LID-DS Report

LID-DS(2021) Unsupervised (1)

LID-DS Dataset 설명

LID-DS는 호스트 기반 침입 탐지(Host Based Intrusion Detection System, HIDS) 데이터 셋으로 Ubuntu 18.04버전에서 기록이 된 데이터이다. Attack Type은 15가지가 있으며,각 공격 유형별로 시나리오 폴더가 존재하고, 폴더안의 .sc파일을 통해 로그를 확인 가능하다.

Attack Type	Train Scenario	Validation Scenario	Test Scenario (Contain Attack)	Test Scenario (Only Normal)	Data Size (모든 파일 압축되어 있음)
Bruteforce_CWE-309	210	60	139	768	1.3GB
CVE-2012-2122	210	60	120	762	0.69GB
CVE-2014-0160	210	60	120	758	0.39GB
CVE-2017-7529	210	52	119	729	5.7GB
CVE-2017-12635_6	208	58	119	753	12.6GB
CVE-2018-3760	210	60	122	737	4GB
CVE-2019-5418	210	60	120	760	7.3GB
CVE-2020-9484	210	60	120	759	5.2GB
CVE-2020-13942	211	60	118	756	15.3GB
CVE-2020-23839	210	57	120	760	1.4GB
CWE-89-SQL-injection	210	60	120	760	3.7GB
EPS_CWE-434	210	60	121	759	10GB
Juice-Shop	210	60	121	760	7.3GB
PHP_CWE-434	210	59	118	760	3.8GB
ZipSlip	210	59	118	760	8.7GB

원본 데이터의 특성은 총 8개로 구성되어 있고 정보는 아래의 표와 같다.

Feature	Information
Event_time	Event가 발생한 시간
cpu	thread id를 나타냄
user_uid	해당 프로세스를 실행한 사용자의 정보
process	실행된 프로세스의 정보
process_id	프로세스 ID
event_type	호출된 System Call
event_direction	프로세스 호출, 반환을 나타냄(호출 :

	>, 반환 : < 으로 표시)
event_argument	System Call의 인자 (ex. res, fd..)

Data Preprocessing

System Call이 중요한 특성이기 때문에, Gensim의 Word2Vec Library를 사용하여 Embedding vector를 생성한다. 각 Scenario의 System call sequence를 하나의 문장으로 보고 그것들을 모아서 Word2Vec의 학습 데이터로 사용한다.

```
            system_calls = sample['event_type']
            sentence.append(system_calls.tolist())
    model = Word2Vec(sentences=sentence, vector_size=v_size, window=window_size, workers=0,
    return model
```

Event argument 특성을 사용하기위하여 특성값을 파싱하고, 값을 불러오기 위한 “extract_params”를 만들어 System call의 반환 값인 “res”특성을 사용하였다.

```
def extract_params(arg_list): #extract r
    arg_list = arg_list.astype(str)
    res = []
    size = []
    fd = []
    for i in range(len(arg_list)):
        tmp = arg_list[i].strip().split(' ')
        params = {}
        for e in tmp:
            try:
                split = e.split('=',1)
                params[split[0]] = split[1]
            except:
                pass
        try:
            if int(params['res']) < 10000000:
                res.append(int(params['res']))
            else:
                res.append(-1)
        except:
            res.append(0)
```

Convoultion 1D의 입력데이터는 3차원이므로 데이터를 3차원으로 변환하여 주는 함수인 “making_model_input” 함수를 만들어주었다. Sequence의 길이는 30으로 고정하고 전처리를 진행하였다.

In [241]:
```
model_input_sliding_size = 30
np.set_printoptions(threshold=np.inf)
```

In [243]:
```
def making_model_input(x,model_input_sliding_size):
    s_size = model_input_sliding_size
    x = np.array(x)
    data = []
    for i in range(x.shape[0]-s_size+1):
        data.append(x[i:i+s_size][:])

    return np.array(data)
```

결론적으로 총 사용한 특성은 System call, Event direction, return value 3개의 특성이다. “res”특성의 스케일을 조정해주기 위하여 MinMaxScaler를 사용하였으며, event_direction은 2가지의 값을 가져 OneHotEncoding을 진행 하고, 하나의 행을 삭제하였다.(LabelEncoding과 같은 의미)

(결론적인 입력데이터 차원은 System call embedding vector size + 2)

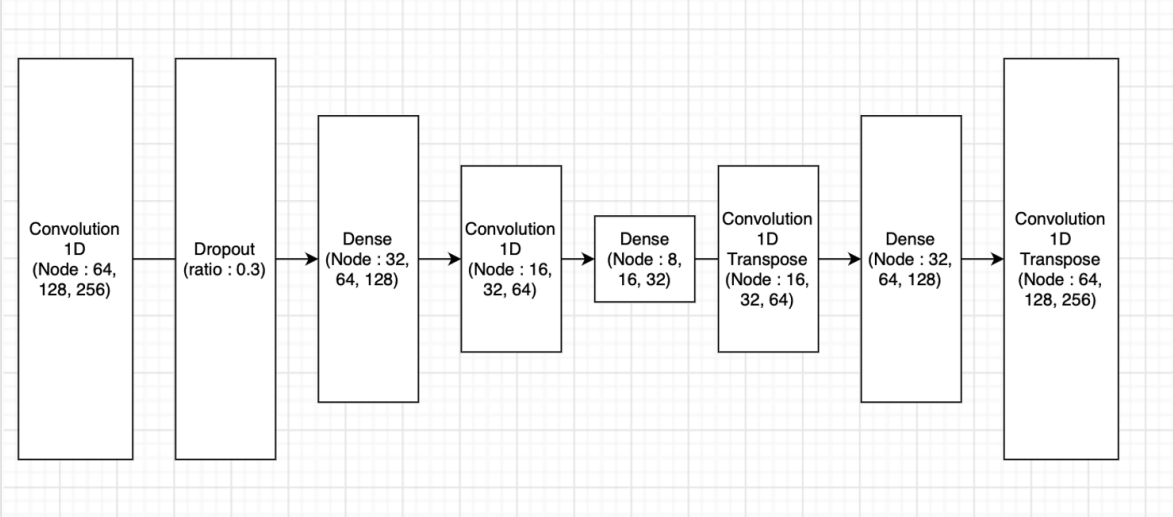
데이터의 크기가 커 Pytorch의 Dataloader와 비슷한 기능인 Keras의 Sequence 라이브러리를 사용하여 데이터를 로드하였다.

In [253]:
```
class MyDataset(Sequence):
    def __init__(self, x, y, batch_size):
        self.x=x
        self.y=y
        self.batch_size = batch_size

    def __getitem__(self, index):
        batch_x = self.x[index*self.batch_size:(index+1)*self.batch_size]
        batch_y = self.y[index*self.batch_size:(index+1)*self.batch_size]
        return batch_x,batch_y

    def __len__(self):
        return math.ceil(len(self.x)/self.batch_size)

```

In [254]:
```
train_loader = MyDataset(train_x,train_x,BS)
val_loader = MyDataset(val_x,val_x, BS)
```

모델 구조

Conv1d - Dense Denoising autoencoder




```
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
conv1d_16 (Conv1D)           (None, 30, 128)           4736

dropout_7 (Dropout)          (None, 30, 128)           0

dense_24 (Dense)             (None, 30, 64)            8256

conv1d_17 (Conv1D)           (None, 30, 32)            6176

dense_25 (Dense)             (None, 30, 16)            528

conv1d_transpose_16 (Conv1D  (None, 30, 32)            1568
Transpose)

dense_26 (Dense)             (None, 30, 64)            2112

conv1d_transpose_17 (Conv1D  (None, 30, 128)           24704
Transpose)

dense_27 (Dense)             (None, 30, 12)            1548

=================================================================
Total params: 49,628
Trainable params: 49,628
Non-trainable params: 0
_________________________________________________________________
```

Example

모델 파라미터

- Batch size = 32
- Epoch = 30
- Conv1d Layer kernel size = 3

- Optimaizer = adam
- Initial learning rate = 1e-3
- Kernel initializer = He initializer
- Activation function = relu
- Loss function = MSE
- EarlyStopping
  - patience = 5
  - monitor = val_liss
- Learning rate scheduler = ReduceOnLRPlateau(patience = 5)
- ModelCheckPoint
  - monitor = val_loss

In [*]:

```
def modeling(x):
    x_train = x
    initializers = keras.initializers.he_normal(seed=SEED)
    model = keras.Sequential()
    model.add(layers.Input(shape=(x.shape[1],x.shape[2])))
    model.add(layers.Conv1D(filters=128, kernel_size=3, padding='same',dilation_rate=1, activation="relu",kernel_in
    model.add(layers.Dropout(0.3))
    model.add(layers.Dense(64,activation='relu',kernel_initializer = initializers))
    model.add(layers.Conv1D(filters=32, kernel_size=3, padding='same',dilation_rate=1, activation="relu",kernel_ini
    model.add(layers.Dense(16,activation='relu',kernel_initializer = initializers))
    model.add(layers.Conv1DTranspose(filters=32, kernel_size=3, padding='same',dilation_rate=1, activation="relu",k
    model.add(layers.Dense(64,activation='relu',kernel_initializer = initializers))
    model.add(layers.Conv1DTranspose(filters=128, kernel_size=3, padding='same',dilation_rate=1, activation="relu",
    model.add(layers.Dense(x_train.shape[2],activation='linear',kernel_initializer = initializers)) #error backprop
    model.compile(optimizer='adam', loss='mse')
    model.build(x_train.shape)
    model.summary()
    return model
early_stopping = EarlyStopping(monitor = 'val_loss', min_delta = 0, patience = 10, mode = 'auto')
model_check_point = ModelCheckpoint(filepath = model_save_dir+'DL/'+model_name+'.h5', monitor = 'val_loss',save_bes
Reduce_lr = ReduceLROnPlateau(monitor= 'val_loss', factor=0.5, patience=5, min_lr=1e-6,mode = 'min')
model_ae = modeling(train_x)
history= model_ae.fit(train_loader, validation_data=val_loader, epochs = 30
                      ,callbacks=[early_stopping,model_check_point,Reduce_lr]).history
```

테스트 방법

먼저 라벨은 LID-DS에 존재하는 공격이 진행된 시간값 이후를 모두 라벨로 설정하였다.

In [ ]:

```
def make_label(directory , scenario_name):
    sample = pd.read_csv(directory+"/test/normal_and_attack/"+scenario_
    json_csv = pd.read_csv(directory+'/outline_json/test_normal_and_at
    #print(json_csv)
    tmp_json_csv =json_csv.loc[json_csv['scenario_name'] == scenario_na
    tmp_json_csv = tmp_json_csv.astype(str)
    tmp_json_csv = tmp_json_csv.tolist()
    #print(tmp_json_csv)
    start_ex = tmp_json_csv[0][:12]
    start_ex = int(start_ex)

    event_record_time = sample['event_time']
    event_record_time = event_record_time.astype(str)
    event_record_time = event_record_time.apply(lambda x : x[:12])
    event_record_time = event_record_time.astype(int)

    label = event_record_time[(event_record_time == start_ex)]
    label = label.index.tolist()
    real_label = label[0]
    return real_label
```

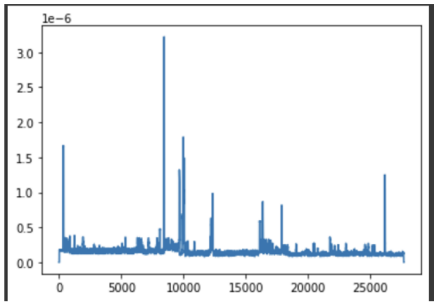
그리고, 원본 데이터는 2차원이므로, 복원한 데이터(3차원)를 다시 2차원으로 변환하여주는 코드를 작성하였다.

In [ ]:

```
def make_2d(x):
    arr = np.empty((x.shape[0],x.shape[2]))
    #print(arr.shape)
    for i in range(x.shape[0]):
            arr[i] = x[i,(x.shape[1]-1),:]
    return arr
```

예측한 데이터를 2차원으로 변환하여 준 후 원본 데이터와 MSE로 재구현 오차를 계산한다. 그리고 시계열 데이터의 특성을 반영하기 위하여 이동평균으로 error smothing 작업을 진행한 후 특정 임계값 이상이면 공격이라고 판단하였고, 이 임계값 이상인 인덱스가 라벨의 존재한 시간 이후의 인덱스라면 공격이라고 판정하였다. 정상 데이터 라벨이 존재하지 않는 것과 같은 것이므로 임계치 이상의 오차가 발생한다면 오탐이라고 판정하였다.

```
error = test_chk- pred
mse = np.mean(np.power(error,2),axis=1)
mean_window = mse.rolling(30,center=True).mean()
window_error = mean_window.fillna(0)
```



.

```
for threshold in threshold_list:
    chk =0
    flag_real = 0
    flag_false=0
    checking_index = np.array(np.where(window_error > threshold)).reshape(-1)
    checking_index = checking_index + 30

    if len(np.where(checking_index >= real_label)) > 0:
        flag_real = 1

    if len(np.where(checking_index < real_label)) > 0 :
        flag_false = 1


    if flag_false > 0:
```

성능 평가

총 4가지의 지표로 성능을 평가하였다.

- FPR
- Recall
- Precision
- F1 Score