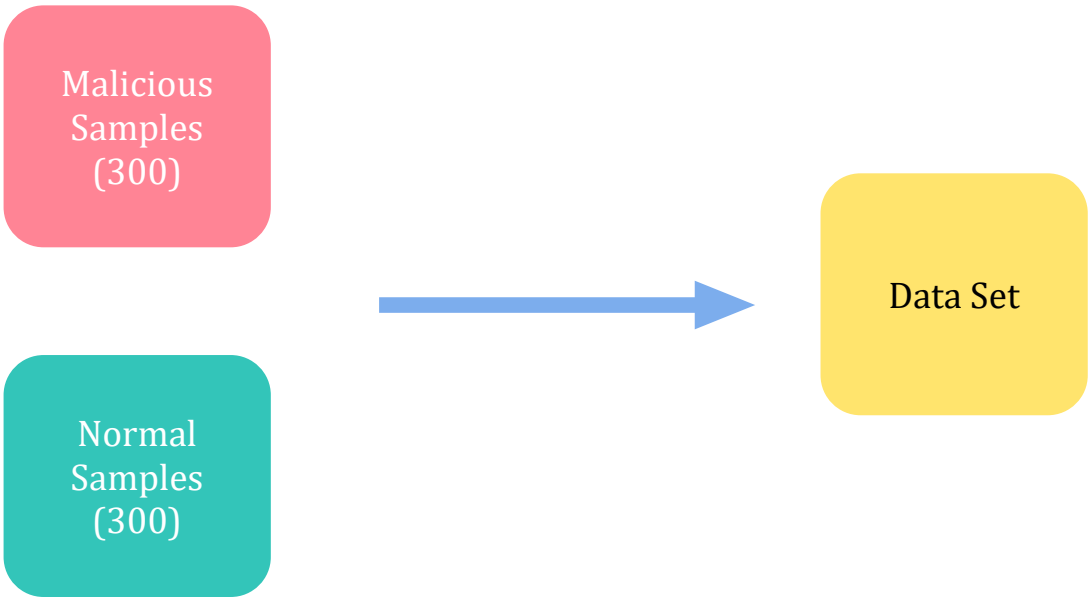
Malware Detection with Hardware Counter Performance Observing

180101012/ İbrahim Yusuf Cosgun

Introduction: In this research, we want to detecting malware with Hardware performance counter . Due to the malware behaviors ,they can obfuscate himself but we now that the wolf is wolf even if he wears sheep'skin. %95 malware behavior same. In this paper we are proof that with observing 300 malicious software and normal software. Also we use to Machine learning technic and train the model. LSTM is one of the good one.

Data Collection

We have collected all our data with the method we mentioned in HPCS_manuel.* Also we are deactivated to Anti-virus programs for observing. Cause of our malwares as known as dataset. So it can easily blocking with AV systems. Three different Virtual machine but same configuration used .



Chosing Machine Learning Model

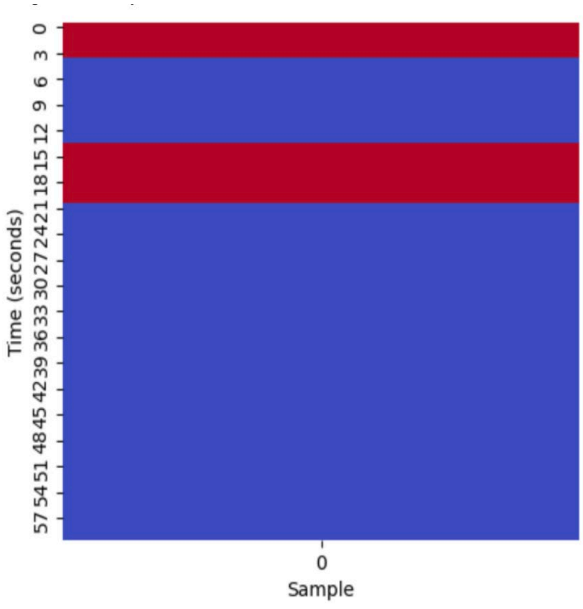
In this part we are focusing witch model is best for detection to malicious. Our first choice was “Decision Tree Classification” proving version “Random Forrest Classifier”. Trained model with 0.2 Test and 0.8 tranin dividing to the dataset. But the problem of overfitting. Model accuracy reached “1”. However, when we tested our model with a new sample outside our data set, our model gave a False-Positive result.

```
Output: This sample is 18.33 percent likely malware.
```

Then, when we printed the prediction array that our model predicted, we realized what happened. Instead of making a prediction for a sample data (csv file) as a whole, our model predicted the sample data line by line.

```
Prediction Array: [1 1 1 1 0 0 0 0 0 0 0 0 0 0 0 1 1 1 1 1 1 1 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0]
```

When we created a heatmap based on the prediction data above, the result was as follows.



What we learned from this result is that instead of training our model with the data of 600 samples, we actually trained our model with data rows of 600 * 60 (that is, the total number of rows). After realizing that this approach produced a result we did not want, we decided to change the type of our model.

After researching we decide Long Short Term Memory (Deep Learning Approach) is best chose , cause of our dataset creating with time-series approach method. We was listen to all samples sixty second.

% Privileged Time	Handle Count	IO Data Bytes/sec	IO Data Operations/sec	IO Other Bytes/sec	IO Other Operations/sec	IO Read Bytes/sec	IO Read Operations/sec	IO Write Bytes/sec	IO Write Operations/sec	Page Faults/sec	Page File Bytes	Pa
703.5098877	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	61440.0	6
741.3031616	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	61440.0	6
728.0539551	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	61440.0	6
689.8632202	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	61440.0	6
645.1057739	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	61440.0	6
672.6997681	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	61440.0	6
731.171936	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	61440.0	6
697.2880859	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	61440.0	6
694.289856	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	61440.0	6
727.9807129	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	61440.0	6

Head of samples data

Codes

We wrote our codes with Jupyter notebook. Python 3.11

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

These lines import the necessary libraries: `pandas` for data manipulation, `numpy` for numerical operations, and `os` for interacting with the operating system.

```
from keras.models import Sequential
from keras.layers import LSTM, Dense
from keras.callbacks import EarlyStopping
```

These lines import the necessary components from Keras: `Sequential` for initializing a linear stack of layers, LSTM and Dense for the types of layers to be used in the model, and `EarlyStopping` for stopping the training when a monitored metric has stopped improving.

```
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler,
MinMaxScaler
```

These lines import `train_test_split` for splitting the data into training and testing sets, and `StandardScaler` and `MinMaxScaler` for scaling the features.

```
scaler = StandardScaler()
data = []
folders = ['dataset/malware', 'dataset/safe']
```

These lines an instance of `StandardScaler` is created, an empty list `data` is initialized to store the data, and a list `folders` is created to store the paths of the `folders` containing the data.

```
for folder in folders:
    for i in range(1, 301):
        file_path = os.path.join(folder, f'{i}.csv')
        df = pd.read_csv(file_path)
        array = df.values.reshape(1, df.values.shape[0],
        df.values.shape[1])
        data.append(array)
```

These lines loop over each folder in `folders`, count the number of files in each `folder`, and then loop over each file in the folder. Each file is read into a pandas DataFrame, reshaped into a 3D array, and appended to the `data` list.

```
data = np.concatenate(data)
```

This line concatenates all the arrays in the `data` list into a single numpy array.

```
targets = np.concatenate([np.ones(300), np.zeros(300)])
```

This line creates a numpy array of `targets`, with 300 ones (representing the malware' class) and 300 zeros (representing the safe' class).

```
X_train, X_test, y_train, y_test = train_test_split(data,
targets, test_size=0.2, random_state=42)
```

This line splits the data and targets into training and testing sets, with 80% of the data used for training and 20% used for testing.

```
model = Sequential()
```

This line initializes a Sequential model.

```
timesteps = 60
features = 23
model.add(LSTM(960, input_shape=(timesteps, features)))
```

These lines adds an LSTM layer to the model with 960 units. The `input_shape` is specified as `(timesteps, features)`. The reason why we chose 60 and 23 is that our data is like this, and the reason why we created a layer consisting of 960 units will be explained in detail "Consulation of Model" section.

```
model.add(Dense(1, activation='sigmoid'))
```

This line adds a `Dense` layer to the model with 1 unit and a `sigmoid` function, which is suitable for binary classification.

———————————————————————————————

```
model.compile(loss='binary_crossentropy', optimizer='adam',
metrics=['accuracy'])
```

This line compiles the model with the binary `crossentropy` loss function, the Adam optimizer, and `accuracy` as the metric.

———————————————————————————————

```
early_stopping = EarlyStopping(monitor='val_loss',
patience=3)
```

This line creates an EarlyStopping callback, which will stop the training if the validation loss doesn't improve for 3 epochs. We used it to avoid overfitting situations.

———————————————————————————————

```
history = model.fit(X_train, y_train, epochs=100,
batch_size=40, validation_data=(X_test, y_test),
callbacks=[early_stopping])
```

Finally, this line fits the model to the training data for 100 epochs with a batch size of 40, using the testing data as validation data, and the `EarlyStopping` callback. The training history is stored in the `history` variable.

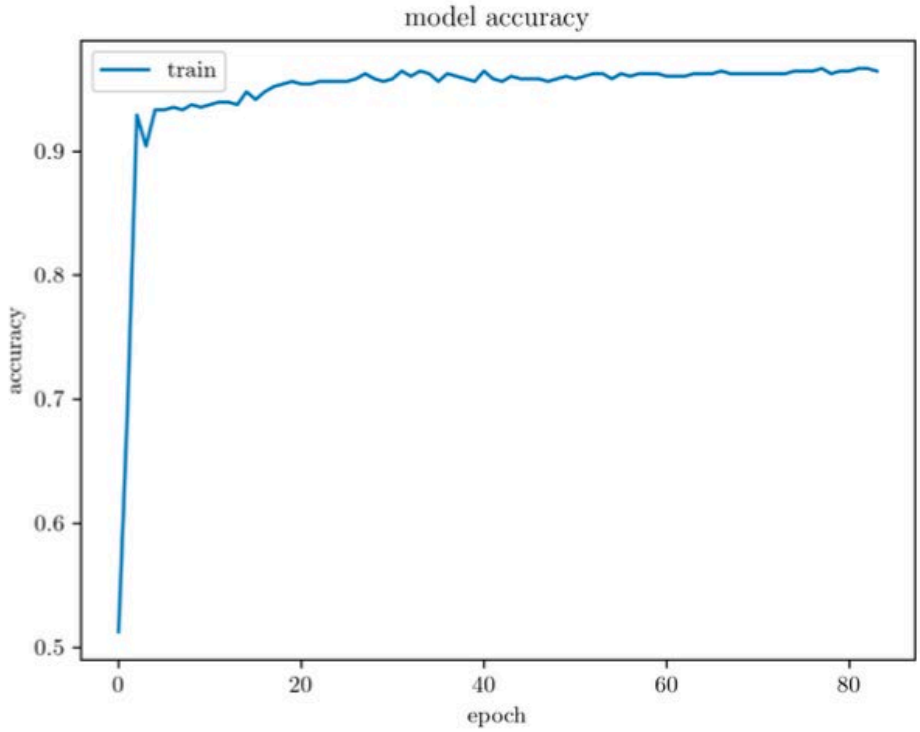
———————————————————————————————

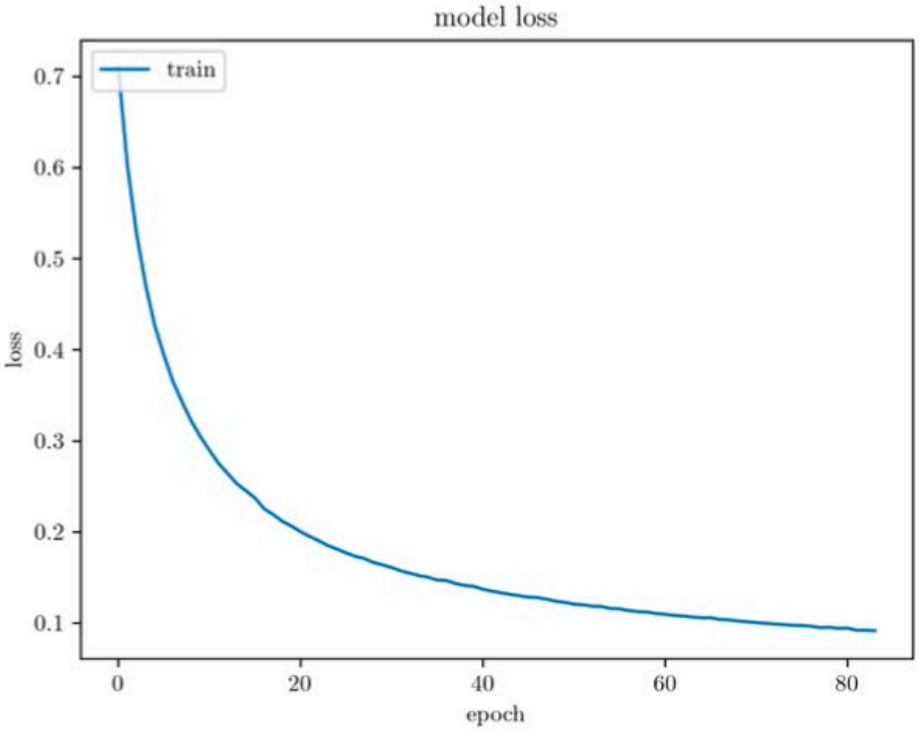
```
import matplotlib.pyplot as plt
plt.figure(figsize=(14,3))
plt.subplot(1, 2, 1)
plt.plot(history.history['accuracy'])
plt.plot(history.history['val_accuracy'])
plt.title('Model accuracy')
plt.ylabel('Accuracy')
plt.xlabel('Epoch')
plt.legend(['Train', 'Test'], loc='upper left')
plt.subplot(1, 2, 2)
plt.plot(history.history['loss'])
plt.plot(history.history['val_loss'])
plt.title('Model loss')
plt.ylabel('Loss')
plt.xlabel('Epoch')
plt.legend(['Train', 'Test'], loc='upper left')
plt.show()
```

These lines provide us with a visual graph using the `matplotlib.pyplot` library to graphically show the `history` of the training of the model.

CONSULATION OF MODEL

Here is shown result of model training history.





Here is training model results which setting with different parameters

Parameters			Malware		Safe		Score
LSTM Layer	Epoch	Batch Size	Malware (**TP**)	Safe (**FP**)	Malware (**FN**)	Safe (**TN**)	
60	20	60	196	104	34	266	**77**
60	100	120	244	56	32	268	**85,33**
60	200	120	241	59	33	267	**84,66**
60	100	240	257	43	18	282	**89,83**
80	50	30	291	9	44	256	**91,16**
80	100	240	250	50	41	259	**84,83**
100	100	240	263	37	35	265	**88**
120	100	240	260	40	42	258	**86,33**
240	100	240	299	1	43	257	**92,66**
240	100	120	294	6	38	262	**92,66**
240	100	60	260	40	22	278	**89,66**
300	100	240	241	59	25	275	**86**
300	200	240	40	260	30	270	**51,66**
300	100	480	207	93	36	264	**78,5**
300	100	240	280	20	36	264	**90,66**
300	100	120	297	3	36	264	**93,5**
300	100	60	293	7	25	275	**94,66**
300	100	30	294	6	20	280	**95,66**
300	100	16	298	2	23	277	**95,83**

*After this point, we trained the model with the **early_stopping** callback to prevent overfitting.

960	100	5	291	9	17	283	**95,66**
960	100	10	297	3	23	277	**95,66**
960	100	20	296	4	12	288	**97,33**

When we tested this model, we learned by trying that the success rate was weak on an sample completely outside of our data set, although the result seemed to be successful. For this reason, we decided to make tests by scaling our model with different scaling methods.

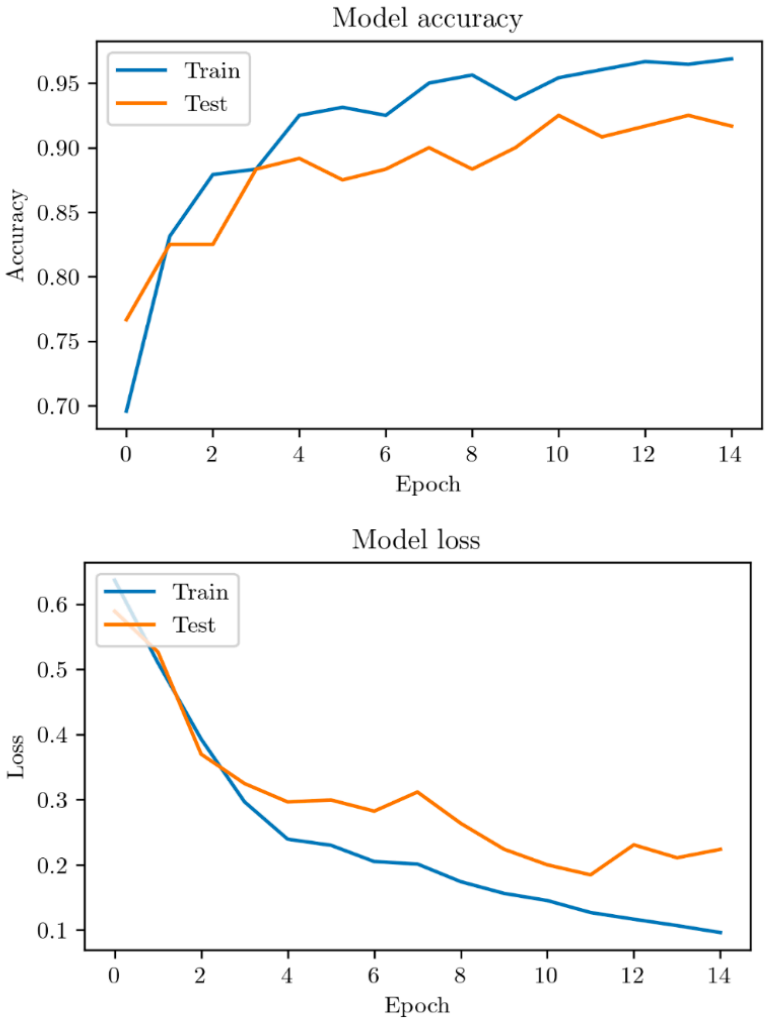
When we selected the parameters to train the model we scaled, I saw that the model terminated the training with `early_stop` after 5-20 epochs, regardless of the parameters we chose. Afterwards, I tried setting the parameters to smaller values. Since the results I got were not satisfactory, we did some research. When adding LSTM layers, the parameters `dropout=0.2,recurrent_dropout=0.2` increased the predictive power of the model. It also minimized the problem we experienced (there was instability during training, as if we had chosen the learning rate (α) too high in linear regression).

After doing that ,here is the final result:

80	10	8	277	23	19	281	**93**	**MinMax**
80	10	8	293	7	14	286	**96,5**	**Standard**
120	10	32	284	16	9	291	**95,83**	**Standard**
240	15	120	291	9	9	291	**97**	**Standard**

Yellow marked line is our final score what we reached.

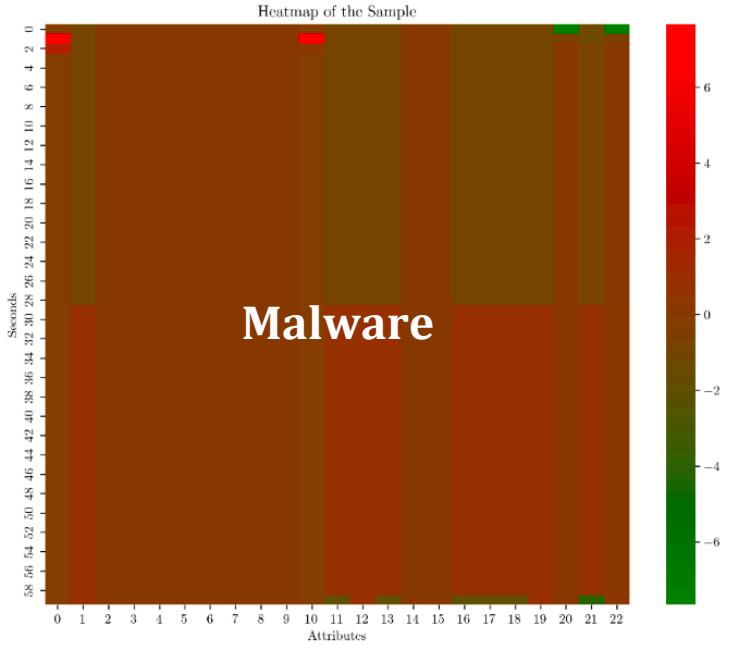
Now let's check new accuracy and loss.

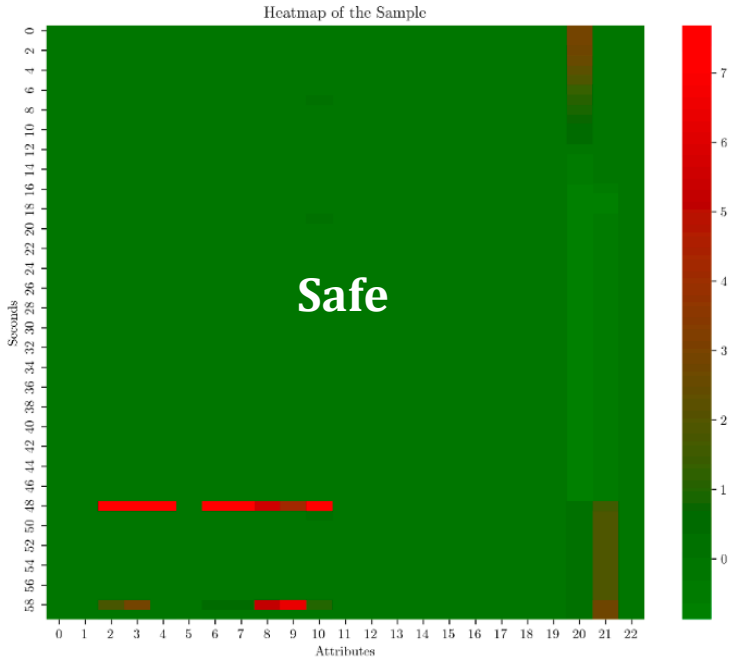


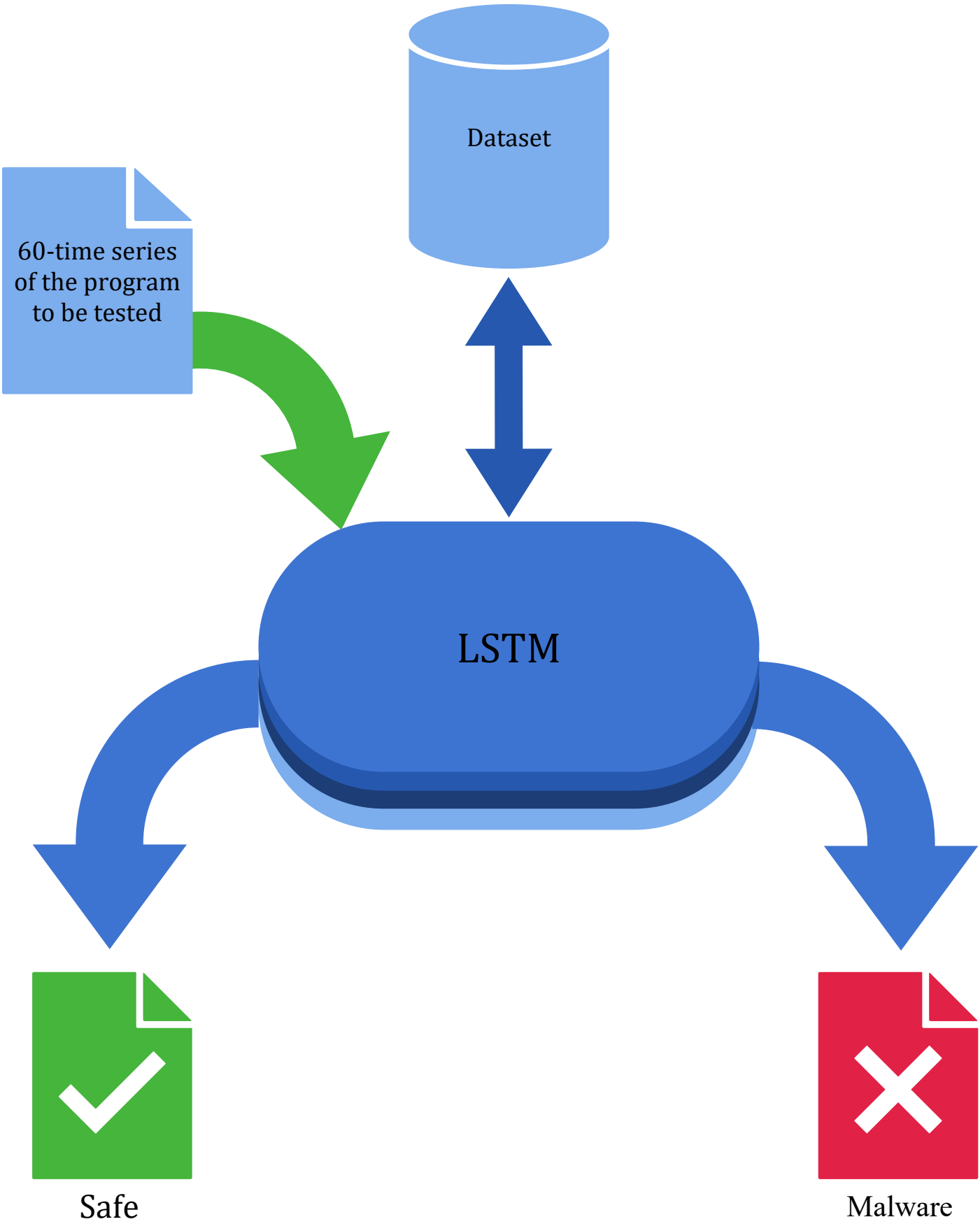
```
def load_single_sample(filepath):
    df = pd.read_csv(filepath)
    df_scaled = scaler.fit_transform(df.values)
    return np.array([df_scaled])

single_sample = load_single_sample("V1.csv")
prediction = model.predict(single_sample)
```

And there is we wrote a method for testing came from outcome(not in dataset) . We carried out our test again with two examples (malware & safe). And we made it show the result with heatmap.






Dataset
60-time series of the program to be tested
LSTM
Safe
Malware