Bypassing Detection of URL-based Phishing Attacks Using GenerativeAdversarial Deep Neural Networks

AYUSH DHIMAN, 19BCE1463

ESHAN MADNANI, 19BCE1470

In [80]:

```
import pandas as pd
import numpy as np
```

In [81]:

```
df = pd.read_csv('./Training-Dataset.csv')
df.head()
```

Out[81]:

	having_IP_Address	URL_Length	Shortining_Service	having_At_Symbol	double_slash_redirecting	Prefix_:
0	-1	1	1	1	-1	
1	1	1	1	1	1	
2	1	0	1	1	1	
3	1	0	1	1	1	
4	1	0	-1	1	1	

5 rows × 31 columns

In [82]:

```
df.shape
```

Out[82]:

```
(11055, 31)
```

PHISING URL's WITH -1 RESULT LABEL

In [32]:

```
df1=df[df['Result']==-1]
df1
```

Out[32]:

	having_IP_Address	URL_Length	Shortining_Service	having_At_Symbol	double_slash_redirecting	Pre
0	-1	1	1	1	-1	
1	1	1	1	1	1	
2	1	0	1	1	1	
3	1	0	1	1	1	
6	1	0	-1	1	1	
...	...	...	...	...	...	
11049	-1	-1	1	1	-1	
11051	-1	1	1	-1	-1	
11052	1	-1	1	1	1	
11053	-1	-1	1	1	1	

	having_IP_Address	URL_Length	Shortining_Service	having_At_Symbol	double_slash_redirecting	Pre
11054	-1	-1	1	1	1	

4898 rows × 31 columns

LEGITIMATE URL's WITH 1 RESULT LABEL

In [33]:

```
df2=df[df['Result']==1]
df2
```

Out[33]:

	having_IP_Address	URL_Length	Shortining_Service	having_At_Symbol	double_slash_redirecting	Pre
4	1	0	-1	1	1	
5	-1	0	-1	1	-1	
8	1	0	-1	1	1	
10	1	1	1	1	1	
14	1	1	-1	1	1	
...	...	...	...	...	...	
11044	-1	-1	-1	1	-1	
11045	1	-1	1	1	1	
11046	-1	-1	1	1	1	
11048	1	-1	1	1	1	
11050	1	-1	1	-1	1	

6157 rows × 31 columns

SUSPICIOUS URL's WITH 0 RESULT LABEL

In [34]:

```
df3=df[df['Result']==0]
df3
```

Out[34]:

having_IP_Address	URL_Length	Shortining_Service	having_At_Symbol	double_slash_redirecting	Prefix_Su

0 rows × 31 columns

ENCODING OUR URL DATASET

In [35]:

```
"""
df['having_IP_Address'] = df['having_IP_Address'].astype(str)
for i in df.index:
    if df.at[i, "having_IP_Address"] == '-1':
        df.at[i, "having_IP_Address"] = '11'
    elif df.at[i, "having_IP_Address"] == '1':
        df.at[i, "having_IP_Address"] = '00'
    elif df.at[i, "having_IP_Address"] == '0':
        df.at[i, "having_IP_Address"] = '01'
df['having_IP_Address_1'] = (df['having_IP_Address'].str[0]).astype(int)
df['having_IP_Address_2'] = (df['having_IP_Address'].str[1]).astype(int)
df.drop('having_IP_Address', axis=1, inplace=True)
"""
```

```
# The above code will require us to run the code 30 times for 30 different columns which

# We Improvised the above code

edf = df
def encode(x):
    if x == '-1':
        return '11'
    elif x == '1':
        return '00'
    elif x == '0':
        return '01'

count = 0

for column in edf:
    count+=1
    if count == 31:
        break
    edf[column] = edf[column].astype(str)
    edf[column] = edf[column].apply(encode)

    edf[column + '1'] = (edf[column].str[0]).astype(int)
    edf[column + '2'] = (edf[column].str[1]).astype(int)
    edf.drop(column, axis=1, inplace=True)

col = edf.pop('Result')
edf.insert(60,'Result',col)

for i in edf:
    print(i)
```

```
having_IP_Address1
having_IP_Address2
URL_Length1
URL_Length2
Shortining_Service1
Shortining_Service2
having_At_Symbol1
having_At_Symbol2
double_slash_redirecting1
double_slash_redirecting2
Prefix_Suffix1
Prefix_Suffix2
having_Sub_Domain1
having_Sub_Domain2
SSLfinal_State1
SSLfinal_State2
Domain_registeration_length1
Domain_registeration_length2
Favicon1
Favicon2
port1
port2
HTTPS_token1
HTTPS_token2
Request_URL1
Request_URL2
URL_of_Anchor1
URL_of_Anchor2
Links_in_tags1
Links_in_tags2
SFH1
SFH2
Submitting_to_email1
```

```
Submitting_to_email2
Abnormal_URL1
Abnormal_URL2
Redirect1
Redirect2
on_mouseover1
on_mouseover2
RightClick1
RightClick2
popUpWidnow1
popUpWidnow2
Iframe1
Iframe2
age_of_domain1
age_of_domain2
DNSRecord1
DNSRecord2
web_traffic1
web_traffic2
Page_Rank1
Page_Rank2
Google_Index1
Google_Index2
Links_pointing_to_page1
Links_pointing_to_page2
Statistical_report1
Statistical_report2
Result
```

In [36]:

```
edf.Result.value_counts()
```

Out[36]:

```
 1    6157
-1    4898
Name: Result, dtype: int64
```

Separating Malicious and Legitimate Values from the Dataset

In [37]:

```
df_mal = edf[edf['Result'] == -1 ]
df_leg = edf[edf['Result'] ==  1 ]
```

In [38]:

```
print("Number of Malacious Values : "  , df_mal.Result.value_counts())
print("Number of Legitimate Values : " , df_leg.Result.value_counts())
```

```
Number of Malacious Values :  -1    4898
Name: Result, dtype: int64
Number of Legitimate Values :  1    6157
Name: Result, dtype: int64
```

Trying to build GAN

In [40]:

```
from __future__ import print_function, division

from keras.layers import Input, Dense, Activation
from keras.layers.merge import Maximum, Concatenate
from keras.models import Model
from tensorflow.keras.optimizers import Adam

from sklearn.ensemble import RandomForestClassifier
from sklearn.neural_network import MLPClassifier
from sklearn.model_selection import train_test_split
import matplotlib.pyplot as plt
```

```python
import pandas as pd

import numpy as np
```

In [92]:
```python
class GAN():
    def __init__(self):
        self.urlfeature_dims = 60
        self.z_dims = 20    # could try 20,malware has to add url calls if you want to defi
                            # Z the larger the dry so
        self.hide_layers = 120
        self.generator_layers = [self.urlfeature_dims + self.z_dims, self.hide_layers, sel
        self.substitute_detector_layers = [self.urlfeature_dims, self.hide_layers, 1]
        self.blackbox = 'MLP'
        optimizer = Adam(lr=0.001)

        # Build and Train blackbox_detector
        self.blackbox_detector = self.build_blackbox_detector()

        # Build and compile the substitute_detector
        self.substitute_detector = self.build_substitute_detector()
        self.substitute_detector.compile(loss='binary_crossentropy', optimizer=optimizer,

        # Build the generator
        self.generator = self.build_generator()

        # The generator takes malware and noise as input and generates adversarial malware
        example = Input(shape=(self.urlfeature_dims,))
        noise = Input(shape=(self.z_dims,))
        input = [example, noise]
        malware_examples = self.generator(input)

        # For the combined model we will only train the generator
        self.substitute_detector.trainable = False

        # The discriminator takes generated URLs as input and determines validity
        validity = self.substitute_detector(malware_examples)

        # The combined model  (stacked generator and substitute_detector)
        # Trains the generator to fool the discriminator
        self.combined = Model(input, validity)
        self.combined.compile(loss='binary_crossentropy', optimizer=optimizer)

    def build_blackbox_detector(self):

        if self.blackbox == 'MLP':
            blackbox_detector = MLPClassifier(hidden_layer_sizes=(50,), max_iter=10, alpha
                                              solver='sgd', verbose=0, tol=1e-4, random_st
                                              learning_rate_init=.1)
        return blackbox_detector

    def build_generator(self):

        example = Input(shape=(self.urlfeature_dims,))
        noise = Input(shape=(self.z_dims,))
        x = Concatenate(axis=1)([example, noise])
        i =0
        for dim in self.generator_layers[1:]:
            x = Dense(dim)(x)
            x = Activation(activation='relu')(x)
        x = Maximum()([example, x])
        generator = Model([example, noise], x, name='generator')
        generator.summary()
        return generator
```

```
    def build_substitute_detector(self):

        input = Input(shape=(self.substitute_detector_layers[0],))
        x = input
        for dim in self.substitute_detector_layers[1:]:
            x = Dense(dim)(x)
            x = Activation(activation='relu')(x)
        substitute_detector = Model(input, x, name='substitute_detector')
        substitute_detector.summary()
        return substitute_detector

    def load_data(self, filename):

        df_xmal = df_mal.iloc[:,:-1]
        df_ymal = df_mal.iloc[:,60:61]
        df_xben = df_leg.iloc[:,:-1]
        df_yben = df_leg.iloc[:,60:61]

        xmal = df_xmal.to_numpy()
        ymal = df_ymal.to_numpy()
        xben = df_xben.to_numpy()
        yben = df_yben.to_numpy()

        return (xmal, ymal), (xben, yben)

    def train(self, epochs, batch_size):

        # Load the dataset
        (xmal, ymal), (xben, yben) = self.load_data('')
        xtrain_mal, xtest_mal, ytrain_mal, ytest_mal = train_test_split(xmal, ymal, test_s
        xtrain_ben, xtest_ben, ytrain_ben, ytest_ben = train_test_split(xben, yben, test_s

        # Train blackbox_detctor
        self.blackbox_detector.fit(np.concatenate([xmal, xben]),
                                   np.concatenate([ymal, yben]))

        ytrain_ben_blackbox = self.blackbox_detector.predict(xtrain_ben)
        Original_Train_TRR = self.blackbox_detector.score(xtrain_mal, ytrain_mal)
        Original_Test_TRR = self.blackbox_detector.score(xtest_mal, ytest_mal)
        Train_TRR, Test_TRR = [], []

        for epoch in range(epochs):

            for step in range(1):#range(xtrain_mal.shape[0] // batch_size):
                # ---------------------
                #  Train substitute_detector
                # ---------------------

                # Select a random batch of malware examples
                idx = np.random.randint(0, xtrain_mal.shape[0], batch_size)
                xmal_batch = xtrain_mal[idx]
                noise = np.random.uniform(0, 1, (batch_size, self.z_dims))
                idx = np.random.randint(0, xmal_batch.shape[0], batch_size)
                xben_batch = xtrain_ben[idx]
                yben_batch = ytrain_ben_blackbox[idx]

                # Generate a batch of new malware examples
                gen_examples = self.generator.predict([xmal_batch, noise])
                ymal_batch = self.blackbox_detector.predict(np.ones(gen_examples.shape)*(g

                # Train the substitute_detector
                d_loss_real = self.substitute_detector.train_on_batch(gen_examples, ymal_b
                d_loss_fake = self.substitute_detector.train_on_batch(xben_batch, yben_bat
                d_loss = 0.5 * np.add(d_loss_real, d_loss_fake)

                # ---------------------
```

```
                #  Train Generator
                # ---------------------

                idx = np.random.randint(0, xtrain_mal.shape[0], batch_size)
                xmal_batch = xtrain_mal[idx]
                noise = np.random.uniform(0, 1, (batch_size, self.z_dims))

                # Train the generator
                g_loss = self.combined.train_on_batch([xmal_batch, noise], np.zeros((batch

            # Compute Train TRR
            noise = np.random.uniform(0, 1, (xtrain_mal.shape[0], self.z_dims))
            gen_examples = self.generator.predict([xtrain_mal, noise])
            TRR = self.blackbox_detector.score(np.ones(gen_examples.shape) * (gen_examples
            Train_TRR.append(TRR)

            # Compute Test TRR
            noise = np.random.uniform(0, 1, (xtest_mal.shape[0], self.z_dims))
            gen_examples = self.generator.predict([xtest_mal, noise])
            TRR = self.blackbox_detector.score(np.ones(gen_examples.shape) * (gen_examples
            Test_TRR.append(TRR)

            # Plot the progress
            print("%d [D loss: %f, acc.: %.2f%%] [G loss: %f]" % (epoch, d_loss[0], 100*d_

        print('Original_Train_TRR: {0}, Adver_Train_TRR: {1}'.format(Original_Train_TRR, T
        print('Original_Test_TRR: {0}, Adver_Test_TRR: {1}'.format(Original_Test_TRR, Test

        # Plot TRR
        plt.figure()
        plt.plot(range(epochs), Train_TRR, c='r', label='Training Set', linewidth=2)
        plt.plot(range(epochs), Test_TRR, c='g', linestyle='--', label='Validation Set', l
        plt.xlabel("Epoch")
        plt.ylabel("TRR")
        plt.legend()
        plt.show()
```

Dataset used in Research Paper

In [93]:

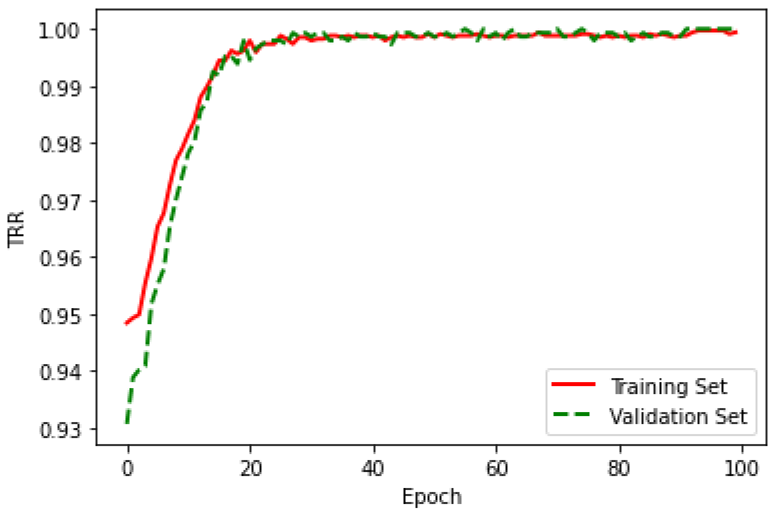
```
gan = GAN()
gan.train(epochs=100, batch_size=60)
```

```
/usr/local/lib/python3.9/site-packages/keras/optimizer_v2/optimizer_v2.py:355: UserWarnin
g: The `lr` argument is deprecated, use `learning_rate` instead.
  warnings.warn(
/usr/local/lib/python3.9/site-packages/sklearn/utils/validation.py:63: DataConversionWarni
ng: A column-vector y was passed when a 1d array was expected. Please change the shape of
y to (n_samples, ), for example using ravel().
  return f(*args, **kwargs)
Model: "substitute_detector"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
input_166 (InputLayer)       [(None, 60)]              0
_________________________________________________________________
dense_132 (Dense)            (None, 120)               7320
_________________________________________________________________
activation_132 (Activation)  (None, 120)               0
_________________________________________________________________
dense_133 (Dense)            (None, 1)                 121
_________________________________________________________________
activation_133 (Activation)  (None, 1)                 0
=================================================================
```

```
Total params: 7,441
Trainable params: 7,441
Non-trainable params: 0
_________________________________________________________________
Model: "generator"
__________________________________________________________________________________________
________
Layer (type)                    Output Shape         Param #     Connected to

==========================================================================================
========
input_167 (InputLayer)          [(None, 60)]         0


__________________________________________________________________________________________
________
input_168 (InputLayer)          [(None, 20)]         0


__________________________________________________________________________________________
________
concatenate_33 (Concatenate)    (None, 80)           0           input_167[0][0]

                                                                 input_168[0][0]


__________________________________________________________________________________________
________
dense_134 (Dense)               (None, 120)          9720        concatenate_33[0][0]


__________________________________________________________________________________________
________
activation_134 (Activation)     (None, 120)          0           dense_134[0][0]


__________________________________________________________________________________________
________
dense_135 (Dense)               (None, 60)           7260        activation_134[0][0]


__________________________________________________________________________________________
________
activation_135 (Activation)     (None, 60)           0           dense_135[0][0]


__________________________________________________________________________________________
________
maximum_33 (Maximum)            (None, 60)           0           input_167[0][0]

                                                                 activation_135[0][0]

==========================================================================================
========
Total params: 16,980
Trainable params: 16,980
Non-trainable params: 0
__________________________________________________________________________________________
________
/usr/local/lib/python3.9/site-packages/sklearn/neural_network/_multilayer_perceptron.py:61
4: ConvergenceWarning: Stochastic Optimizer: Maximum iterations (10) reached and the optim
ization hasn't converged yet.
  warnings.warn(
0 [D loss: -3.826010, acc.: 0.00%] [G loss: 0.009283]
1 [D loss: -2.825183, acc.: 6.67%] [G loss: 0.008340]
2 [D loss: -4.865208, acc.: 15.83%] [G loss: 0.017597]
3 [D loss: -5.033371, acc.: 26.67%] [G loss: 0.051540]
4 [D loss: -4.646569, acc.: 25.00%] [G loss: 0.064792]
5 [D loss: -5.266895, acc.: 34.17%] [G loss: 0.042022]
6 [D loss: -6.137485, acc.: 31.67%] [G loss: 0.018964]
7 [D loss: -6.414992, acc.: 30.00%] [G loss: 0.004480]
8 [D loss: -7.411030, acc.: 32.50%] [G loss: 0.000000]
```

```
9 [D loss: -7.460519, acc.: 32.50%] [G loss: 0.000397]
10 [D loss: -7.565059, acc.: 39.17%] [G loss: 0.000000]
11 [D loss: -7.596383, acc.: 43.33%] [G loss: 0.000000]
12 [D loss: -6.503525, acc.: 43.33%] [G loss: 0.000000]
13 [D loss: -7.089465, acc.: 46.67%] [G loss: 0.000000]
14 [D loss: -7.519398, acc.: 43.33%] [G loss: 0.000000]
15 [D loss: -7.390031, acc.: 45.83%] [G loss: 0.000000]
16 [D loss: -7.604171, acc.: 45.00%] [G loss: 0.000000]
17 [D loss: -7.349665, acc.: 46.67%] [G loss: 0.000000]
18 [D loss: -7.612669, acc.: 47.50%] [G loss: 0.000000]
19 [D loss: -7.524066, acc.: 44.17%] [G loss: 0.000000]
20 [D loss: -7.672643, acc.: 48.33%] [G loss: 0.000000]
21 [D loss: -7.651368, acc.: 47.50%] [G loss: 0.000000]
22 [D loss: -7.653996, acc.: 49.17%] [G loss: 0.000000]
23 [D loss: -7.640510, acc.: 47.50%] [G loss: 0.000000]
24 [D loss: -7.641852, acc.: 48.33%] [G loss: 0.000000]
25 [D loss: -7.629498, acc.: 47.50%] [G loss: 0.000000]
26 [D loss: -7.664812, acc.: 49.17%] [G loss: 0.000000]
27 [D loss: -7.687308, acc.: 49.17%] [G loss: 0.000000]
28 [D loss: -7.674849, acc.: 45.83%] [G loss: 0.000000]
29 [D loss: -7.679146, acc.: 45.83%] [G loss: 0.000000]
30 [D loss: -7.691936, acc.: 49.17%] [G loss: 0.000000]
31 [D loss: -7.677589, acc.: 47.50%] [G loss: 0.000000]
32 [D loss: -7.683736, acc.: 50.00%] [G loss: 0.000000]
33 [D loss: -7.692602, acc.: 46.67%] [G loss: 0.000000]
34 [D loss: -7.666907, acc.: 49.17%] [G loss: 0.000000]
35 [D loss: -7.662211, acc.: 50.00%] [G loss: 0.000000]
36 [D loss: -7.674671, acc.: 47.50%] [G loss: 0.000000]
37 [D loss: -7.715312, acc.: 49.17%] [G loss: 0.000000]
38 [D loss: -7.672292, acc.: 49.17%] [G loss: 0.000000]
39 [D loss: -7.800968, acc.: 48.33%] [G loss: 0.000000]
40 [D loss: -7.679394, acc.: 50.00%] [G loss: 0.000000]
41 [D loss: -7.666254, acc.: 48.33%] [G loss: 0.000000]
42 [D loss: -7.817544, acc.: 48.33%] [G loss: 0.000000]
43 [D loss: -7.691709, acc.: 49.17%] [G loss: 0.000000]
44 [D loss: -7.826694, acc.: 48.33%] [G loss: 0.000000]
45 [D loss: -7.691092, acc.: 50.00%] [G loss: 0.000000]
46 [D loss: -7.930916, acc.: 47.50%] [G loss: 0.000000]
47 [D loss: -7.807697, acc.: 48.33%] [G loss: 0.000000]
48 [D loss: -7.938061, acc.: 46.67%] [G loss: 0.000000]
49 [D loss: -7.647470, acc.: 46.67%] [G loss: 0.000000]
50 [D loss: -7.696811, acc.: 50.00%] [G loss: 0.000000]
51 [D loss: -7.954729, acc.: 48.33%] [G loss: 0.000000]
52 [D loss: -7.817534, acc.: 48.33%] [G loss: 0.000000]
53 [D loss: -8.205403, acc.: 45.00%] [G loss: 0.000000]
54 [D loss: -7.706065, acc.: 50.00%] [G loss: 0.000000]
55 [D loss: -7.819134, acc.: 47.50%] [G loss: 0.000000]
56 [D loss: -7.699920, acc.: 49.17%] [G loss: 0.000000]
57 [D loss: -7.697052, acc.: 50.00%] [G loss: 0.000000]
58 [D loss: -7.826278, acc.: 48.33%] [G loss: 0.000000]
59 [D loss: -7.819420, acc.: 48.33%] [G loss: 0.000000]
60 [D loss: -7.704058, acc.: 50.00%] [G loss: 0.000000]
61 [D loss: -7.701469, acc.: 49.17%] [G loss: 0.000000]
62 [D loss: -7.695981, acc.: 50.00%] [G loss: 0.000000]
63 [D loss: -7.959141, acc.: 47.50%] [G loss: 0.000000]
64 [D loss: -7.698106, acc.: 48.33%] [G loss: 0.000000]
65 [D loss: -7.820521, acc.: 47.50%] [G loss: 0.000000]
66 [D loss: -7.831015, acc.: 48.33%] [G loss: 0.000000]
67 [D loss: -7.826322, acc.: 49.17%] [G loss: 0.000000]
68 [D loss: -8.090405, acc.: 46.67%] [G loss: 0.000000]
69 [D loss: -7.699824, acc.: 50.00%] [G loss: 0.000000]
70 [D loss: -7.706574, acc.: 49.17%] [G loss: 0.000000]
71 [D loss: -7.832272, acc.: 48.33%] [G loss: 0.000000]
72 [D loss: -7.960332, acc.: 47.50%] [G loss: 0.000000]
73 [D loss: -8.086976, acc.: 46.67%] [G loss: 0.000000]
74 [D loss: -7.711436, acc.: 50.00%] [G loss: 0.000000]
```

```
75 [D loss: -7.772388, acc.: 47.50%] [G loss: 0.000000]
76 [D loss: -7.834894, acc.: 48.33%] [G loss: 0.000000]
77 [D loss: -8.219784, acc.: 46.67%] [G loss: 0.000000]
78 [D loss: -7.823807, acc.: 49.17%] [G loss: 0.000000]
79 [D loss: -7.845628, acc.: 45.83%] [G loss: 0.000000]
80 [D loss: -7.694725, acc.: 49.17%] [G loss: 0.000000]
81 [D loss: -7.824977, acc.: 48.33%] [G loss: 0.000000]
82 [D loss: -7.982281, acc.: 45.00%] [G loss: 0.000000]
83 [D loss: -7.822961, acc.: 49.17%] [G loss: 0.000000]
84 [D loss: -7.835285, acc.: 47.50%] [G loss: 0.000000]
85 [D loss: -7.702292, acc.: 48.33%] [G loss: 0.000000]
86 [D loss: -7.580235, acc.: 47.50%] [G loss: 0.000000]
87 [D loss: -8.193836, acc.: 45.00%] [G loss: 0.000000]
88 [D loss: -7.714229, acc.: 47.50%] [G loss: 0.000000]
89 [D loss: -7.702556, acc.: 50.00%] [G loss: 0.000000]
90 [D loss: -7.841977, acc.: 48.33%] [G loss: 0.001145]
91 [D loss: -7.715112, acc.: 49.17%] [G loss: 0.000000]
92 [D loss: -7.839866, acc.: 49.17%] [G loss: 0.000000]
93 [D loss: -7.712330, acc.: 49.17%] [G loss: 0.002579]
94 [D loss: -7.699109, acc.: 50.00%] [G loss: 0.000000]
95 [D loss: -8.086882, acc.: 47.50%] [G loss: 0.000000]
96 [D loss: -7.718426, acc.: 49.17%] [G loss: 0.000000]
97 [D loss: -7.835585, acc.: 49.17%] [G loss: 0.000000]
98 [D loss: -8.101211, acc.: 46.67%] [G loss: 0.000000]
99 [D loss: -7.836003, acc.: 49.17%] [G loss: 0.000000]
Original_Train_TRR: 0.926487747957993, Adver_Train_TRR: 0.9994165694282381
Original_Test_TRR: 0.9136054421768708, Adver_Test_TRR: 1.0
```



New Dataset with double the features

In [96]:

```
class GAN():
    def __init__(self):
        self.urlfeature_dims = 128
        self.z_dims = 20    # could try 20,malware has to add url calls if you want to defi
                            # Z the larger the dry so
        self.hide_layers = 256
        self.generator_layers = [self.urlfeature_dims + self.z_dims, self.hide_layers, sel
        self.substitute_detector_layers = [self.urlfeature_dims, self.hide_layers, 1]
        self.blackbox = 'MLP'
        optimizer = Adam(lr=0.001)

        # Build and Train blackbox_detector
        self.blackbox_detector = self.build_blackbox_detector()

        # Build and compile the substitute_detector
        self.substitute_detector = self.build_substitute_detector()
        self.substitute_detector.compile(loss='binary_crossentropy', optimizer=optimizer,
```

```
        # Build the generator
        self.generator = self.build_generator()

        # The generator takes malware and noise as input and generates adversarial malware
        example = Input(shape=(self.urlfeature_dims,))
        noise = Input(shape=(self.z_dims,))
        input = [example, noise]
        malware_examples = self.generator(input)

        # For the combined model we will only train the generator
        self.substitute_detector.trainable = False

        # The discriminator takes generated URL as input and determines validity
        validity = self.substitute_detector(malware_examples)

        # The combined model  (stacked generator and substitute_detector)
        # Trains the generator to fool the discriminator
        self.combined = Model(input, validity)
        self.combined.compile(loss='binary_crossentropy', optimizer=optimizer)

    def build_blackbox_detector(self):

        if self.blackbox == 'MLP':
            blackbox_detector = MLPClassifier(hidden_layer_sizes=(50,), max_iter=10, alpha
                                              solver='sgd', verbose=0, tol=1e-4, random_st
                                              learning_rate_init=.1)
        return blackbox_detector

    def build_generator(self):

        example = Input(shape=(self.urlfeature_dims,))
        noise = Input(shape=(self.z_dims,))
        x = Concatenate(axis=1)([example, noise])
        i =0
        for dim in self.generator_layers[1:]:
            x = Dense(dim)(x)
            x = Activation(activation='sigmoid')(x)
        x = Maximum()([example, x])
        generator = Model([example, noise], x, name='generator')
        generator.summary()
        return generator

    def build_substitute_detector(self):

        input = Input(shape=(self.substitute_detector_layers[0],))
        x = input
        for dim in self.substitute_detector_layers[1:]:
            x = Dense(dim)(x)
            x = Activation(activation='sigmoid')(x)
        substitute_detector = Model(input, x, name='substitute_detector')
        substitute_detector.summary()
        return substitute_detector

    def load_data(self, filename):

        data = np.load(filename)
        xmal, ymal, xben, yben = data['xmal'], data['ymal'], data['xben'], data['yben']
        return (xmal, ymal), (xben, yben)


    def train(self, epochs, batch_size):

        # Load the dataset
        (xmal, ymal), (xben, yben) = self.load_data('data.npz')
        xtrain_mal, xtest_mal, ytrain_mal, ytest_mal = train_test_split(xmal, ymal, test_s
        xtrain_ben, xtest_ben, ytrain_ben, ytest_ben = train_test_split(xben, yben, test_s
```

```
        # Train blackbox_detctor
        self.blackbox_detector.fit(np.concatenate([xmal, xben]),
                                   np.concatenate([ymal, yben]))

        ytrain_ben_blackbox = self.blackbox_detector.predict(xtrain_ben)
        Original_Train_TRR = self.blackbox_detector.score(xtrain_mal, ytrain_mal)
        Original_Test_TRR = self.blackbox_detector.score(xtest_mal, ytest_mal)
        Train_TRR, Test_TRR = [], []

        for epoch in range(epochs):

            for step in range(1):#range(xtrain_mal.shape[0] // batch_size):
                # ---------------------
                #  Train substitute_detector
                # ---------------------

                # Select a random batch of malware examples
                idx = np.random.randint(0, xtrain_mal.shape[0], batch_size)
                xmal_batch = xtrain_mal[idx]
                noise = np.random.uniform(0, 1, (batch_size, self.z_dims))
                idx = np.random.randint(0, xmal_batch.shape[0], batch_size)
                xben_batch = xtrain_ben[idx]
                yben_batch = ytrain_ben_blackbox[idx]

                # Generate a batch of new malware examples
                gen_examples = self.generator.predict([xmal_batch, noise])
                ymal_batch = self.blackbox_detector.predict(np.ones(gen_examples.shape)*(g

                # Train the substitute_detector
                d_loss_real = self.substitute_detector.train_on_batch(gen_examples, ymal_b
                d_loss_fake = self.substitute_detector.train_on_batch(xben_batch, yben_bat
                d_loss = 0.5 * np.add(d_loss_real, d_loss_fake)

                # ---------------------
                #  Train Generator
                # ---------------------

                idx = np.random.randint(0, xtrain_mal.shape[0], batch_size)
                xmal_batch = xtrain_mal[idx]
                noise = np.random.uniform(0, 1, (batch_size, self.z_dims))

                # Train the generator
                g_loss = self.combined.train_on_batch([xmal_batch, noise], np.zeros((batch

            # Compute Train TRR
            noise = np.random.uniform(0, 1, (xtrain_mal.shape[0], self.z_dims))
            gen_examples = self.generator.predict([xtrain_mal, noise])
            TRR = self.blackbox_detector.score(np.ones(gen_examples.shape) * (gen_examples
            Train_TRR.append(TRR)

            # Compute Test TRR
            noise = np.random.uniform(0, 1, (xtest_mal.shape[0], self.z_dims))
            gen_examples = self.generator.predict([xtest_mal, noise])
            TRR = self.blackbox_detector.score(np.ones(gen_examples.shape) * (gen_examples
            Test_TRR.append(TRR)

            # Plot the progress
            print("%d [D loss: %f, acc.: %.2f%%] [G loss: %f]" % (epoch, d_loss[0], 100*d_

        print('Original_Train_TRR: {0}, Adver_Train_TRR: {1}'.format(Original_Train_TRR, T
        print('Original_Test_TRR: {0}, Adver_Test_TRR: {1}'.format(Original_Test_TRR, Test

        # Plot TRR
        plt.figure()
        plt.plot(range(epochs), Train_TRR, c='r', label='Training Set', linewidth=2)
```

```python
        plt.plot(range(epochs), Test_TRR, c='g', linestyle='--', label='Validation Set', l
        plt.xlabel("Epoch")
        plt.ylabel("TRR")
        plt.legend()
        plt.show()
```

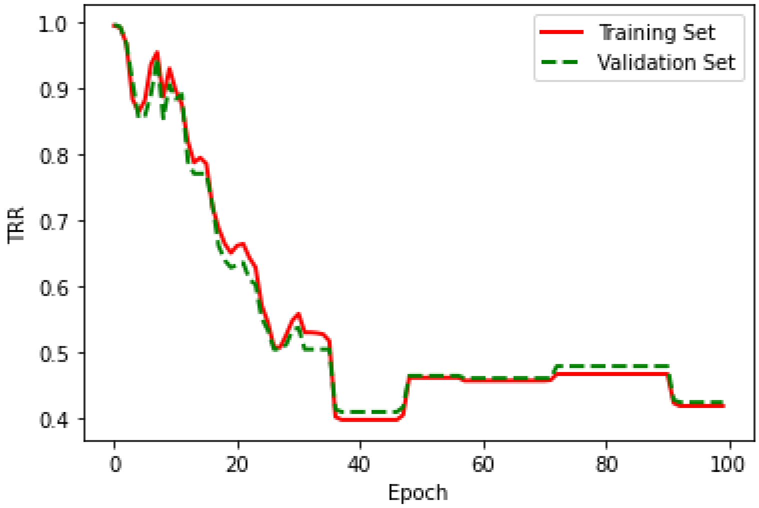
In [97]:

```python
gan = GAN()
gan.train(epochs=100, batch_size=128)
```

```
/usr/local/lib/python3.9/site-packages/keras/optimizer_v2/optimizer_v2.py:355: UserWarnin
g: The `lr` argument is deprecated, use `learning_rate` instead.
  warnings.warn(
/usr/local/lib/python3.9/site-packages/sklearn/neural_network/_multilayer_perceptron.py:61
4: ConvergenceWarning: Stochastic Optimizer: Maximum iterations (10) reached and the optim
ization hasn't converged yet.
  warnings.warn(
Model: "substitute_detector"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
input_176 (InputLayer)       [(None, 128)]             0
_________________________________________________________________
dense_140 (Dense)            (None, 256)               33024
_________________________________________________________________
activation_140 (Activation)  (None, 256)               0
_________________________________________________________________
dense_141 (Dense)            (None, 1)                 257
_________________________________________________________________
activation_141 (Activation)  (None, 1)                 0
=================================================================
Total params: 33,281
Trainable params: 33,281
Non-trainable params: 0
_________________________________________________________________
Model: "generator"
__________________________________________________________________________________________
________
Layer (type)                    Output Shape         Param #     Connected to

==========================================================================================
========
input_177 (InputLayer)          [(None, 128)]        0


__________________________________________________________________________________________
________
input_178 (InputLayer)          [(None, 20)]         0


__________________________________________________________________________________________
________
concatenate_35 (Concatenate)    (None, 148)          0           input_177[0][0]

                                                                 input_178[0][0]


__________________________________________________________________________________________
________
dense_142 (Dense)               (None, 256)          38144       concatenate_35[0][0]


__________________________________________________________________________________________
________
activation_142 (Activation)     (None, 256)          0           dense_142[0][0]


__________________________________________________________________________________________
________
dense_143 (Dense)               (None, 128)          32896       activation_142[0][0]
```

```
________________________________________________________________________________________________
________
activation_143 (Activation)     (None, 128)          0           dense_143[0][0]


________________________________________________________________________________________________
________
maximum_35 (Maximum)            (None, 128)          0           input_177[0][0]

                                                                 activation_143[0][0]

================================================================================================
========
Total params: 71,040
Trainable params: 71,040
Non-trainable params: 0
________________________________________________________________________________________________
________
0 [D loss: 0.781978, acc.: 57.03%] [G loss: 1.230921]
1 [D loss: 0.731020, acc.: 52.34%] [G loss: 1.328501]
2 [D loss: 0.686657, acc.: 51.95%] [G loss: 1.350267]
3 [D loss: 0.648237, acc.: 51.56%] [G loss: 1.325608]
4 [D loss: 0.651853, acc.: 44.92%] [G loss: 1.256318]
5 [D loss: 0.608786, acc.: 56.64%] [G loss: 1.163922]
6 [D loss: 0.569364, acc.: 71.09%] [G loss: 1.109389]
7 [D loss: 0.522514, acc.: 88.28%] [G loss: 1.020716]
8 [D loss: 0.533411, acc.: 88.28%] [G loss: 0.995506]
9 [D loss: 0.503714, acc.: 85.94%] [G loss: 1.018713]
10 [D loss: 0.496190, acc.: 86.33%] [G loss: 0.993055]
11 [D loss: 0.486116, acc.: 85.94%] [G loss: 1.053880]
12 [D loss: 0.481208, acc.: 82.03%] [G loss: 1.094486]
13 [D loss: 0.460867, acc.: 86.72%] [G loss: 1.101271]
14 [D loss: 0.452157, acc.: 85.94%] [G loss: 1.118625]
15 [D loss: 0.451968, acc.: 82.42%] [G loss: 1.163654]
16 [D loss: 0.431376, acc.: 85.16%] [G loss: 1.199059]
17 [D loss: 0.417077, acc.: 89.45%] [G loss: 1.124848]
18 [D loss: 0.399203, acc.: 89.06%] [G loss: 1.189390]
19 [D loss: 0.395833, acc.: 89.45%] [G loss: 1.160972]
20 [D loss: 0.403804, acc.: 88.28%] [G loss: 1.013785]
21 [D loss: 0.357251, acc.: 88.28%] [G loss: 1.021309]
22 [D loss: 0.343609, acc.: 91.80%] [G loss: 0.895685]
23 [D loss: 0.356019, acc.: 86.72%] [G loss: 0.815911]
24 [D loss: 0.368073, acc.: 87.50%] [G loss: 0.943958]
25 [D loss: 0.324043, acc.: 91.80%] [G loss: 0.901651]
26 [D loss: 0.323038, acc.: 88.28%] [G loss: 0.778078]
27 [D loss: 0.339961, acc.: 86.72%] [G loss: 0.804437]
28 [D loss: 0.296995, acc.: 89.84%] [G loss: 0.836193]
29 [D loss: 0.362348, acc.: 83.59%] [G loss: 0.756870]
30 [D loss: 0.310188, acc.: 87.11%] [G loss: 0.784535]
31 [D loss: 0.305380, acc.: 85.55%] [G loss: 0.736203]
32 [D loss: 0.375662, acc.: 81.25%] [G loss: 0.849697]
33 [D loss: 0.331100, acc.: 85.55%] [G loss: 0.879571]
34 [D loss: 0.327544, acc.: 87.11%] [G loss: 0.787990]
35 [D loss: 0.334307, acc.: 87.50%] [G loss: 0.921843]
36 [D loss: 0.286061, acc.: 91.80%] [G loss: 1.248961]
37 [D loss: 0.373962, acc.: 81.64%] [G loss: 0.993514]
38 [D loss: 0.323942, acc.: 87.50%] [G loss: 0.824201]
39 [D loss: 0.320244, acc.: 84.38%] [G loss: 0.704336]
40 [D loss: 0.277346, acc.: 87.89%] [G loss: 0.657996]
41 [D loss: 0.287314, acc.: 89.06%] [G loss: 0.684422]
42 [D loss: 0.244031, acc.: 91.80%] [G loss: 0.629411]
43 [D loss: 0.269794, acc.: 90.62%] [G loss: 0.546009]
44 [D loss: 0.337331, acc.: 84.77%] [G loss: 0.503772]
45 [D loss: 0.285121, acc.: 89.45%] [G loss: 0.564704]
46 [D loss: 0.252708, acc.: 89.06%] [G loss: 0.611613]
47 [D loss: 0.257297, acc.: 90.62%] [G loss: 0.604938]
```

```
48 [D loss: 0.244191, acc.: 91.41%] [G loss: 0.655069]
49 [D loss: 0.271167, acc.: 88.28%] [G loss: 0.723370]
50 [D loss: 0.299015, acc.: 86.72%] [G loss: 0.811491]
51 [D loss: 0.256939, acc.: 89.84%] [G loss: 0.879012]
52 [D loss: 0.259763, acc.: 87.11%] [G loss: 0.766847]
53 [D loss: 0.258713, acc.: 88.28%] [G loss: 0.835461]
54 [D loss: 0.272965, acc.: 88.28%] [G loss: 0.986917]
55 [D loss: 0.241988, acc.: 91.02%] [G loss: 1.137153]
56 [D loss: 0.273576, acc.: 88.28%] [G loss: 0.819868]
57 [D loss: 0.268945, acc.: 89.84%] [G loss: 0.904635]
58 [D loss: 0.221217, acc.: 91.80%] [G loss: 0.953425]
59 [D loss: 0.261768, acc.: 89.45%] [G loss: 0.875978]
60 [D loss: 0.254653, acc.: 92.19%] [G loss: 0.907187]
61 [D loss: 0.305397, acc.: 85.55%] [G loss: 0.820016]
62 [D loss: 0.247523, acc.: 89.84%] [G loss: 0.947330]
63 [D loss: 0.254720, acc.: 90.23%] [G loss: 0.885010]
64 [D loss: 0.258922, acc.: 88.67%] [G loss: 0.841163]
65 [D loss: 0.260687, acc.: 89.45%] [G loss: 1.041038]
66 [D loss: 0.250898, acc.: 92.19%] [G loss: 0.947163]
67 [D loss: 0.215982, acc.: 92.58%] [G loss: 1.116484]
68 [D loss: 0.234695, acc.: 91.41%] [G loss: 0.966236]
69 [D loss: 0.262651, acc.: 89.84%] [G loss: 1.203867]
70 [D loss: 0.256436, acc.: 92.58%] [G loss: 1.025297]
71 [D loss: 0.261866, acc.: 91.41%] [G loss: 0.988736]
72 [D loss: 0.236021, acc.: 90.23%] [G loss: 1.059554]
73 [D loss: 0.238034, acc.: 89.06%] [G loss: 1.143010]
74 [D loss: 0.253513, acc.: 89.06%] [G loss: 0.944537]
75 [D loss: 0.258223, acc.: 87.11%] [G loss: 0.862337]
76 [D loss: 0.262677, acc.: 88.28%] [G loss: 1.108879]
77 [D loss: 0.245561, acc.: 90.62%] [G loss: 1.051178]
78 [D loss: 0.206718, acc.: 92.19%] [G loss: 0.973508]
79 [D loss: 0.260429, acc.: 87.50%] [G loss: 1.186132]
80 [D loss: 0.215749, acc.: 92.19%] [G loss: 1.189877]
81 [D loss: 0.231629, acc.: 90.62%] [G loss: 0.925854]
82 [D loss: 0.236298, acc.: 90.23%] [G loss: 0.899351]
83 [D loss: 0.223892, acc.: 92.19%] [G loss: 1.053852]
84 [D loss: 0.202934, acc.: 92.58%] [G loss: 0.913822]
85 [D loss: 0.253467, acc.: 90.23%] [G loss: 1.056876]
86 [D loss: 0.223322, acc.: 90.23%] [G loss: 1.001502]
87 [D loss: 0.253707, acc.: 89.84%] [G loss: 1.216334]
88 [D loss: 0.193784, acc.: 94.14%] [G loss: 1.241309]
89 [D loss: 0.223222, acc.: 91.41%] [G loss: 1.184116]
90 [D loss: 0.225768, acc.: 91.02%] [G loss: 1.161214]
91 [D loss: 0.174051, acc.: 93.75%] [G loss: 1.164741]
92 [D loss: 0.215559, acc.: 92.58%] [G loss: 1.467723]
93 [D loss: 0.221807, acc.: 91.02%] [G loss: 1.006514]
94 [D loss: 0.174793, acc.: 93.36%] [G loss: 0.965723]
95 [D loss: 0.212321, acc.: 90.23%] [G loss: 0.894228]
96 [D loss: 0.157924, acc.: 94.92%] [G loss: 1.021028]
97 [D loss: 0.208529, acc.: 91.41%] [G loss: 0.750909]
98 [D loss: 0.243951, acc.: 89.45%] [G loss: 0.922338]
99 [D loss: 0.186426, acc.: 92.97%] [G loss: 0.863404]
Original_Train_TRR: 0.9817184643510055, Adver_Train_TRR: 0.41773308957952465
Original_Test_TRR: 0.9781021897810219, Adver_Test_TRR: 0.4233576642335766
```



In [ ]:

Result : As soon as we used a new dataset with 128 features instead of the initial dataset with 60 features i.e double the number of features the accuracy got increased from 49 % to 92 %.