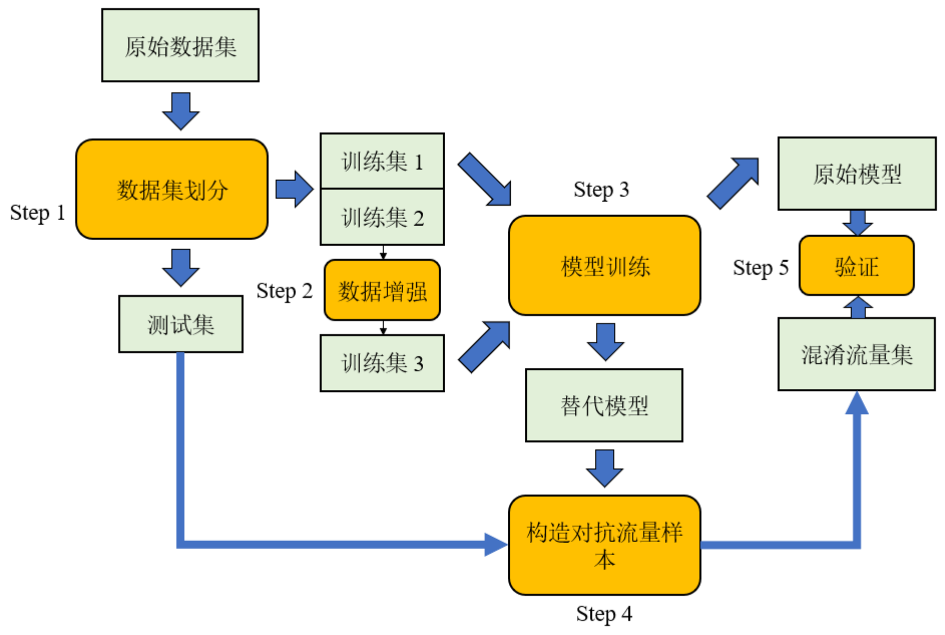
黑盒攻击实验流程如下：



Step 1

数据集划分：

- 训练集 ： 测试集 ＝ 9 ： 1
- 训练集 1 ： 训练集 2 ＝ 7 ： 3

获取原始数据：

```
# -*- coding: utf-8 -*-

import numpy as np
import pandas as pd
from sklearn.utils import shuffle
from sklearn.model_selection import train_test_split
from pandas import DataFrame

def getData(names):

    X = np.empty(shape=(0, 10))
    y = []
    rows_1 = [2000, 1225, 967, 2000, 2814, 2000, 2000]
    rows_2 = [1797, 208, 273, 1725, 598, 1559, 1484]

    for i, name in enumerate(names):

        if rows_1[i] != 0:
            data = pd.read_csv("/root/WorkPlace/ori_csv/ori_" + name + "_lenseq.csv", nrows=rows_1[i],
index_col=0)
            _X = data.values
            X = np.vstack((X, _X))
            y += [[1, 0]] * rows_1[i]

        if rows_2[i] != 0:
            data = pd.read_csv("/root/WorkPlace/tor_csv/tor_" + name + "_lenseq.csv", nrows=rows_2[i],
index_col=0)
            _X = data.values
            X = np.vstack((X, _X))
            y += [[0, 1]] * rows_2[i]

    X, y = shuffle(X, y)
    return X, y
```

```
names = ["browsing", "email", "chat", "streaming", "file", "voip", "p2p"]
X, y = getData(names)
```

首先划分原始数据集：

```
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=.1)
```

然后划分训练集：

```
X_train_1, X_train_2, y_train_1, y_train_2 = train_test_split(X_train, y_train, test_size=.3)
```

保存为 .csv 文件：

```
pd_X_train_1 = DataFrame(X_train_1)
pd_X_train_2 = DataFrame(X_train_2)
pd_X_test = DataFrame(X_test)
pd_y_train_1 = DataFrame(y_train_1)
pd_y_train_2 = DataFrame(y_train_2)
pd_y_test = DataFrame(y_test)

pd_X_train_1.to_csv("/root/WorkPlace/black_csv/X_train_1.csv")
pd_X_train_2.to_csv("/root/WorkPlace/black_csv/X_train_2.csv")
pd_X_test.to_csv("/root/WorkPlace/black_csv/X_test.csv")
pd_y_train_1.to_csv("/root/WorkPlace/black_csv/y_train_1.csv")
pd_y_train_2.to_csv("/root/WorkPlace/black_csv/y_train_2.csv")
pd_y_test.to_csv("/root/WorkPlace/black_csv/y_test.csv")
```

Step 2

采用 SMOTE 方法进行数据增强：

```
from imblearn.over_sampling import SMOTE

model_smote = SMOTE()

X_train_2 = pd.read_csv("/root/WorkPlace/black_csv/X_train_2.csv", index_col=0).values
y_train_2 = pd.read_csv("/root/WorkPlace/black_csv/y_train_2.csv", index_col=0).values

X_train_3, y_train_3 = model_smote.fit_sample(X_train_2, y_train_2[...,0])
y_train_3 = np.concatenate((y_train_3.reshape(len(y_train_3),1), np.subtract(np.ones(len(y_train_3)),
y_train_3).reshape(len(y_train_3),1)), axis=1)
```

该过程可以在训练替代模型前进行。

Step 3

采用数据集 1 分别训练得到 DNN、CNN 和 LSTM 模型，并保存为 .h5 文件。

DNN：

```
import numpy as np
import pandas as pd
import tensorflow as tf

X_train = pd.read_csv("/root/WorkPlace/black_csv/X_train_1.csv", index_col=0).values
X_test = pd.read_csv("/root/WorkPlace/black_csv/X_test.csv", index_col=0).values
y_train = pd.read_csv("/root/WorkPlace/black_csv/y_train_1.csv", index_col=0).values
y_test = pd.read_csv("/root/WorkPlace/black_csv/y_test.csv", index_col=0).values

X_train = np.asarray(X_train).astype("float32")
X_test = np.asarray(X_test).astype("float32")
y_train = np.asarray(y_train).astype("float32")
y_test = np.asarray(y_test).astype("float32")
```

```
model = tf.keras.Sequential()

model.add(tf.keras.layers.Dense(128, activation='relu', input_dim=10))
model.add(tf.keras.layers.Dense(128, activation='relu'))
model.add(tf.keras.layers.Dense(128, activation='relu'))
model.add(tf.keras.layers.Dense(2, activation='softmax'))

model.compile(optimizer="adam",
              loss="binary_crossentropy",
              metrics=["acc"])
model.summary()

history = model.fit(X_train, y_train, epochs=100, batch_size=20, validation_split=0.2)
loss_and_metrics = model.evaluate(X_test, y_test)
print('\n%s: %.2f%%' % (model.metrics_names[1], loss_and_metrics[1]*100))

model.save("/root/WorkPlace/model/dnn_black.h5")
```

准确率为 94.67%。

CNN：

```
import numpy as np
import pandas as pd
import tensorflow as tf

X_train = pd.read_csv("/root/WorkPlace/black_csv/X_train_1.csv", index_col=0).values
X_test = pd.read_csv("/root/WorkPlace/black_csv/X_test.csv", index_col=0).values
y_train = pd.read_csv("/root/WorkPlace/black_csv/y_train_1.csv", index_col=0).values
y_test = pd.read_csv("/root/WorkPlace/black_csv/y_test.csv", index_col=0).values

X_train = tf.reshape(X_train, (X_train.shape[0], 10, 1))
X_test = tf.reshape(X_test, (X_test.shape[0], 10, 1))
y_train = np.asarray(y_train).astype("float32")
y_test = np.asarray(y_test).astype("float32")

model = tf.keras.Sequential()

model.add(tf.keras.layers.Conv1D(100, 2, padding='valid', activation='relu', input_shape=(10,1)))
model.add(tf.keras.layers.MaxPool1D(pool_size=2))
model.add(tf.keras.layers.Conv1D(100, 2, padding='valid', activation='relu'))
model.add(tf.keras.layers.MaxPool1D(pool_size=2))
model.add(tf.keras.layers.Dropout(0.25))
model.add(tf.keras.layers.Flatten())
model.add(tf.keras.layers.Dropout(0.5))
model.add(tf.keras.layers.Dense(100, activation='relu'))
model.add(tf.keras.layers.Dense(2, activation='softmax'))

model.compile(optimizer="adam",
              loss="binary_crossentropy",
              metrics=["acc"])
model.summary()

history = model.fit(X_train, y_train, epochs=100, batch_size=20, validation_split=0.2)
loss_and_metrics = model.evaluate(X_test, y_test)
print('\n%s: %.2f%%' % (model.metrics_names[1], loss_and_metrics[1]*100))

model.save("/root/WorkPlace/model/cnn_black.h5")
```

准确率为 87.70%。

LSTM：

```
import numpy as np
import pandas as pd
import tensorflow as tf

X_train = pd.read_csv("/root/WorkPlace/black_csv/X_train_1.csv", index_col=0).values
X_test = pd.read_csv("/root/WorkPlace/black_csv/X_test.csv", index_col=0).values
```

```
y_train = pd.read_csv("/root/WorkPlace/black_csv/y_train_1.csv", index_col=0).values
y_test = pd.read_csv("/root/WorkPlace/black_csv/y_test.csv", index_col=0).values

X_train = tf.reshape(X_train, (X_train.shape[0], X_test.shape[1], 1))
X_test = tf.reshape(X_test, (X_test.shape[0], X_test.shape[1], 1))
y_train = np.asarray(y_train).astype("float32")
y_test = np.asarray(y_test).astype("float32")

model = tf.keras.Sequential()

model.add(tf.keras.layers.LSTM(128, activation='tanh', return_sequences=True, input_shape=(10,1)))
model.add(tf.keras.layers.LSTM(128, activation='tanh', return_sequences=True))
model.add(tf.keras.layers.LSTM(128, activation='tanh'))
model.add(tf.keras.layers.Dense(2, activation='softmax'))

model.compile(optimizer="rmsprop",
              loss="binary_crossentropy",
              metrics=["acc"])
model.summary()

history = model.fit(X_train, y_train, epochs=100, batch_size=20, validation_split=0.2)
loss_and_metrics = model.evaluate(X_test, y_test)
print('\n%s: %.2f%%' % (model.metrics_names[1], loss_and_metrics[1]*100))

model.save("/root/WorkPlace/model/lstm_black.h5")
```

准确率为 97.29%。

另外，采用训练集 1 训练 KNN 和 随机森林两种机器学习模型，该过程可以在 Step 5 进行。

KNN：

```
import pandas as pd
from sklearn.neighbors import KNeighborsClassifier

X_train = pd.read_csv("/root/WorkPlace/black_csv/X_train_1.csv", index_col=0).values
X_test = pd.read_csv("/root/WorkPlace/black_csv/X_test.csv", index_col=0).values
y_train = pd.read_csv("/root/WorkPlace/black_csv/y_train_1.csv", index_col=0).values
y_test = pd.read_csv("/root/WorkPlace/black_csv/y_test.csv", index_col=0).values

model_knn = KNeighborsClassifier().fit(X_train, y_train)
```

随机森林：

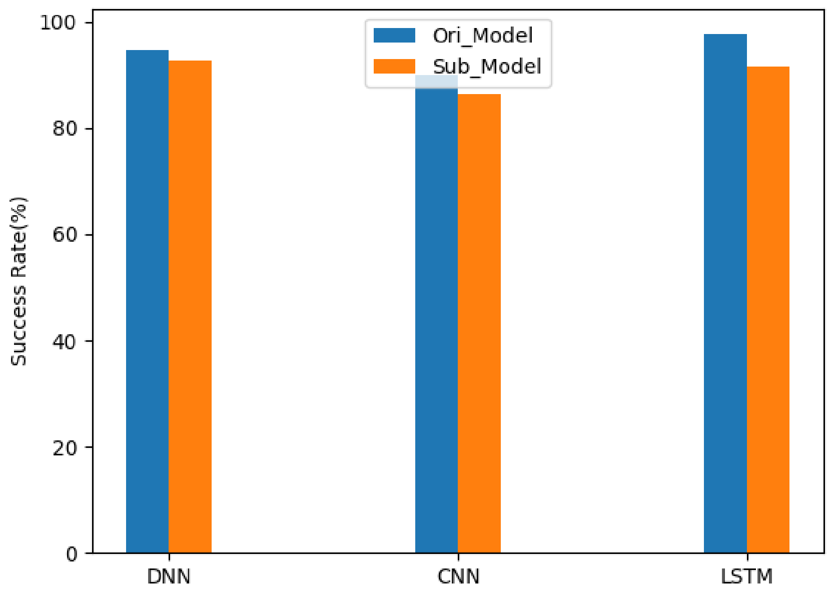
```
import pandas as pd
from sklearn.ensemble import RandomForestClassifier

X_train = pd.read_csv("/root/WorkPlace/black_csv/X_train_1.csv", index_col=0).values
X_test = pd.read_csv("/root/WorkPlace/black_csv/X_test.csv", index_col=0).values
y_train = pd.read_csv("/root/WorkPlace/black_csv/y_train_1.csv", index_col=0).values
y_test = pd.read_csv("/root/WorkPlace/black_csv/y_test.csv", index_col=0).values

model_randomforest = RandomForestClassifier(n_estimators=10, random_state=0).fit(X_train, y_train)
```

原始模型训练完成后，分别选择 DNN、CNN、 LSTM 作为替代模型，采用训练集 3 进行训练：

```
X_train = X_train_3
X_test = pd.read_csv("/root/WorkPlace/black_csv/X_test.csv", index_col=0).values
y_train = y_train_3
y_test = pd.read_csv("/root/WorkPlace/black_csv/y_test.csv", index_col=0).values

X_train_dnn = np.asarray(X_train).astype("float32")
X_test_dnn = np.asarray(X_test).astype("float32")
X_train_cnn = tf.reshape(X_train, (X_train.shape[0], 10, 1))
X_test_cnn = tf.reshape(X_test, (X_test.shape[0], 10, 1))
X_train_lstm = tf.reshape(X_train, (X_train.shape[0], X_test.shape[1], 1))
X_test_lstm = tf.reshape(X_test, (X_test.shape[0], X_test.shape[1], 1))
y_train = np.asarray(y_train).astype("float32")
```

```
y_test = np.asarray(y_test).astype("float32")

...

model_dnn.save("/root/WorkPlace/model/dnn_substitute.h5")
model_cnn.save("/root/WorkPlace/model/cnn_substitute.h5")
model_lstm.save("/root/WorkPlace/model/lstm_substitute.h5")
```

DNN 准确率为 92.74%，CNN 准确率为 86.34%，LSTM 准确率为 91.53%。



Step 4

读取测试集和替代模型 (LSTM)：

```
X_test = pd.read_csv("E://WorkPlace/black_csv/X_test.csv", index_col=0).values.astype("float32")
X_test_lstm = tf.reshape(X_test, (X_test.shape[0], X_test.shape[1], 1))

model = load_model("E://WorkPlace/model/lstm_substitute.h5")
logits_model = Model(model.input, model.layers[-1].output)
```

使用 MIM 方法构造对抗流量样本，将混淆流量集保存为 .csv 文件：

```
mim_params = {"norm": np.inf, "eps": 25, "eps_iter": 1, "nb_iter": 1000, "clip_min": 0.0, "clip_max":
200.0, "sanity_checks": False}
adv_x = momentum_iterative_method(logits_model, X_test_lstm, **mim_params)
adv_pred = model.predict(adv_x)

pd_adv_x = DataFrame(tf.reshape(adv_x, (X_test.shape[0], X_test.shape[1])).numpy())
pd_adv_x.to_csv("E://WorkPlace/black_csv/adv_lstm_mim.csv")

pred = model.predict(X_test_lstm)
count = 0
total = 0
for i in range(len(X_test)):
    if pred[i][0] < 0.5:
        total += 1
        if adv_pred[i][0] > 0.5:
            count += 1
print("Success rate of MIM: " + str(round(count/(total*0.01), 2)) + "%")
```

替代模型 (LSTM) 的白盒攻击 (MIM) 成功率为 97.98%。

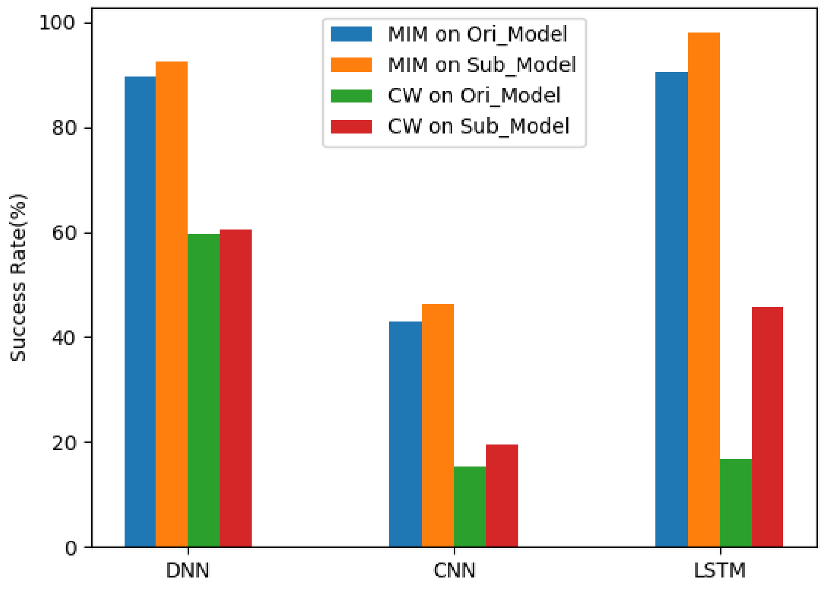
使用 CW 方法构造对抗流量样本，将混淆流量集保存为 .csv 文件：

```
cw_params = {"clip_min": 0.0, "clip_max": 200.0}
adv_x = carlini_wagner_l2(logits_model, X_test_lstm, **cw_params)
adv_pred = model.predict(adv_x)

pd_adv_x = DataFrame(tf.reshape(adv_x, (X_test.shape[0], X_test.shape[1])).numpy())
pd_adv_x.to_csv("E://WorkPlace/black_csv/adv_lstm_cw.csv")

pred = model.predict(X_test_lstm)
count = 0
total = 0
for i in range(len(X_test)):
    if pred[i][0] < 0.5:
        total += 1
        if adv_pred[i][0] > 0.5:
            count += 1
print("Success rate of CW: " + str(round(count/(total*0.01), 2)) + "%")
```

替代模型 (LSTM) 的白盒攻击 (CW) 成功率为 45.63%。

使用 DNN、CNN 作为替代模型的白盒攻击成功率分别为：

- DNN + MIM：92.45%
- DNN + CW：60.68%
- CNN + MIM：46.28%
- CNN + CW：19.56%



Step 5

读取 LSTM 生成的混淆流量集：

```
import pandas as pd

adv_mim = pd.read_csv("E://WorkPlace/black_csv/adv_lstm_mim.csv", index_col=0).values.astype("float32")
adv_cw = pd.read_csv("E://WorkPlace/black_csv/adv_lstm_cw.csv", index_col=0).values.astype("float32")
```

分别使用 5 种原始模型进行测试，计算黑盒攻击的成功率：

```
from tensorflow.keras.models import load_model
from sklearn.neighbors import KNeighborsClassifier
from sklearn.ensemble import RandomForestClassifier
import tensorflow as tf

X_train = pd.read_csv("E://WorkPlace/black_csv/X_train_1.csv", index_col=0).values
y_train = pd.read_csv("E://WorkPlace/black_csv/y_train_1.csv", index_col=0).values

model_knn = KNeighborsClassifier().fit(X_train, y_train)
model_randomforest = RandomForestClassifier(n_estimators=10, random_state=0).fit(X_train, y_train)
model_dnn = load_model("E://WorkPlace/model/dnn_black.h5")
model_cnn = load_model("E://WorkPlace/model/cnn_black.h5")
model_lstm = load_model("E://WorkPlace/model/lstm_black.h5")

X_test = pd.read_csv("E://WorkPlace/black_csv/X_test.csv", index_col=0).values.astype("float32")

# KNN

pred = model_knn.predict(X_test)
adv_pred = model_knn.predict(adv_mim)
count = 0
total = 0
for i in range(len(X_test)):
    if pred[i][0] < 0.5:
        total += 1
        if adv_pred[i][0] > 0.5:
            count += 1
print("Success rate of MIM on KNN: " + str(round(count/(total*0.01), 2)) + "%")

adv_pred = model_knn.predict(adv_cw)
count = 0
total = 0
for i in range(len(X_test)):
    if pred[i][0] < 0.5:
        total += 1
        if adv_pred[i][0] > 0.5:
            count += 1
print("Success rate of CW on KNN: " + str(round(count/(total*0.01), 2)) + "%")

# Random Forest

pred = model_randomforest.predict(X_test)
adv_pred = model_randomforest.predict(adv_mim)
count = 0
total = 0
for i in range(len(X_test)):
    if pred[i][0] < 0.5:
        total += 1
        if adv_pred[i][0] > 0.5:
            count += 1
print("Success rate of MIM on Random Forest: " + str(round(count/(total*0.01), 2)) + "%")

adv_pred = model_randomforest.predict(adv_cw)
count = 0
total = 0
for i in range(len(X_test)):
    if pred[i][0] < 0.5:
        total += 1
        if adv_pred[i][0] > 0.5:
            count += 1
print("Success rate of CW on Random Forest: " + str(round(count/(total*0.01), 2)) + "%")

# DNN

pred = model_dnn.predict(X_test)
adv_pred = model_dnn.predict(adv_mim)
count = 0
total = 0
for i in range(len(X_test)):
    if pred[i][0] < 0.5:
        total += 1
        if adv_pred[i][0] > 0.5:
            count += 1
print("Success rate of MIM on DNN: " + str(round(count/(total*0.01), 2)) + "%")
```

```
adv_pred = model_dnn.predict(adv_cw)
count = 0
total = 0
for i in range(len(X_test)):
    if pred[i][0] < 0.5:
        total += 1
        if adv_pred[i][0] > 0.5:
            count += 1
print("Success rate of CW on DNN: " + str(round(count/(total*0.01), 2)) + "%")

# CNN

pred = model_cnn.predict(tf.reshape(X_test, (X_test.shape[0], 10, 1)))
adv_mim_ = tf.reshape(adv_mim, (adv_mim.shape[0], 10, 1))
adv_pred = model_cnn.predict(adv_mim_)
count = 0
total = 0
for i in range(len(X_test)):
    if pred[i][0] < 0.5:
        total += 1
        if adv_pred[i][0] > 0.5:
            count += 1
print("Success rate of MIM on CNN: " + str(round(count/(total*0.01), 2)) + "%")

adv_cw_ = tf.reshape(adv_cw, (adv_cw.shape[0], 10, 1))
adv_pred = model_cnn.predict(adv_cw_)
count = 0
total = 0
for i in range(len(X_test)):
    if pred[i][0] < 0.5:
        total += 1
        if adv_pred[i][0] > 0.5:
            count += 1
print("Success rate of CW on CNN: " + str(round(count/(total*0.01), 2)) + "%")

# LSTM

pred = model_lstm.predict(tf.reshape(X_test, (X_test.shape[0], X_test.shape[1], 1)))
adv_mim_ = tf.reshape(adv_mim, (adv_mim.shape[0], adv_mim.shape[1], 1))
adv_pred = model_lstm.predict(adv_mim_)
count = 0
total = 0
for i in range(len(X_test)):
    if pred[i][0] < 0.5:
        total += 1
        if adv_pred[i][0] > 0.5:
            count += 1
print("Success rate of MIM on LSTM: " + str(round(count/(total*0.01), 2)) + "%")

adv_cw_ = tf.reshape(adv_cw, (adv_cw.shape[0], adv_cw.shape[1], 1))
adv_pred = model_lstm.predict(adv_cw_)
count = 0
total = 0
for i in range(len(X_test)):
    if pred[i][0] < 0.5:
        total += 1
        if adv_pred[i][0] > 0.5:
            count += 1
print("Success rate of CW on LSTM: " + str(round(count/(total*0.01), 2)) + "%")
```

使用 DNN、CNN、LSTM 作为替代模型的白盒攻击成功率分别为：

- DNN+MIM vs. KNN：15.98%
- DNN+CW vs. KNN：1.5%
- CNN+MIM vs. KNN：33.83%
- CNN+CW vs. KNN：1.37%
- **LSTM+MIM vs. KNN：34.46%**
- LSTM+CW vs. KNN：3.62%
- DNN+MIM vs. Random Forest：33.9%
- DNN+CW vs. Random Forest：11.69%
- CNN+MIM vs. Random Forest：58.18%

- CNN+CW vs. Random Forest：11.56%
- **LSTM+MIM vs. Random Forest：66.49%**
- LSTM+CW vs. Random Forest：11.95%
- **DNN+MIM vs. DNN：47.2%**
- DNN+CW vs. DNN：7.1%
- CNN+MIM vs. DNN：17.68%
- CNN+CW vs. DNN：2.37%
- LSTM+MIM vs. DNN：18.31%
- LSTM+CW vs. DNN：1.62%
- DNN+MIM vs. CNN：14.73%
- DNN+CW vs. CNN：3.5%
- **CNN+MIM vs. CNN：42.12%**
- CNN+CW vs. CNN：1.75%
- LSTM+MIM vs. CNN：16.27%
- LSTM+CW vs. CNN：0.21%
- DNN+MIM vs. LSTM：67.39%
- DNN+CW vs. LSTM：6.88%
- CNN+MIM vs. LSTM：65.1%
- CNN+CW vs. LSTM：6.62%
- **LSTM+MIM vs. LSTM：73.63%**
- LSTM+CW vs. LSTM：2.17%

时间序列模型 vs. 转移概率矩阵模型

均使用长度为 50 的序列，分别构建原始数据集，特征变换函数如下：

```
def getTranMatrix(flows):

    data = []
    numRows = 10
    tranMat = np.zeros((numRows,numRows))

    for flow in flows:
        for i in range(1, len(flow)):
            prevPacketSize = min(int(flow[i-1]), numRows-1)
            curPacketSize = min(int(flow[i]), numRows-1)
            tranMat[prevPacketSize,curPacketSize] += 1
            for i in range(numRows):
                if float(np.sum(tranMat[i:i+1])) != 0:
                    tranMat[i:i+1] = tranMat[i:i+1]/float(np.sum(tranMat[i:i+1]))

        data.append(list(tranMat.flatten()))

    return data
```

分别训练时间序列模型和转移概率矩阵模型，模型包括：

- 原始模型：
  - knn_tran
  - rf_tran
  - svm_tran
  - dnn_tran
  - cnn_tran
  - lstm_tran
- 替代模型：
  - lstm_seq

使用 MIM 方法创建对抗样本集 A（时间序列），并将对抗样本集 A 转换为对抗样本集 B（转移概率矩阵）：

```
X_test = pd.read_csv("E://WorkPlace/black_plus_csv/X_test.csv", index_col=0).values.astype("float32")
X_test = tf.reshape(X_test, (X_test.shape[0], X_test.shape[1], 1))
```

```
model = load_model("E://WorkPlace/model/lstm_seq.h5")
logits_model = Model(model.input, model.layers[-1].output)

mim_params = {"norm": np.inf, "eps": 25, "eps_iter": 1, "nb_iter": 1000, "clip_min": 0.0, "clip_max":
200.0, "sanity_checks": False}
adv_x = momentum_iterative_method(logits_model, X_test, **mim_params)

adv_x = tf.reshape(adv_x, (X_test.shape[0], X_test.shape[1])).numpy()
adv_x_tran = getTranMatrix(adv_x)

pd_adv_x = DataFrame(adv_x_tran)
pd_adv_x.to_csv("E://WorkPlace/black_plus_csv/adv_lstm_mim_tran.csv")
```

验证：

```
adv_mim = pd.read_csv("E://WorkPlace/black_plus_csv/adv_lstm_mim_tran.csv",
index_col=0).values.astype("float32")

X_train = pd.read_csv("/root/WorkPlace/black_plus_csv/X_train_1_.csv",
index_col=0).values.astype("float32")
y_train = pd.read_csv("/root/WorkPlace/black_plus_csv/y_train_1_.csv",
index_col=0).values.astype("float32")

model_knn = KNeighborsClassifier().fit(X_train, y_train)
model_randomforest = RandomForestClassifier(n_estimators=10, random_state=0).fit(X_train, y_train)
model_dnn = load_model("E://WorkPlace/model/dnn_tran.h5")
model_cnn = load_model("E://WorkPlace/model/cnn_tran.h5")
model_lstm = load_model("E://WorkPlace/model/lstm_tran.h5")

X_test = pd.read_csv("E://WorkPlace/black_plus_csv/X_test.csv", index_col=0).values.astype("float32")

# KNN

pred = model_knn.predict(X_test)
adv_pred = model_knn.predict(adv_mim)
count = 0
total = 0
for i in range(len(X_test)):
    if pred[i][0] < 0.5:
        total += 1
        if adv_pred[i][0] > 0.5:
            count += 1
print("Success rate of MIM on KNN: " + str(round(count/(total*0.01), 2)) + "%")

# Random Forest

pred = model_randomforest.predict(X_test)
adv_pred = model_randomforest.predict(adv_mim)
count = 0
total = 0
for i in range(len(X_test)):
    if pred[i][0] < 0.5:
        total += 1
        if adv_pred[i][0] > 0.5:
            count += 1
print("Success rate of MIM on Random Forest: " + str(round(count/(total*0.01), 2)) + "%")

# DNN

pred = model_dnn.predict(X_test)
adv_pred = model_dnn.predict(adv_mim)
count = 0
total = 0
for i in range(len(X_test)):
    if pred[i][0] < 0.5:
        total += 1
        if adv_pred[i][0] > 0.5:
            count += 1
print("Success rate of MIM on DNN: " + str(round(count/(total*0.01), 2)) + "%")

# CNN
```

```
pred = model_cnn.predict(tf.reshape(X_test, (X_test.shape[0], 50, 1)))
adv_mim_ = tf.reshape(adv_mim, (adv_mim.shape[0], 100, 1))
adv_pred = model_cnn.predict(adv_mim_)
count = 0
total = 0
for i in range(len(X_test)):
    if pred[i][0] < 0.5:
        total += 1
        if adv_pred[i][0] > 0.5:
            count += 1
print("Success rate of MIM on CNN: " + str(round(count/(total*0.01), 2)) + "%")

# LSTM

pred = model_lstm.predict(tf.reshape(X_test, (X_test.shape[0], X_test.shape[1], 1)))
adv_mim_ = tf.reshape(adv_mim, (adv_mim.shape[0], adv_mim.shape[1], 1))
adv_pred = model_lstm.predict(adv_mim_)
count = 0
total = 0
for i in range(len(X_test)):
    if pred[i][0] < 0.5:
        total += 1
        if adv_pred[i][0] > 0.5:
            count += 1
print("Success rate of MIM on LSTM: " + str(round(count/(total*0.01), 2)) + "%")
```

成功率为：

- KNN：79.81%
- Random Forest：61.89%
- SVM：43.4%
- DNN：58.57%
- CNN：48.4%
- LSTM：0%（没有预测为负的样本）

攻击成功率更高，可能是因为使用序列长度为 50 的转移概率矩阵模型本身性能太差。