스니핑 기능을 제공하고 있다. 파이썬에서 sniff 함수에 Sniffing을 수행할 라우터 이름을 파라미터로 넘기면, 아래와 같이 패킷을 계속해서 수집하게 되는 것을 볼 수 있다.

```
>>> a = sniff(session=IPSession, iface="en0")
^C>>> a.nsummary()
0000 Ether / IP / TCP 172.18.151.120:51608 > 216.239.36.180:https A / Raw
0001 Ether / IP / TCP 172.18.151.120:51608 > 216.239.36.180:https PA / Raw
0002 Ether / IP / TCP 172.18.151.120:51608 > 216.239.36.180:https PA / Raw
0003 Ether / IP / TCP 216.239.36.180:https > 172.18.151.120:51608 A
0004 Ether / IP / TCP 216.239.36.180:https > 172.18.151.120:51608 A
```

Packet sniffer 결과, 그림 8

위 과정을 통해, 수집한 패킷들 중 첫 번째 패킷의 정보를 살펴보면, 아래와 같이 패킷 데이터가 Ether, IP Header, TCP Header, Raw 형태로 구성되어 있다.



Packet content, 그림 9

1.4.1.3. 특징 추출 (Feature Extraction)

특징 추출은 패킷 정보로부터 정보들을 수집하여 특징(Feature)을 추출하는 작업을 한다. 특징 추출이 수행되고 나면, CSV 파일에 데이터가 쓰이게 된다.

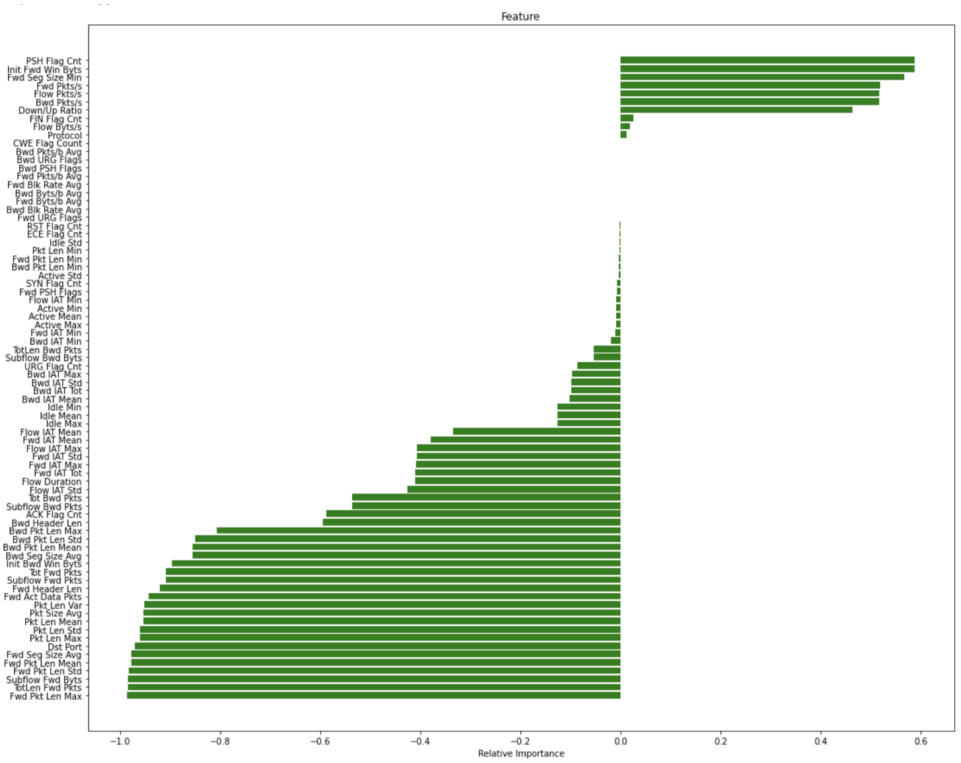
1.4.2. 머신러닝을 사용한 IDS (IDS using Machine Learning)

1.4.2.1. IDS 데이터 셋 (IDS Dataset)

IDS 데이터 셋은 Kaggle - IDS 2018 Intrusion CSVs (CSE-CIC-IDS2018) 데이터 셋에 02-16-2018.csv 파일을 사용하였다. 데이터 개수는 100만개 가량 존재하고, 데이터 셋에는 총 79개 정도의 Feature가 존재하였다. Label은 Benign, DoS attacks-Hulk, DoS attacks-SlowHTTPTest 총 3개가 존재했다. 이 중 DoS attacks-Hulk와 DoS attacks-SlowHTTPTest 공격은 다른 유형의 DoS 공격이지만, 둘 다 DoS 공격은 맞기에 DoS attacks으로 라벨링하였다.

100만개 가량의 데이터를 모두 사용할 경우, 모델을 훈련 시키는데 너무 오랜 시간이 들기에, Benign Label 과 DoS attacks Label 에서 각각 20만개씩 뽑아 사용하였다.

DoS attacks을 결정 짓는 Feature가 무엇인지 알아보고자 Feature Selection을 진행하였고, Filter method 방식을 사용하였다. 각 Feature가 Label과 어떤 상관 관계를 가지고 있는지 확인하고자 했다. Threshold 값은 0.5로 지정한 후, 진행하였다.



Feature Selection, 그림 10

표 1. 양의 상관 관계 특성

Flow Pkts/s	flow packets rate that is number of packets transferred per second
Fwd Pkts/s	Number of forward packets per second
Bwd Pkts/s	Number of backward packets per second
PSH Flag cnt	Number of PSH Flag of TCP
Init Fwd Win Bytes	Number of bytes sent in initial window in the forward direction
Fwd Seg Size Min	Minimum segment size observed in the forward direction

표 2. 음의 상관 관계 특성

Dst Port	Destination Port Number
Tot Fwd Pkts	Total packets in the forward direction
Tot Bwd Pkts	Total packets in the backward direction

TotLen Fwd Pkts	Total size of packet in forward direction
Fwd Pkt Len Max	Maximum size of packet in forward direction
Fwd Pkt Len Mean	Mean of size of packet in forward direction
Fwd Pkt Len Std	Std of size of packet in forward direction
Bwd Pkt Len Max	Maximum size of packet in backward direction
Bwd Pkt Len Mean	Mean size of packet in backward direction
Bwd Pkt Len Std	Standard deviation size of packet in backward direction
Fwd Header Len	Total bytes used for headers in the forward direction
Bwd Header Len	Total bytes used for headers in the forward direction
Pkt Len Max	Maximum length of a flow
Pkt Len Mean	Mean length of a flow
Pkt Len Std	Standard deviation length of a flow
Pkt Len Var	Minimum inter-arrival time of packet
ACK Flag Cnt	Number of packets with ACK
Pkt Size Avg	Average size of packet
Fwd Seg Size Avg	Average size observed in the forward direction
Bwd Seg Size Avg	Average size observed in the backward direction
Subflow Fwd Pkts	The average number of packets in a sub flow in the forward direction
Subflow Fwd Byts	The average number of bytes in a sub flow in the forward direction
Subflow Bwd Pkts	The average number of packets in a sub flow in the backward direction
Init Bwd Win Byts	# of bytes sent in initial window in the backward direction
Fwd Act Data Pkts	# of packets with at least 1 byte of TCP data payload in the forward direction

1.4.2.2. CSV 파일 읽기 및 전처리 (Preprocessing)

INF 값 혹은 NaN 값을 모두 처리해주었고, Timestamp 열은 모델의 입력으로 사용되지 않아 삭제해주었다. 또한 앞서 데이터 셋 부분에서 말했듯이 DoS

attacks-SlowHTTPTest와 DoS attacks-Hulk를 DoS attacks 라벨 하나로 통일시켰다.

```
def experiment(dataFile):
    #Creating data for analysis
    time_gen = int(time.time())
    global model_name
    global tensorboard
    model_name = f"{dataFile}_{time_gen}"
    #$ tensorboard --logdir=logs/
    tensorboard = TensorBoard(log_dir='logs/{}'.format(model_name))

    pickleDump = '{}.pickle'.format(dataFile)
    if os.path.exists(pickleDump):
        df = pd.read_pickle(pickleDump)
    else:
        df = pd.read_csv(dataFile)
        # replace +ve and -ve infinity with NaN
        df.replace([np.inf, -np.inf], np.nan, inplace=True)
        df.dropna(inplace=True)
        df.fillna(0, inplace=True) # NaN -> 0
        df.to_pickle(pickleDump)
    return df

dataFile = 'dos_attack_100000.csv'
df = experiment(dataFile)
df = df.drop(['Unnamed: 0', 'Timestamp'], axis=1)
# df = df.drop(['Timestamp'], axis=1)
df.replace(to_replace=["DoS attacks-SlowHTTPTest", "DoS attacks-Hulk"],\
            value="DoS attacks", inplace=True)

# label name change
df = df.replace({'Label', 'Benign'}, 0)
df = df.replace({'Label', 'DoS attacks'}, 1)

df = df.astype(float)
df.reset_index(inplace=True)
```

1.4.2.3. 모델 구성 (Model Composition)

모델은 Input Layer, Hidden Layer 2개, Output Layer로 구성된 Neural Network Model을 사용하였다.

```python
def baseline_model(inputDim=-1, out_shape=(-1,)):
    global model_name
    model = Sequential()

    if inputDim > 0 and out_shape[1] > 0:
        model.add(Dense(inputDim, activation='relu', input_shape=(inputDim,)))
        print(f"out_shape[1]:{out_shape[1]}")
        model.add(Dense(128, activation='sigmoid'))
        model.add(Dropout(0.2))

        model.add(Dense(out_shape[1], activation='softmax')) #This is the output layer

        if out_shape[1] > 2:
            print('Categorical Cross-Entropy Loss Function')
            model_name += "_categorical"
            model.compile(optimizer='adam',
                     loss='categorical_crossentropy',
                     metrics=['accuracy'])
        else:
            model_name += "_binary"
            print('Binary Cross-Entropy Loss Function')
            model.compile(optimizer='adam',
                    loss='binary_crossentropy',
                    metrics=['accuracy'])
    return model
```

1.4.2.4. 모델 훈련 및 성능 테스트 (Model Training)

Outlier를 방지하고자 Normalization을 진행한 후, Train DataSet과 Test Dataset을 8대 2로 비율로 분리시키고, 훈련을 진행하였다. 이후, model.predict를 통해 나온 예측 값과 실제 값을 비교하여 정확성을 측정하였다.

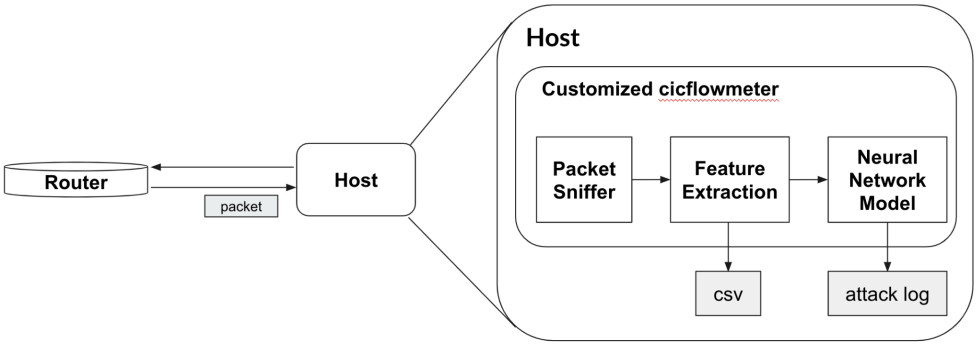
```
def experiment(data, optimizer='adam', epochs=10, batch_size=10):

    data_x = data[:-1]
    data_y = data.pop('Label')
    encoder = LabelEncoder()
    encoder.fit(data_y)
    data_y = encoder.transform(data_y)
    dummy_y = to_categorical(data_y)

    data_x = normalize(data.values)

    #define 5-fold cross validation test harness
    inputDim = len(data_x[0])

    print('input= ', data_x.shape[0],'inputdim = ', data_x.shape[1])
    num=0
    sss = StratifiedShuffleSplit(n_splits=1, test_size=0.2, random_state=7)
    start = timer()

    for train_index, test_index in sss.split(X=np.zeros(data_x.shape[0]), y=dummy_y):
        X_train, X_test = data_x[train_index], data_x[test_index]
        y_train, y_test = dummy_y[train_index], dummy_y[test_index]

        #create model
        model = baseline_model(inputDim, y_train.shape)

        #train
        model.fit(x=X_train, y=y_train, epochs=epochs, batch_size=batch_size, \
                  verbose=2, callbacks=[tensorboard], validation_data=(X_test, y_test))

        #save model
        model.save('ids_dos_attack.h5')

        num+=1

    elapsed = timer() - start
    pred = model.predict(X_test)
    print("Pred:", pred)

    scores = model.evaluate(X_test, y_test, verbose=1)
    print(model.metrics_names)
    acc, loss = scores[1]*100, scores[0]*100
    print('Baseline: accuracy: {:.2f}%: loss: {:.2f}'.format(acc, loss))
```

측정한 결과 정확도는 0.999가 나왔다. 하지만, 이는 Feature가 많이 사용되었고, Regularation을 진행하지 않았기에 때문에 어느 정도 OverFitting 되었을 가능성이 있다고 생각한다.

```
Pred: [[9.9999994e-01 0.0000000e+00]
 [9.9999994e-01 8.1760084e-37]
 [9.9999994e-01 5.2788853e-38]
 ...
 [9.9999994e-01 9.9191959e-37]
 [3.6415936e-11 9.9999994e-01]
 [9.9999994e-01 8.0504451e-38]]
2500/2500 [==============================] - 4s 2ms/step - loss: 0.0022 - accuracy: 0.9990
['loss', 'accuracy']
Baseline: accuracy: 99.90%: loss: 0.22
```

1.4.2.5. cicflowmeter에 모델 적용 (Combination of cicflowmeter and IDS model)

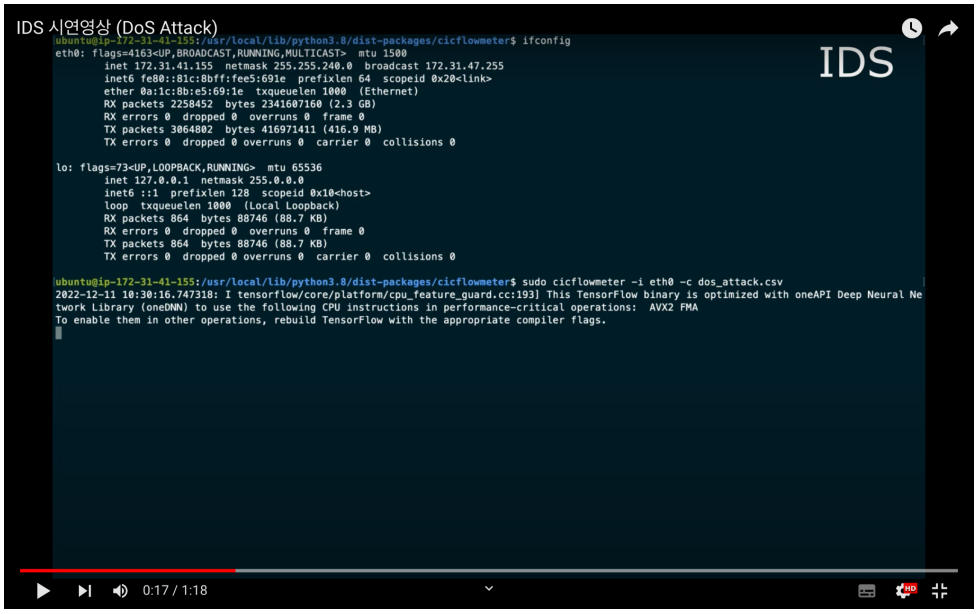
기존 cicflowmeter에서 Model을 추가해주었고, Attack이 들어왔을 때 공격이 들어왔는지를 파악하기 위해 attack log를 남기도록 구성하였다.



Customized cicflowmeter 구조, 그림 11

1.4.2.6 결과

URL Link : https://www.youtube.com/watch?v=wBjDQ6sChoc



IV. Conclusion

IDS를 사용하는 다양한 어플리케이션을 사용해 봄으로써 IDS에 대한 이해 및 분석을 쉽게 할 수 있었고, 또한 IDS의 장,단점을 파악 할 수 있었다. 침입 탐지 시스템을(suricata, tripwire) 이용하여 정해둔 Rule을 통해 패킷들을 확인할 수 있었고, 호스트 상에 파일 변화를 감지 및 분석을 할 수 있었으며, 머신러닝을 통한 IDS 모델을 만들어 DoS 공격을 탐지할 수 있었다.

V. References

1. Solarmainframe, 2020. IDS 2018 Intrusion CSVs (CSE-CIC-IDS2018) https://www.kaggle.com/datasets/solarmainframe/ids-intrusion-csv
2. CICFlowMeter github, https://github.com/ahlashkari/CICFlowMeter
3. Slowloris DoS attack, https://blog.naver.com/is_king/221629346522
4. [Snort] 스노트 기본 동작 / Snort Rule 구조, https://latte-is-horse.tistory.com/21