LID-DS(2021) Unsupervised

Data set summary

LID-DS is a modern host based intrusion detection system (HIDS) data set.

Recorded on a modern operating system (Ubuntu 18.04).

- Attack Type
  - Bruteforce_CWE-309
  - CVE-2012-2122
  - CVE-2014-0160
  - CVE-2017-7529
  - CVE-2017-12635_6
  - CVE-2018-3760
  - CVE-2019-5418
  - CVE-2020-9484
  - CVE-2020-13942
  - CVE-2020-23839
  - CWE-89-SQL-injection
  - EPS_CWE-434
  - Juice-Shop
  - PHP_CWE-434
  - ZipSlip

  ∴ 15개의 공격유형이 존재함
- 각 공격 유형별로 Scenario folder가 존재하고, 폴더 내에 .sc파일을 통해 Log 확인가능
  - .sc file을 .csv파일로 변환해서 사용함
- Feature
  - Event_time → Event가 발생한 시간

- cpu → thread_id
- user_uid → 해당 프로세스를 실행한 사용자의 정보
- process → 실행된 process의 정보
- process_id
- event_type → 호출된 System Call
- event_direction → 호출, 반환을 나타냄
- event_argument → System call의 인자(res,fd…)

∴ 총 8개의 특성을 가지고 있음

	event_time	cpu	user_uid	process	process_id	event_type	event_direction	event_argument
0	1631011407593552353	0	937365	apache2	937365	select	<	res=0
1	1631011407593556052	0	937365	apache2	937365	wait4	>	
2	1631011407593558216	0	937365	apache2	937365	wait4	<	
3	1631011407593559018	0	937365	apache2	937365	select	>	
4	1631011407697748500	33	937368	apache2	937368	epoll_wait	<	res=1
5	1631011407697752150	33	937368	apache2	937368	accept	>	flags=0
6	1631011407697758445	33	937368	apache2	937368	accept	<	fd=12(<4t>192.168.208.8:33586->192.168.208.24:443) tuple
7	1631011407697762626	33	937368	apache2	937368	semop	>	semid=0
8	1631011407697766942	33	937368	apache2	937368	semop	<	res=0 nsops=1 sem_num_0=0 sem_op_0=1 sem_flg_0=2(SE
9	1631011407697769816	33	937366	apache2	937366	semop	<	res=0 nsops=1 sem_num_0=0 sem_op_0=-1 sem_flg_0=2(SI
10	1631011407697770238	33	937368	apache2	937368	getsockname	>	
11	1631011407697770766	33	937368	apache2	937368	getsockname	<	

Data Example (.sc file converts to .csv)

- Data size(.zip으로 압축된 상태의 size)
  - Bruteforce_CWE-309
    - 1.3GB
  - CVE-2012-2122
    - 691.2MB
  - CVE-2014-0160
    - 391.5MB
  - CVE-2017-7529
    - 5.7GB
  - CVE-2017-12635_6
    - 12.6GB

    - CVE-2018-3760
        - 4GB
    - CVE-2019-5418
        - 7.3GB
    - CVE-2020-9484
        - 5.2GB
    - CVE-2020-13942
        - 15.3GB
    - CVE-2020-23839
        - 1.4GB
    - CWE-89-SQL-injection
        - 3.7GB
    - EPS_CWE-434
        - 10GB
    - Juice-Shop
        - 7.3GB
    - PHP_CWE-434
        - 3.8GB
    - ZipSlip
        - 8.7GB

Data Split Information

- Training(Only normal behaivor)
    - 210개의 scenario
- Validation(Only normal behaivor)
    - 60개의 scenario
- Test
    - Only normal behaivor

        - 공격 유형별로 약 120개의 scenario
    - Contain attack behaivor
        - 공격 유형별로 약 750개의 scenario
- .csv파일의 크기가 크므로 데이터를 얼마나 선택해서 할 건지 토의가 필요함

Data Preprocessing

- System Call은 Gensim의 Word2Vec Library를 사용하여 Embedding vector를 생성한다. 데이터에서 연속적으로 불려지는 System call의 Sequence를 하나의 문장으로 보고 Sliding window 알고리즘을 사용하여 문장들을 “sentence” list에 넣고, 그것을 Word2vec의 학습 데이터로 사용한다. 예제 코드는 문장을 구성하는 System Call의 개수는 30개(W2V_Sliding_Size), window size 4(W2V_Window_size), Embedding vector의 size은 100(W2V_Vector_size)를 사용하였다.

```
In [5]: W2V_Sliding_size = 30
        W2V_Window_size = 4
        W2V_Vector_size = 100

In [6]: def W2V(name,sliding_size,window_size,v_size): # .sc -> .csv
            Base_tmp_dir = Base_dir+name
            file_dir = Base_tmp_dir+"/"+ 'training'
            file_list = sorted(os.listdir(file_dir))
            sentence = []
            if '.DS_Store' in file_list:
                 file_list.remove('.DS_Store') #Mac의 경우만 존재하는 것 같음 Mac환경이 아니라면 이 row를 무방함
            print(len(file_list))
            for e in tqdm(file_list):
                sample = pd.read_csv(file_dir + "/" + e + "/" + e+".csv")

                system_calls = sample['event_type']
                for i in range(len(system_calls)-sliding_size+1):
                    if system_calls[i:i+sliding_size].tolist() not in sentence:
                        sentence.append(system_calls[i:i+sliding_size].tolist())
            model = Word2Vec(sentences=sentence, vector_size=v_size, window=window_size, workers=4, sg=1)
            return model

In [7]: w2v_model = W2V('CVE-2014-0160',W2V_Sliding_size,W2V_Window_size,W2V_Vector_size)
        #model = W2V('CWE-89-SQL-injection',W2V_Sliding_size,W2V_Window_size,W2V_Vector_size)
        #plz set attack name..
```

word2vec code

- “embedding_syscall” 함수는 학습된 Word2vec 모델을 사용하여 System Call을 Embedding하는 함수로, Data를 전처리하는 부분에서 사용하였다.

```
In [8]: def embedding_syscall(syscall): #해당하는 w2v모델과 공격이 일치하는지 확인해야함.
            embedding_vector = w2v_model.wv.get_vector(syscall)
            return embedding_vector
```

embedding_syscall function

- Data의 event_argument는 공백을 기준으로 데이터를 parsing할 수 있게 되어있는데, “extract_param”함수는 이 특성을 parsing하고 “res”, “size”의 값만을 사용할 수 있게 하는 함수이다. “res” 값은 System call의 return value를 담고있는 값이고, “size”는 “read”, “write”와 같은 System call에서 작업이 필요한 Data의 크기를 알려주는 값이다. “fd”는 file descriptor를 나타낸다. System Call의 Return value인 “res” 값은 System call의 작업 반환 값을 담고 있어 특성으로 사용하였고, “size”는 비정상적인 데이터 크기를 가지는 작업을 알아낼 수 있는 특성이라고 생각하여 사용하였다. “fd”는 파일의 접근 정보를 담고 있으므로 특성으로 선택해야 한다고 생각하여 사용하였다. 일반적인 값보다 비정상적으로 큰 값을 가지면 sklearn의 MinMaxScaler를 사용한다 하더라도 정상적인 값을 제대로 표현하지 못한다고 판단하여(ex. 56465CC2B000와 같은 값 → 매우 큼) “res”의 값이 문자열을 포함하고 있거나, 비정상적으로 큰 값이라면 이상이라고 간주하였다. Linux System call의 return value는 System call이 에러가 있으면 음수를 반환하는데, 이 이유로 이상값이라고 판단한 값은 -1로 치환하였다. “fd”, “size”도 같은 방법을 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)

        try:
            if int(params['size']) < 10000000:
                size.append(int(params['size']))
            else:
                size.append(-1)
        except:
            size.append(0)
        try:
            tmp_fd = params['fd'].split('(')[0]
            if int(tmp_fd) < 10000000:
                fd.append(tmp_fd)

            else:
                fd.append(-1)
        except:
            fd.append(0)
    return res, size, fd;
```

- 우리가 사용하는 LID-DS Dataset은 Event_time, cpu(thread_id로 추정), user_uid, process, process_id, event_type(System Call), event_direction, event_argument 총 8개로 구성이 되어있다. Event_time은 그저 Event가 발생한 시간을 기록한 정보이기에 사용하지 않고, cpu 특성도 이상 행위에대한 유의미한 정보를 담고 있을 거라고 생각되지 않아 사용하지 않았다. process 특성은 결측값이 많이 있어 결측값 처리에 어려

움이 있었다. 그렇기에 제거를 하였고 process 특성을 사용하지 않기 때문에 process_id 특성도 제거하여 결론적으로 user_uid, event_type, event_direction, event_argument(res,size,fd) 총 4개의 특성을 사용하였다. uid, res, size, fd의 값은 스케일의 크기가 커 Embedding된 event_type(System call)의 특성을 모델이 잘 반영하지 못할 것이라고 판단되어 sklearn의 MinMaxScaler library 를 사용하여 정규화를 진행하였다. event_direction은 “>” 또는 “<”의 값을 가지는데, 이를 모델의 입력값으로 사용하기 위하여 OneHotEncoding을 진행하였다. 램의 부족으로 인해 ‘CVE-2014-0160’ 공격유형에 대해서만 작업을 진행하였고 training data는 총 60개의 csv파일을 사용하였으며, validation data는 총 10개의 csv파일을 사용하였다.

In [227]:
```
def make_training(name):
    Base_tmp_dir = Base_dir+name
    file_dir = Base_tmp_dir+"/"+ 'training'
    file_list = sorted(os.listdir(file_dir))
    if '.DS_Store' in file_list:
        file_list.remove('.DS_Store') #Mac의 경우만 존재하는 것 같음 Mac환경이 아니라면 이 row를 무방함
    print(len(file_list))
    sample_event_direction = []
    sample_event_type = []
    #sample_cpu = []
    sample_user_uid = []
    sample_res = []
    sample_size = []
    sample_fd = []
    file_list = file_list[:30]
    for e in tqdm(file_list):
        sample = pd.read_csv(file_dir + "/" + e + "/" + e+".csv")
        sample_event_direction.extend(sample['event_direction'])

        sample_event_type_tmp = sample['event_type'].tolist()
        sample_event_type_tmp = list(map(embedding_syscall,sample_event_type_tmp))
        sample_event_type.extend(sample_event_type_tmp)
        #sample_cpu.extend(sample['cpu'])
        sample_user_uid.extend(sample['user_uid'])
        sample_param = sample['event_argument']
        tmp_res, tmp_size, tmp_fd = extract_params(sample_param)
        sample_fd.extend(tmp_fd)
        sample_res.extend(tmp_res)
        sample_size.extend(tmp_size)

   # return_x = pd.DataFrame(zip(sample_cpu,sample_user_uid), columns=['cpu','user_uid'])
    return_x = pd.DataFrame(zip(sample_user_uid,sample_res,sample_size,sample_fd),columns =['user_uid','res','size','fd
    '''
    Normalization part
    '''
    return_x = scaler.fit_transform(return_x)
    return_x = pd.DataFrame(return_x,columns=['uid','res','size','fd'])



    return_x['event_direction'] = sample_event_direction
    return_x = pd.get_dummies(return_x, columns=['event_direction'])
    sample_event_type_df = pd.DataFrame(sample_event_type)
    return_x = pd.concat([return_x, sample_event_type_df],axis=1)



    return return_x
```

make_training function

In [222]:
```
training_data
```

Out[222]:

	uid	res	size	fd	event_direction_<	event_direction_>	0	1	2	3	4	5	6	
0	0.020928	0.000122	0.0	0.575758	1	0	-0.002023	-0.161582	-0.179201	-0.121125	0.245174	-0.008946	0.333280	0.3
1	0.020928	0.000000	0.0	0.575758	0	1	-0.013264	-0.173307	0.010680	-0.052367	-0.142286	-0.263078	0.141409	0.0
2	0.020928	0.000000	0.0	0.969697	1	0	-0.013264	-0.173307	0.010680	-0.052367	-0.142286	-0.263078	0.141409	0.0
3	0.020928	0.000000	0.0	0.575758	0	1	-0.144213	-0.201155	-0.057150	0.117878	0.171947	-0.175354	0.274668	0.0
4	0.020928	0.000061	0.0	0.575758	1	0	-0.144213	-0.201155	-0.057150	0.117878	0.171947	-0.175354	0.274668	0.0
...	...	...	...	...	...	...	...	...	...	...	...	...	...	
117605	0.996785	0.000000	0.0	0.575758	0	1	0.203530	-0.289314	-0.122947	-0.075929	-0.226778	-0.020818	0.137693	0.3
117606	0.996785	0.000061	0.0	0.575758	1	0	0.203530	-0.289314	-0.122947	-0.075929	-0.226778	-0.020818	0.137693	0.3
117607	0.996785	0.000000	0.0	0.575758	0	1	0.389419	-0.445630	-0.082433	0.142438	-0.105448	-0.019965	-0.007656	0.5
117608	0.996785	0.000000	0.0	0.575758	1	0	0.389419	-0.445630	-0.082433	0.142438	-0.105448	-0.019965	-0.007656	0.5
117609	0.996785	0.000000	0.0	0.575758	0	1	0.203530	-0.289314	-0.122947	-0.075929	-0.226778	-0.020818	0.137693	0.3

117610 rows × 56 columns

Example of Training data(Type : Dataframe)

```
def make_validation(name):
    Base_tmp_dir = Base_dir+name
    file_dir = Base_tmp_dir+"/"+ 'validation'
    file_list = sorted(os.listdir(file_dir))
    if '.DS_Store' in file_list:
         file_list.remove('.DS_Store') #Mac의 경우만 존재하는 것 같음 Mac환경이 아니라면 이 row를 무방함
    print(len(file_list))
    sample_event_direction = []
    sample_event_type = []
    #sample_cpu = []
    sample_user_uid = []
    sample_res = []
    sample_size = []
    sample_fd = []
    file_list = file_list[:10]
    for e in tqdm(file_list):
        sample = pd.read_csv(file_dir + "/" + e + "/" + e+".csv")
        sample_event_direction.extend(sample['event_direction'])

        sample_event_type_tmp = sample['event_type'].tolist()
        sample_event_type_tmp = list(map(embedding_syscall,sample_event_type_tmp))
        sample_event_type.extend(sample_event_type_tmp)


        #sample_cpu.extend(sample['cpu'])
        sample_user_uid.extend(sample['user_uid'])
        sample_param = sample['event_argument']
        tmp_res, tmp_size,tmp_fd = extract_params(sample_param)
        sample_res.extend(tmp_res)
        sample_size.extend(tmp_size)
        sample_fd.extend(tmp_fd)

   # return_x = pd.DataFrame(zip(sample_cpu,sample_user_uid), columns=['cpu','user_uid'])
    return_x = pd.DataFrame(zip(sample_user_uid,sample_res,sample_size,sample_fd),
                            columns =['user_uid','res','size','fd'])
    return_x = scaler.transform(return_x)
    return_x = pd.DataFrame(return_x,columns=['uid','res','size','fd'])


    return_x['event_direction'] = sample_event_direction
    return_x = pd.get_dummies(return_x, columns=['event_direction'])
    sample_event_type_df = pd.DataFrame(sample_event_type)
    return_x = pd.concat([return_x, sample_event_type_df],axis=1)

    return return_x
```

make_validation function

- “make_training”, “make_function” 의 반환 값으로 얻은 데이터를 LSTM(or Conv1d)의 input으로 넣어주기 위해 3차원으로 변환하여 주는 작업을 하는 “making_model_input” 함수를 제작하였다. sliding window방법을 사용하여 3차원으로 변환하였다.

```
In [230]: model_input_sliding_size = 30
```

```
In [231]: def making_model_input(x,model_input_sliding_size):
              s_size = model_input_sliding_size
              x = np.array(x)
              data = []
              for i in tqdm(range(x.shape[0]-s_size+1)):
                  data.append(x[i:i+s_size][:])

              return np.array(data)
```

Making_model_input function

```
In [174]: #Make model input
          train_x = making_model_input(training_data,model_input_sliding_size)
          val_x  = making_model_input(validation_data,model_input_sliding_size)

          100%|██████████| 117581/117581 [00:00<00:00, 3073785.61it/s]
          100%|██████████| 27707/27707 [00:00<00:00, 3062551.55it/s]
```

```
In [232]: print(train_x.shape) #shape[0] -> Data - sliding_size + 1
          #val_x = val_x[:10000]
          print(val_x.shape)

          (117581, 30, 56)
          (27707, 30, 56)
```

Model part –

Result of making_model_input

Model

1. Isolation Forest
2. One-Class SVM
3. EilipticEnvelope

---

4. Autoencoder
    - DNN Autoencoder
    - LSTM Autoencoder
    - Bidirectional-LSTM Autoencoder
    - Conv1d Autoencoder
    - Conv1d - LSTM Autoencoder
    - Conv1d - Bidirectional LSTM Autoencdoer
    - Denosing DNN Autoencoder
    - Denosing LSTM autoencoder
    - Denosing Bidirectional-LSTM Autoencoder

- Denosing Conv1d Autoencoder(이 내용은 추가적으로 확인 해봐야합니다.)
- Denosing Conv1d-LSTM Autoencoder
- Denosing Conv1d-Bidirectional LSTM Autoencoder

∴ 현존하는 Autoencoder들 중 성능이 잘 나오는 Autoencoder를 사용할 예정

(Denosing AE는 Dropout layer을 Encoder layer에 추가하여 구현할 예정입니다.)

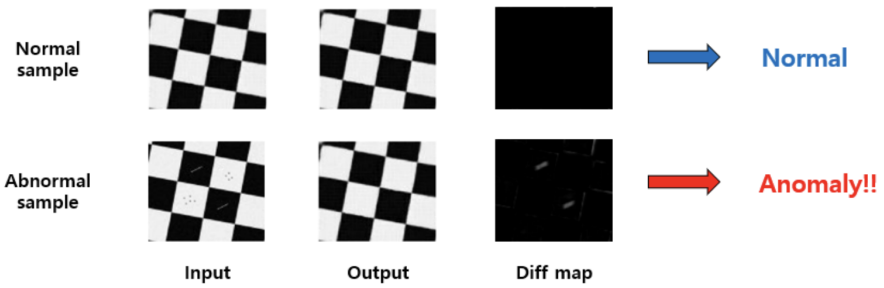


그림 출처: Improving Unsupervised Defect Segmentation by Applying Structural Similarity To Autoencoders, 2019 arXiv

[autoencoder 기반 unsupervised anomaly detection]

Autoencoder를 이용하면 데이터에 대한 labeling을 하지 않아도 데이터의 주성분이 되는 정상 영역의 특징들을 배울 수 있습니다. 이때, 학습된 autoencoder에 정상 sample을 넣어주면 위의 그림과 같이 잘 복원을 하므로 input과 output의 차이가 거의 발생하지 않는 반면, 비정상적인 sample을 넣으면 autoencoder는 정상 sample처럼 복원하기 때문에 input과 output의 차이를 구하는 과정에서 차이가 도드라지게 발생하므로 비정상 sample을 검출할 수 있습니다.

reference : https://hoya012.github.io/blog/anomaly-detection-overview-1/

```python
class AutoEncoder(nn.Module):
    def __init__(self,feature_num):
        super(AutoEncoder, self).__init__()
        self.feature_num = feature_num
        self.conv1d_1 = nn.Conv1d(in_channels = self.feature_num,out_channels=128,kernel_size=4,stride=1,padding='same')
        self.lstm_1 = nn.LSTM(128,64,model_input_sliding_size,batch_first=True)
        self.lstm_2 = nn.LSTM(64,32,model_input_sliding_size,batch_first=True)
        self.lstm_3 = nn.LSTM(32,16,model_input_sliding_size,batch_first=True)
        self.lstm_4 = nn.LSTM(16,32,model_input_sliding_size,batch_first=True)
        self.lstm_5 = nn.LSTM(32,64,model_input_sliding_size,batch_first=True)
        self.lstm_6 = nn.LSTM(64,128,model_input_sliding_size,batch_first=True)
        self.conv1d_2 = nn.Conv1d(in_channels = 128,out_channels=self.feature_num,kernel_size=4,stride=1,padding='same')
        self.tanh = nn.Tanh()
    def forward(self, x):
        x = x.transpose(1,2)
        x = self.conv1d_1(x)
        x = self.tanh(x)
        x = x.transpose(1,2)
        x, _ = self.lstm_1(x)
        x = self.tanh(x)
        x, _  = self.lstm_2(x)
        x = self.tanh(x)
        x, _  = self.lstm_3(x)
        x = self.tanh(x)
        x, _  = self.lstm_4(x)
        x = self.tanh(x)
        x, _  = self.lstm_5(x)
        x = self.tanh(x)
        x, _  = self.lstm_6(x)
        x = self.tanh(x)
        x = x.transpose(1,2)
        x = self.conv1d_2(x)
        x = x.transpose(1,2)
        return x
```

Pytorch Example code

```
In [21]: def modeling(x):
             x_train = x
             model = keras.Sequential()
             model.add(layers.Conv1D(filters=128, kernel_size=8, padding='same', data_format='channels_last',dilation_rate=1, ac
             model.add(layers.Bidirectional(layers.LSTM(64, activation='tanh', input_shape=(x_train.shape[1],x_train.shape[2]),r
             model.add(layers.Dropout(0.2))
             model.add(layers.Bidirectional(layers.LSTM(32, activation='tanh', return_sequences=True,kernel_initializer='glorot_
             model.add(layers.Bidirectional(layers.LSTM(16, activation='tanh', return_sequences=True,kernel_initializer='glorot_
             model.add(layers.Bidirectional(layers.LSTM(32, activation='tanh', return_sequences=True,kernel_initializer='glorot_
             model.add(layers.Bidirectional(layers.LSTM(64, activation='tanh',return_sequences=True,kernel_initializer='glorot_r
             model.add(layers.Conv1D(filters=128, kernel_size=4, padding='same', data_format='channels_last',dilation_rate=1, ac
             model.add(layers.TimeDistributed(layers.Dense(x_train.shape[2],activation='linear'))) #error backpropagation
             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')
         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_x, train_x, epochs = 30, batch_size=16, validation_data=(val_x,val_x)
                                ,callbacks=[early_stopping,Reduce_lr]).history
```

Keras Example code

1. Input Layer
2. Conv1d
    - Param
        - Filter = 128
        - kernel_size=8
        - padding = ‘same’
        - kernel_initializer = ‘glorot_normal’
        - activation = Tangent Hyperbolic

3. Bi-LSTM(5개의 layer로 구성되어 있음)
    - Param
        - Nodes : 64 - 32 - 16 - 32 - 64
        - kernel_initializer = ‘glorot_normal’
        - activation = Tangent Hyperbolic
4. Conv1d
    - Param
        - Filter = 128
        - kernel_size=8
        - padding = ‘same’
        - kernel_initializer = ‘glorot_normal’
        - activation = Tangent Hyperbolic
5. Output Layer

Test Model

- 복원오차에 이동평균을 적용하여 시계열 데이터의 이상을 좀 더 잘 탐지할 수 있게 하였다.

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

이동평균 적용

- 적절한 threshold값을 정하여 특정 threshold값 이상인 error가 특정 횟수 이상 연속적으로 발생한다면 공격이라고 판단하는 정책을 사용하였다. 이 공격유형은 1번의 걸쳐서 일어나기 때문에, 2번이상 공격이라고 판단한다면 그것은 오탐을 한것이기 때문에 “false_negative_count” 변수를 통해 오탐 횟수를 기록하였다.

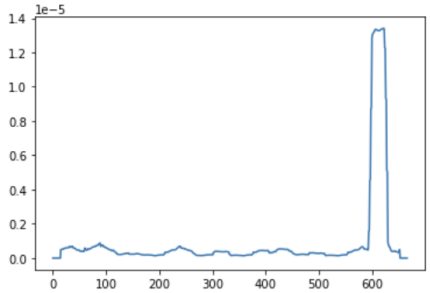
```
flag = 0
chk = 0
for i in tqdm(range(test_chk.shape[0])):
    if window_error[i] > 5*1e-6:
        flag = flag+1
        #print(i)
    else:
        flag = 0

    if flag == 5 and chk > 0:
        false_negative_count = false_negative_count + 1
        print("FALSE ALARM!!!!!!!!!!!!!!!!!!!")


    elif flag == 5 and chk == 0:
        anomaly_count = anomaly_count + 1
        chk=1
        print("ALARM!!!!!!!!!!!!!!!")
        if i < test_chk.shape[0]-30:
            i=i+30
```

Threshold Setting

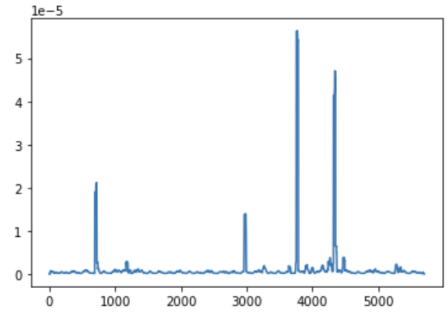
- 공격이 섞여있는 시나리오는 아래와 같이 높은 reconstruction error가 특정 구간에 발생한다. 실제로 plot에서 비정상적으로 error가 측정이 된 부분은 공격이 시작된 부분을 포함하고 있었다. 이러한 경우에 한 scenario에 대해 공격을 감지하였다고 판단하였다.




```
100%|██████████████████████████████████████████| 667/667 [00:00<00:00, 448333.46it/s]
  1%|▏                                          | 1/120 [00:02<04:08,  2.09s/it]
598
599
600
601
602
ALARM!!!!!!!!!!!!!!!
```

Reconstruction error plot(Attack Data)

- 하지만 아래의 사진과 같이 공격 구간이 아닌데도 공격이라고 판단하는 부분이 존재하였다.



Reconstruction error plot(Attack Data), 하지만 False Alarm이 존재함.

```
692
693
694
695
696
ALARM!!!!!!!!!!!!!!!
697
698
699
700
701
702
703
704
705
706
707
708
709
710
711
712
713
714
715
716
717
718
719
720
721
2964
2965
2966
2967
2968
FALSE ALARM!!!!!!!!!!!!!!!!!!
2969
```

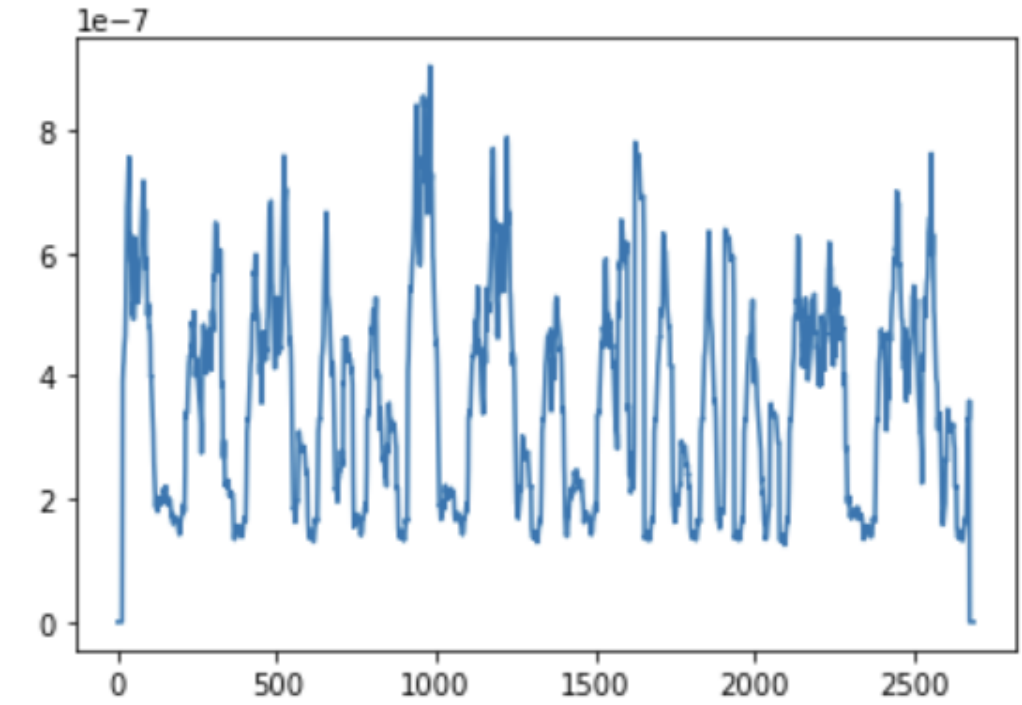
Example

- 결론적으로 제작한 모델에 대해 공격이 섞여있는 scnario에 대해서는 모든 공격을 잡아낸 것으로 판단이 되지만, 어느정도 오탐이 존재하는 것으로 모델이 판단하고 있다. 120개의 scenario중 공격이 존재한다고 판단한 scenario는 120개, 하지만 120개의 공격구간이 존재하지만 158개의 공격구간이 존재한다고 판단하고 있으므로 오탐을 줄일 정책을 강구하여야한다.

```
889
Data count : 120
Whole count : 158
anomaly_count : 120
false negative count : 38
758
```

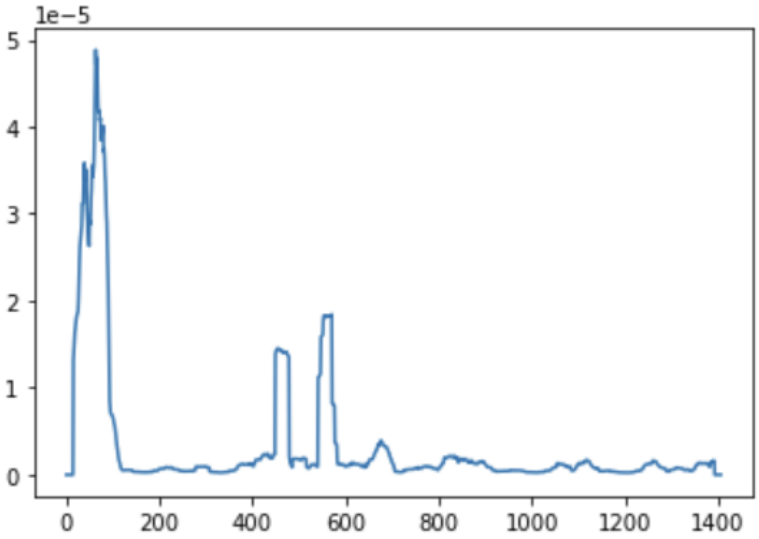
Result

- 정상데이터만을 가지고있는 scenario에 대해서 모델이 예측을 시행하였을 때는 정상적으로 모델이 scenario을 재구현한 경우는 아래와 같이 오차가 $n * (1e-7)$ 정도가 나오는 것을 확인할 수 있다.



Reconstruction error plot(Normal Data)

- 하지만 정상데이터만을 가지고 있는 scenario에 대해서 모델이 예측을 시행하였을 때 모델이 공격이라고 판단한 구간이 존재하는 경우도 있다.




```
100%|██████████████████████████████████████████████████| 1407/14
  0%|                                                   | 2/758 [00:05<3
ALARM!!!!!!!!!!!!!!!
FALSE ALARM!!!!!!!!!!!!!!!!!!!
FALSE ALARM!!!!!!!!!!!!!!!!!!!
clean kapitsa 4236
```

Reconstruction error plot(Normal Data) → 오탐 존재

- 결론적으로 758개의 scenario중에 정상이라고 분류한 scenario는 758-404=354개의 scenario이고, 이상이라고 분류한 scenario는 404개의 scenario이다. threshold값을 조정한다면 충분히 오탐을 줄일 수 있다고 생각하지만, 오탐을 줄일 근본적인 정책이 필요할 것이라고 생각된다.

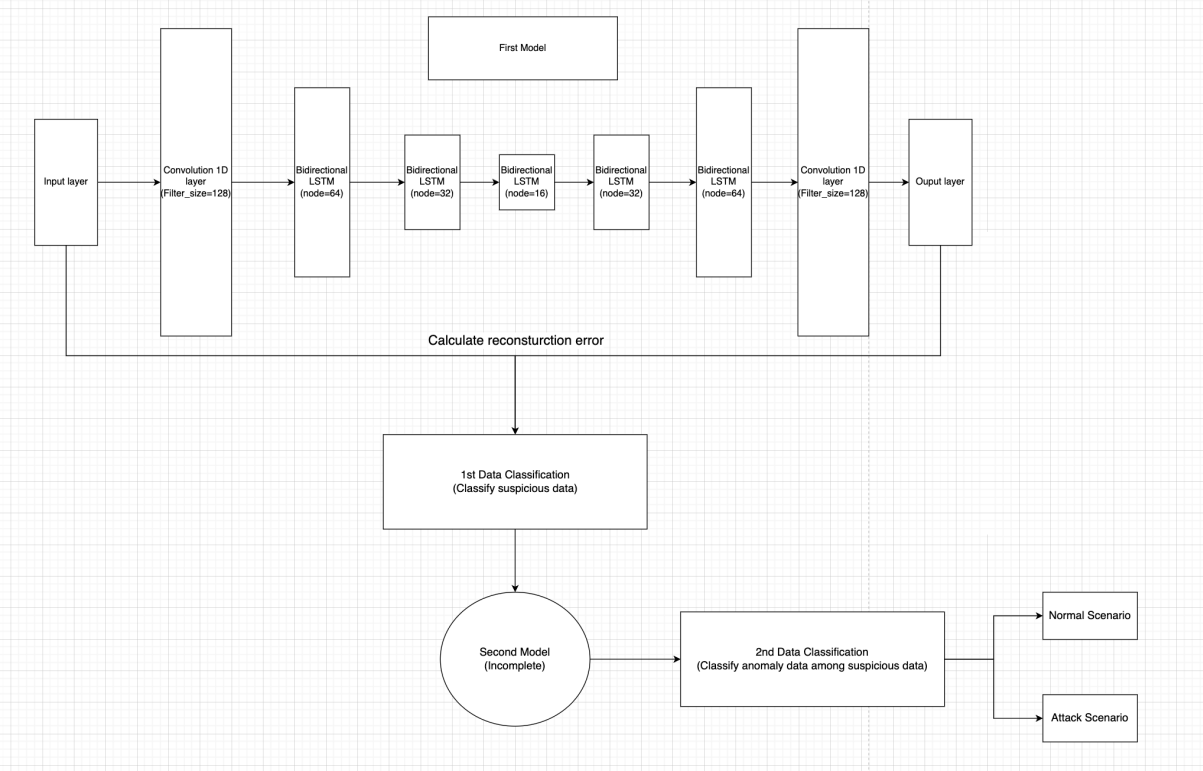
```
Data count : 758
Whole count : 1074
anomaly_count : 404
false negative count : 720
```

Result

개선방법

- 우선 적절한 threshold 값들을 통하여 모델이 이상이라고 탐지한 구간들을 각각 sample로 구성하여 “공격 의심 데이터” 라고 1차적 분류를 진행한다. 그리고 난 후 구성한 sample들을 재분류하는 2차 모델을 구성하여 Recall을 높이는 방법을 생각하였다.

- 1차적 분류를 진행한 Autoencoder는 전체적 데이터에서 reconsturction error를 통해 의심데이터를 분류하는 역할을 한다. 현재 Autoencoder를 사용하면 특정 정상 행위가 reconstrution error가 크게 나오는 것을 확인할 수 있는데, 이러한 이유는 특정 정상 행위의 구간이 전체 데이터에서 매우 적은 비율로 존재하여 이러한 현상이 발생하는 것으로 생각된다. 그러므로 Train data에서 적절한 정책을 통해 1차 모델에서 reconstruction error가 크게 나오는 구간을 선별하여 2차적 모델의 Train data로 사용하는 방법을 진행해 보려고 한다.
- 2차적 모델도 실질적으로는 정상적 데이터만을 가지고 학습을 하기 때문에 비지도학습을 사용해야하고, 그렇기 때문에 의심데이터를 좀 더 세밀하게 재구현할 수 있는 2차적 모델도 Autoencoder를 채택하여 사용하려고 한다.



Reference

- Linux system call manual https://man7.org/linux/man-pages/man2/syscalls.2.html
- LID-DS github https://github.com/LID-DS