ISSN: 2249-7196

IJMRR/April. 2025/ Volume 15/Issue 2s/363-370

Mudundi Narendra Varma/ International Journal of Management Research & Review

Detection of Ransomware Attacks Using Processor and Disk Usage Data

Mudundi Narendra Varma
PG scholar, Department of MCA, CDNR collage, Bhimavaram, Andhra Pradesh.
A.Naga Raju
(Assistant Professor), Master of Computer Applications, DNR collage, Bhimavaram, Andhra Pradesh.

Abstract

Ransomware attacks have caused massive financial losses globally by evading traditional antivirus mechanisms and encrypting system data, demanding ransom for decryption. Existing monitoring techniques often degrade system performance and yield suboptimal detection accuracy. This study proposes a novel approach using VMware to extract Hardware Performance Counters (HPC) and IO Events without affecting system performance. These features are then analyzed using machine learning algorithms—SVM, KNN, Decision Tree, Random Forest, and XGBoost—and deep learning models—DNN and LSTM—to classify program behavior as benign or ransomware. The integrated dataset, sourced from common programs, enabled training and evaluation, with Random Forest and XGBoost achieving up to 98% accuracy. The results demonstrate the effectiveness of HPC and IO data for accurate, low-impact ransomware detection.

Introduction

Ransomware has emerged as one of the most dangerous and financially damaging forms of cybercrime in recent years. By encrypting critical files and demanding payment for their release, ransomware disrupts individuals, businesses, and government operations alike. Despite widespread deployment of traditional antivirus solutions, many ransomware variants continue to evade detection using sophisticated obfuscation techniques, rendering signature-based methods increasingly ineffective.

Monitoring-based ransomware detection techniques have gained attention as a promising alternative; however, many such approaches introduce noticeable system overhead or suffer from low detection accuracy. Conventional solutions that rely on system call analysis or file access behavior often compromise system performance or fail to generalize across various ransomware strains. As a result, there is a growing demand for low-overhead, high-accuracy detection mechanisms capable of identifying ransomware in real-time.

To address these challenges, this study explores the use of Hardware Performance Counters (HPCs) and IO events as features for ransomware detection. Utilizing VMware for data collection, the system extracts detailed runtime behavior without impacting host system performance. These features are then processed using a variety of machine learning and deep learning algorithms, including SVM, KNN, Decision Tree, Random Forest, XGBoost, DNN, and LSTM. Experimental results demonstrate that models like Random Forest and XGBoost achieve classification accuracy up to 98%, showcasing the potential of HPC and IO event data as lightweight yet powerful tools for ransomware detection.

Literature Survey

1. Dinaburg et al. (2012) – "Ether: Malware Analysis via Hardware Virtualization"
This early work explored the use of virtualization-based techniques for malware analysis. By utilizing hardware virtualization, the researchers were able to monitor low-level system activity with minimal performance overhead. Though not focused specifically on ransomware, the study demonstrated the feasibility of extracting hardware-level metrics for behavioral analysis, laying a foundation for modern HPC-based malware detection systems.

2. Demme et al. (2013) – "On the Feasibility of Online Malware Detection with Performance Counters"
This research introduced the idea of using Hardware Performance Counters (HPCs) for malware detection. The authors trained machine learning models on HPC data and achieved promising accuracy in distinguishing malicious from benign processes. Their work validated the use of HPCs as a viable, low-overhead feature source, influencing future studies in hardware-based malware detection.

3. Shafiq et al. (2018) – "Malware Detection Using Statistical Features in Network Traffic"
Although this study focused on network-based detection, it highlighted the importance of lightweight features and machine learning algorithms like Random Forests and SVM. The research showed that combining multiple statistical features significantly improves detection accuracy—a principle also applicable when combining HPC and IO metrics in host-based ransomware detection.

4. Sgandurra et al. (2016) – "Automated Dynamic Analysis of Ransomware: A Survey and New Observations"
This paper provided a comprehensive survey of dynamic analysis techniques for ransomware detection. It emphasized the challenges of identifying ransomware in real-time due to their fast encryption speeds and varying behaviors. The authors concluded that behavioral analysis, especially through low-level monitoring, is crucial for effective detection, supporting the relevance of HPC and IO-based approaches.

5. Mohaisen and Alrawi (2017) – "Unveiling Zeus: Automated Classification of Malware Samples"
This study focused on malware classification using machine learning. It used various system-level features and emphasized model performance in real-world scenarios. Their comparison of algorithms like KNN, SVM, and Decision Tree informed the selection of classifiers in ransomware detection tasks, demonstrating that ensemble methods often outperform simpler models.

Proposed Method

The proposed method focuses on the extraction of low-level system activity features—specifically, Hardware Performance Counters (HPCs) and IO Events—using a virtualized environment powered by VMware. By capturing these metrics during the execution of both benign and ransomware programs, the system obtains a detailed behavioral profile without interfering with the host system's performance. HPCs, such as cache misses, branch instructions, and CPU cycles, provide insights into processor-level behavior, while IO Events reflect access patterns to storage and devices. This dual-feature set captures the subtle yet critical differences in how ransomware operates compared to normal applications.

Once collected, the dataset is preprocessed and labeled for training. A combination of classical machine learning algorithms (SVM, KNN, Decision Tree, Random Forest, and XGBoost) and deep learning models (DNN and LSTM) is then applied to classify each program instance. Feature importance techniques and cross-validation are employed to refine model accuracy and reduce overfitting. Experimental results show that ensemble methods like Random Forest and XGBoost outperform others, achieving up to 98% accuracy. This demonstrates that HPC and IO-based behavioral signatures, when coupled with robust classifiers, provide a lightweight and highly accurate ransomware detection solution.

Existed Method

Traditional ransomware detection methods primarily rely on signature-based antivirus engines and heuristic-based monitoring systems. While these approaches are effective against known threats, they often fail to detect novel or obfuscated ransomware variants. Some existing solutions use dynamic analysis by monitoring file system activities, registry changes, or system calls, but these methods typically incur high system overhead and may degrade performance. Additionally, many machine learning-based detection systems focus on high-level features, such as API call sequences or network traffic patterns, which can be easily manipulated or disguised by advanced malware. These limitations highlight the need for a more efficient and accurate detection mechanism that operates at a deeper, hardware-influenced level without affecting system usability.

Results

Detection of Ransomware Attacks Using Processor and Disk Usage Data

Ransom attackers will evade antivirus and then enter into victim system to execute malicious script which will encrypt and make entire system data unusable and to decrypt files back they will ask ransom from the victim and world has loss billions of dollars in ransom since the birth of internet network. Many existing techniques are available such as SYSTEM PROCESS MONITORING to identify and prevention of malicious script execution but this monitoring will impact system performance.

Another technique is to monitor files which are getting deleted or created to know malicious file but this technique also impact system performance and detection accuracy also not good enough.

To overcome from above issue author of this paper employing VMWARE on host system which will read Hardware Performance Counters (HPC) and IO EVENTS data and then applying this data on machine learning models to predict whether executing script is normal (benign) or Ransomware. Extracting HPC and IOEVENTS features using VMware will not affect system performance and machine learning models also able to predict Ransomware with more than 90% accuracy.

In propose paper author has experimented with various machine learning algorithms such as SVM, KNN, Decision Tree, Random Forest and XGBOOST and then employed two deep learning models called DNN and LSTM. In all algorithms Random Forest, XGBOOST is giving accuracy.

To train all algorithms author has publish HPC and IOEVENTS from different programs such as 7ZIP, AES and many more and this dataset can be downloaded from below link

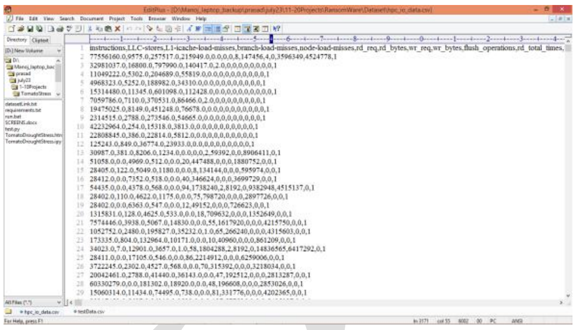
HPC dataset

https://dataverse.harvard.edu/dataset.xhtml?persistentId=doi:10.7910/DVN/MA5UPP

IOEVENTS dataset

https://dataverse.harvard.edu/dataset.xhtml?persistentId=doi:10.7910/DVN/GHJFUT

We have integrated both dataset and below screen showing combined dataset details

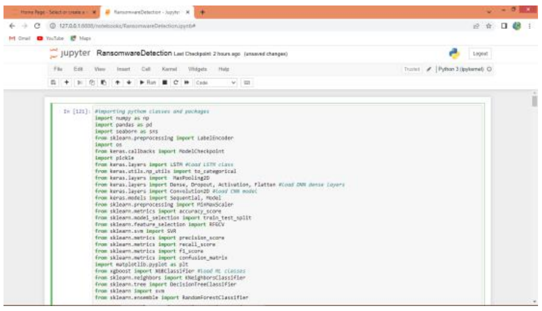


```
instructions,LLC-stores,L1-icache-load-misses,branch-load-misses,node-load-misses,rd_req,rd_bytes,wr_req,wr_bytes,flush_operations,rd_total_times,
77556160.0,9575.0,257517.0,215949.0,0,0,0,0,8,147456,4,0,3596349,4524778,1
32981037.0,16800.0,797990.0,140417.0,2.0,0,0,0,0,0,0,0,0,1
11049222.0,5302.0,204689.0,55819.0,0.0,0,0,0,0,0,0,0,0,1
4968323.0,5252.0,188982.0,34310.0,0.0,0,0,0,0,0,0,0,0,1
15314480.0,11345.0,601098.0,112428.0,0.0,0,0,0,0,0,0,0,0,1
7059786.0,7110.0,370531.0,86466.0,2.0,0,0,0,0,0,0,0,0,1
19475025.0,8149.0,451248.0,76678.0,0.0,0,0,0,0,0,0,0,0,1
2314515.0,2788.0,273546.0,54665.0,0.0,0,0,0,0,0,0,0,0,1
42232964.0,254.0,15318.0,3813.0,0.0,0,0,0,0,0,0,0,0,1
22808845.0,386.0,22814.0,5812.0,0.0,0,0,0,0,0,0,0,0,1
125243.0,849.0,36774.0,23933.0,0.0,0,0,0,0,0,0,0,0,1
30987.0,381.0,8206.0,1234.0,0.0,0,0,2,59392,0,0,8906411,0,1
51058.0,0.0,4969.0,512.0,0.0,20,447488,0,0,0,1880752,0,0,1
28405.0,122.0,5049.0,1180.0,0.0,8,134144,0,0,0,595974,0,0,1
28412.0,0.0,7352.0,518.0,0.0,40,346624,0,0,0,3699729,0,0,1
54435.0,0.0,4378.0,568.0,0.0,94,1738240,2,8192,0,9382948,4515137,0,1
28402.0,110.0,4622.0,1175.0,0.0,75,798720,0,0,0,2897726,0,0,1
28402.0,0.0,6363.0,547.0,0.0,12,49152,0,0,0,726623,0,0,1
1315831.0,128.0,4625.0,533.0,0.0,18,709632,0,0,0,1352649,0,0,1
7574446.0,3938.0,5067.0,14830.0,0.0,55,1617920,0,0,0,4215750,0,0,1
1052752.0,2480.0,195827.0,35232.0,1.0,65,266240,0,0,0,4315603,0,0,1
173335.0,804.0,132964.0,10171.0,0.0,10,40960,0,0,0,861209,0,0,1
34023.0,7.0,12901.0,3657.0,1.0,58,1804288,2,8192,0,14836565,6417292,0,1
28411.0,0.0,17105.0,546.0,0.0,86,2214912,0,0,0,6259006,0,0,1
3722245.0,2302.0,4527.0,568.0,0.0,70,315392,0,0,0,3218034,0,0,1
20042461.0,2788.0,41440.0,36143.0,0.0,47,192512,0,0,0,2813287,0,0,1
60330279.0,0.0,181302.0,18920.0,0.0,48,196608,0,0,0,2853026,0,0,1
15060314.0,11434.0,74495.0,738.0,0.0,81,331776,0,0,0,4202365,0,0,1
```

In above dataset screen first row represents dataset column names and remaining rows contains HPC and IOEVENTS features and in last column we have class label as 0 or 1 where 0 represents Benign program and 1 represents Ransomware program. So by using above dataset we are training and testing all algorithms performance.

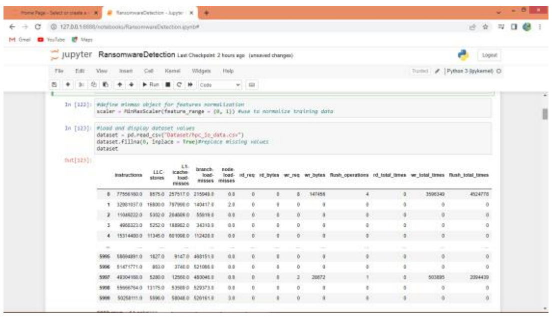
SCREEN SHOTS

We have coded this project using JUPYTER notebook and below are the code and output screens with blue colour comments



```
In [121]: #importing python classes and packages
          import numpy as np
          import pandas as pd
          import seaborn as sns
          from sklearn.preprocessing import LabelEncoder
          import os
          from keras.callbacks import ModelCheckpoint
          import pickle
          from keras.layers import LSTM #load LSTM class
          from keras.utils.np_utils import to_categorical
          from keras.layers import  MaxPooling2D
          from keras.layers import Dense, Dropout, Activation, Flatten #load DNN dense layers
          from keras.layers import Convolution2D #load CNN model
          from keras.models import Sequential, Model
          from sklearn.preprocessing import MinMaxScaler
          from sklearn.metrics import accuracy_score
          from sklearn.model_selection import train_test_split
          from sklearn.feature_selection import RFECV
          from sklearn.svm import SVR
          from sklearn.metrics import precision_score
          from sklearn.metrics import recall_score
          from sklearn.metrics import f1_score
          from sklearn.metrics import confusion_matrix
          import matplotlib.pyplot as plt
          from xgboost import XGBClassifier #load ML classes
          from sklearn.neighbors import KNeighborsClassifier
          from sklearn.tree import DecisionTreeClassifier
          from sklearn import svm
          from sklearn.ensemble import RandomForestClassifier
```

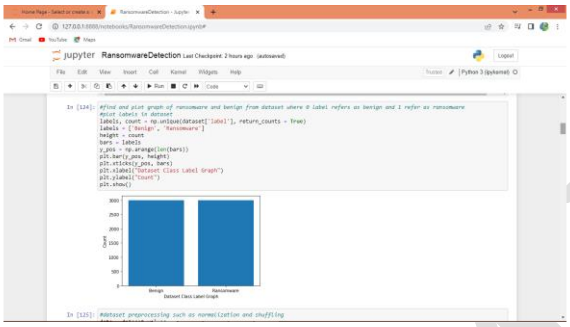
In above screen loading all packages and classes



```
In [122]: #define minmax object for features normalization
          scaler = MinMaxScaler(feature_range = (0, 1)) #use to normalize training data

In [123]: #load and display dataset values
          dataset = pd.read_csv("Dataset/hpc_io_data.csv")
          dataset.fillna(0, inplace = True)#replace missing values
          dataset
Out[123]:
```

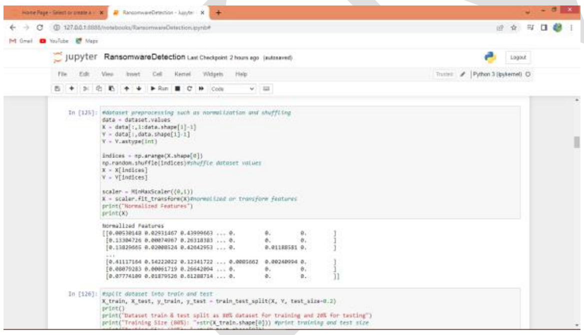
	instructions	LLC-stores	L1-icache-load-misses	branch-load-misses	node-load-misses	rd_req	rd_bytes	wr_req	wr_bytes	flush_operations	rd_total_times	wr_total_times	flush_total_times
0	77558160.0	8575.0	257517.0	215949.0	0.0	0	0	0	147456	4	0	3596349	4524778
1	32081037.0	16800.0	787990.0	140417.0	2.0	0	0	0	0	0	0	0	0
2	11049222.0	5302.0	264089.0	55819.0	0.0	0	0	0	0	0	0	0	0
3	4968323.0	5252.0	188962.0	34310.0	0.0	0	0	0	0	0	0	0	0
4	15314400.0	11345.0	601098.0	112428.0	0.0	0	0	0	0	0	0	0	0
...	...	...	...	...	...	...	...	...	...	...	...	...	...
5995	58694891.0	1627.0	9147.0	460151.0	0.0	0	0	0	0	0	0	0	0
5996	51471771.0	803.0	3740.0	521066.0	0.0	0	0	0	0	0	0	0	0
5997	49304168.0	5280.0	12560.0	480046.0	0.0	0	0	2	20672	1	0	503695	2094439
5998	55966764.0	13175.0	53589.0	529373.0	0.0	0	0	0	0	0	0	0	0
5999	50258111.0	5596.0	58048.0	526161.0	3.0	0	0	0	0	0	0	0	0

In above screen defining class to normalize features and then loading and displaying dataset values



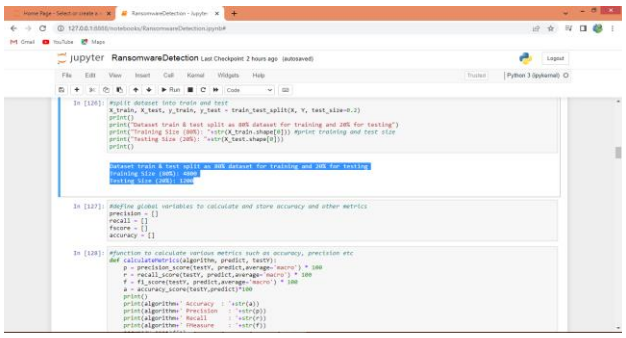
```
In [124]: #find and plot graph of ransomware and benign from dataset where 0 label refers as benign and 1 refer as ransomware
          #plot labels in dataset
          labels, count = np.unique(dataset['label'], return_counts = True)
          labels = ['Benign', 'Ransomware']
          height = count
          bars = labels
          y_pos = np.arange(len(bars))
          plt.bar(y_pos, height)
          plt.xticks(y_pos, bars)
          plt.xlabel("Dataset Class Label Graph")
          plt.ylabel("Count")
          plt.show()

In [125]: #dataset preprocessing such as normalization and shuffling
```

In above screen we are plotting graph of benign and Ransomware dataset size where x-axis represents class label and y-axis represents count



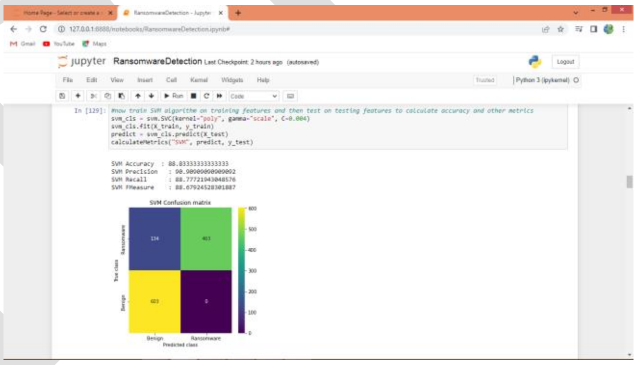
```
In [125]: #dataset preprocessing such as normalization and shuffling
          data = dataset.values
          X = data[:,1:data.shape[1]-1]
          Y = data[:,data.shape[1]-1]
          Y = Y.astype(int)

          indices = np.arange(X.shape[0])
          np.random.shuffle(indices)#shuffle dataset values
          X = X[indices]
          Y = Y[indices]

          scaler = MinMaxScaler((0,1))
          X = scaler.fit_transform(X)#normalized or transform features
          print("Normalized Features")
          print(X)

Normalized Features
[[0.00530148 0.02931467 0.43999663 ... 0.         0.         0.        ]
 [0.13304726 0.00074967 0.26318383 ... 0.         0.         0.        ]
 [0.13829665 0.02008524 0.42642953 ... 0.         0.01188581 0.        ]
 ...
 [0.41117164 0.54222022 0.12341722 ... 0.0005662  0.00240994 0.        ]
 [0.08079283 0.00061719 0.26642094 ... 0.         0.         0.        ]
 [0.07774109 0.01879526 0.61288714 ... 0.         0.         0.        ]]

In [126]: #split dataset into train and test
          X_train, X_test, y_train, y_test = train_test_split(X, Y, test_size=0.2)
          print()
          print("Dataset train & test split as 80% dataset for training and 20% for testing")
          print("Training Size (80%): "+str(X_train.shape[0])) #print training and test size
```

In above screen performing dataset pre-processing such as normalizing and shuffling and then displaying processed values



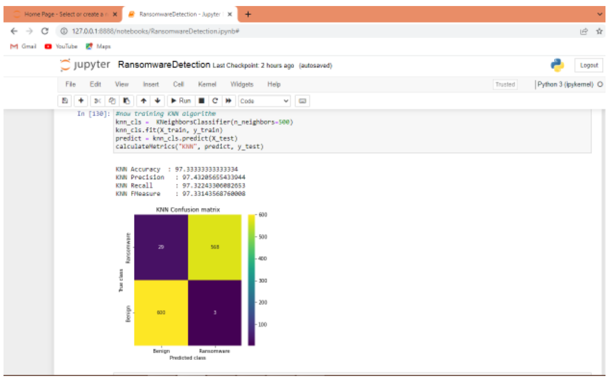
```
In [126]: #split dataset into train and test
          X_train, X_test, y_train, y_test = train_test_split(X, Y, test_size=0.2)
          print()
          print("Dataset train & test split as 80% dataset for training and 20% for testing")
          print("Training Size (80%): "+str(X_train.shape[0])) #print training and test size
          print("Testing Size (20%): "+str(X_test.shape[0]))
          print()

          Dataset train & test split as 80% dataset for training and 20% for testing
          Training Size (80%): 4800
          Testing Size (20%): 1200

In [127]: #define global variables to calculate and store accuracy and other metrics
          precision = []
          recall = []
          fscore = []
          accuracy = []

In [128]: #function to calculate various metrics such as accuracy, precision etc
          def calculateMetrics(algorithm, predict, testY):
              p = precision_score(testY, predict,average='macro') * 100
              r = recall_score(testY, predict,average='macro') * 100
              f = f1_score(testY, predict,average='macro') * 100
              a = accuracy_score(testY,predict)*100
              print()
              print(algorithm+' Accuracy  : '+str(a))
              print(algorithm+' Precision   : '+str(p))
              print(algorithm+' Recall      : '+str(r))
              print(algorithm+' FMeasure    : '+str(f))
```

In above screen splitting dataset into train and test where application using 80% dataset for training and 20% for testing and then defined function to calculate accuracy and other metrics



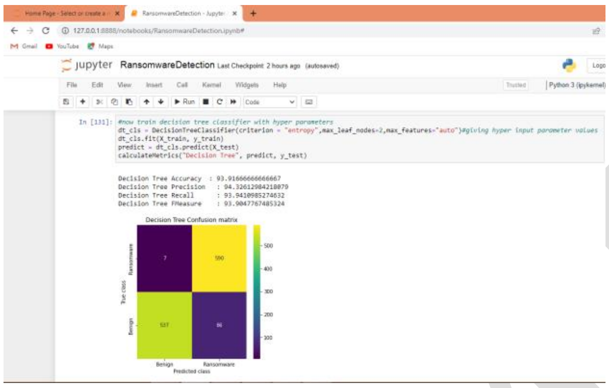
```
In [129]: #now train SVM algorithm on training features and then test on testing features to calculate accuracy and other metrics
          svm_cls = svm.SVC(kernel="poly", gamma="scale", C=0.004)
          svm_cls.fit(X_train, y_train)
          predict = svm_cls.predict(X_test)
          calculateMetrics("SVM", predict, y_test)

          SVM Accuracy  : 88.83333333333333
          SVM Precision   : 90.90909090909092
          SVM Recall      : 88.77721943048576
          SVM FMeasure    : 88.67924528301887
```

In above screen training SVM algorithm on 80% training data and then performing prediction on 20% test data and then calculating accuracy and other metrics. In above screen SVM got 88% accuracy and can see other metrics also and in confusion matrix graph x-axis represents Predicted Labels and y-axis represents True Labels where yellow and green boxes contains correct prediction count and blue boxes represents incorrect prediction count.



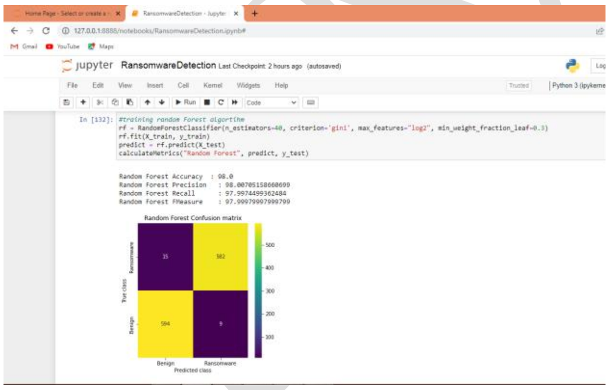
```
In [130]: #now training KNN algorithm
          knn_cls =  KNeighborsClassifier(n_neighbors=500)
          knn_cls.fit(X_train, y_train)
          predict = knn_cls.predict(X_test)
          calculateMetrics("KNN", predict, y_test)

          KNN Accuracy  : 97.33333333333334
          KNN Precision  : 97.43205655433044
          KNN Recall     : 97.32243306082653
          KNN FMeasure   : 97.33143568760008
```

In above screen KNN got 97% accuracy



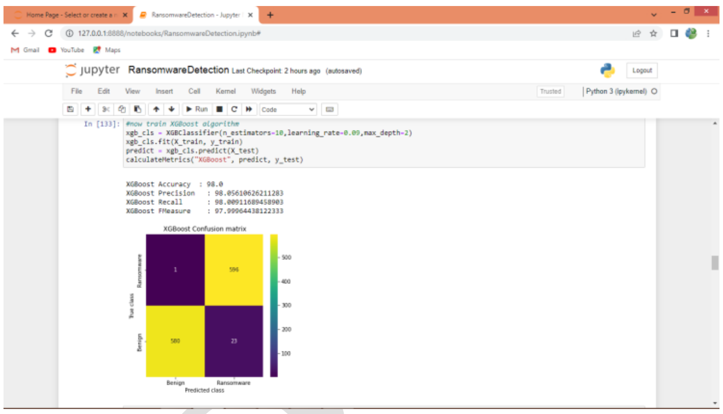
```
In [131]: #now train decision tree classifier with hyper parameters
          dt_cls = DecisionTreeClassifier(criterion = "entropy",max_leaf_nodes=2,max_features="auto")#giving hyper input parameter values
          dt_cls.fit(X_train, y_train)
          predict = dt_cls.predict(X_test)
          calculateMetrics("Decision Tree", predict, y_test)

          Decision Tree Accuracy  : 93.91666666666667
          Decision Tree Precision  : 94.32612984218079
          Decision Tree Recall     : 93.9410985274632
          Decision Tree FMeasure   : 93.9047767485324
```

In above screen decision tree got 93% accuracy



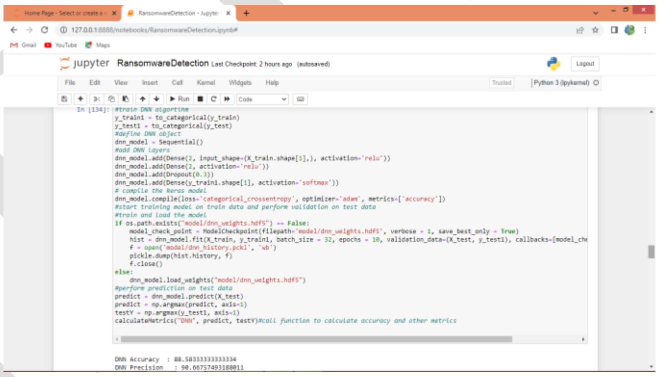
```
In [132]: #training random Forest algortihm
          rf = RandomForestClassifier(n_estimators=40, criterion='gini', max_features="log2", min_weight_fraction_leaf=0.3)
          rf.fit(X_train, y_train)
          predict = rf.predict(X_test)
          calculateMetrics("Random Forest", predict, y_test)

          Random Forest Accuracy  : 98.0
          Random Forest Precision  : 98.00705158660699
          Random Forest Recall     : 97.9974499362484
          Random Forest FMeasure   : 97.99979997999799
```

In above screen Random Forest got 98% accuracy



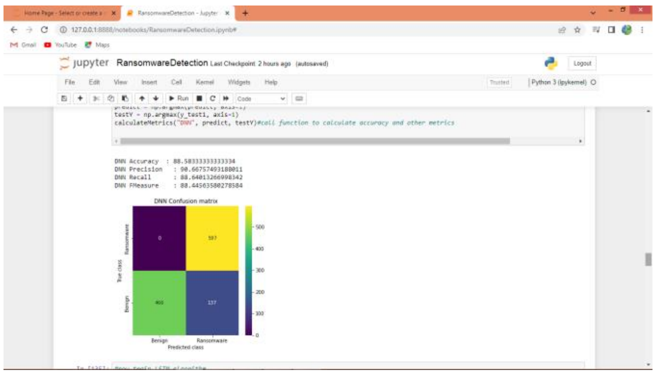
```
In [133]: #now train XGBoost algorithm
          xgb_cls = XGBClassifier(n_estimators=10,learning_rate=0.09,max_depth=2)
          xgb_cls.fit(X_train, y_train)
          predict = xgb_cls.predict(X_test)
          calculateMetrics("XGBoost", predict, y_test)

          XGBoost Accuracy  : 98.0
          XGBoost Precision  : 98.05610626211283
          XGBoost Recall     : 98.00911689458903
          XGBoost FMeasure   : 97.99964438122333
```

In above screen XGBOOST also got 98% accuracy



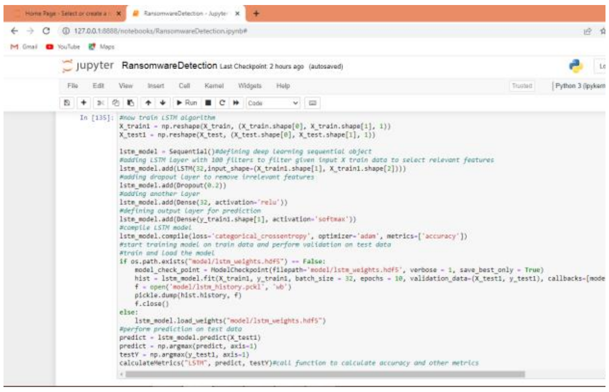
```
In [134]: #train DNN algortihm
          y_train1 = to_categorical(y_train)
          y_test1 = to_categorical(y_test)
          #define DNN object
          dnn_model = Sequential()
          #add DNN layers
          dnn_model.add(Dense(2, input_shape=(X_train.shape[1],), activation='relu'))
          dnn_model.add(Dense(2, activation='relu'))
          dnn_model.add(Dropout(0.3))
          dnn_model.add(Dense(y_train1.shape[1], activation='softmax'))
          # compile the keras model
          dnn_model.compile(loss='categorical_crossentropy', optimizer='adam', metrics=['accuracy'])
          #start training model on train data and perform validation on test data
          #train and load the model
          if os.path.exists("model/dnn_weights.hdf5") == False:
              model_check_point = ModelCheckpoint(filepath='model/dnn_weights.hdf5', verbose = 1, save_best_only = True)
              hist = dnn_model.fit(X_train, y_train1, batch_size = 32, epochs = 10, validation_data=(X_test, y_test1), callbacks=[model_che
              f = open('model/dnn_history.pckl', 'wb')
              pickle.dump(hist.history, f)
              f.close()
          else:
              dnn_model.load_weights("model/dnn_weights.hdf5")
          #perform prediction on test data
          predict = dnn_model.predict(X_test)
          predict = np.argmax(predict, axis=1)
          testY = np.argmax(y_test1, axis=1)
          calculateMetrics("DNN", predict, testY)#call function to calculate accuracy and other metrics

          DNN Accuracy  : 88.58333333333334
          DNN Precision  : 90.66757493188011
```

In above screen training DNN algorithm and below is the DNN output



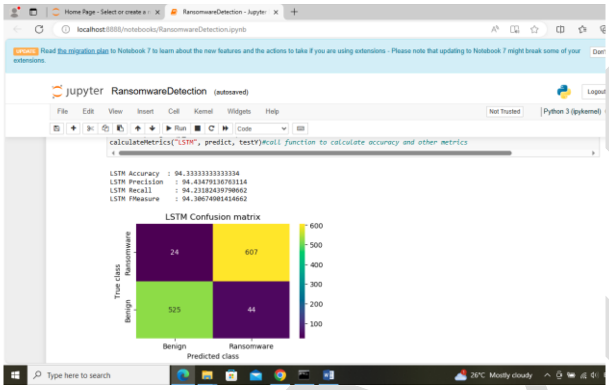
```
          testY = np.argmax(y_test1, axis=1)
          calculateMetrics("DNN", predict, testY)#call function to calculate accuracy and other metrics

          DNN Accuracy  : 88.58333333333334
          DNN Precision  : 90.66757493188011
          DNN Recall     : 88.64013266998342
          DNN FMeasure   : 88.44563580278584
```

In above screen DNN got 88%accuracy



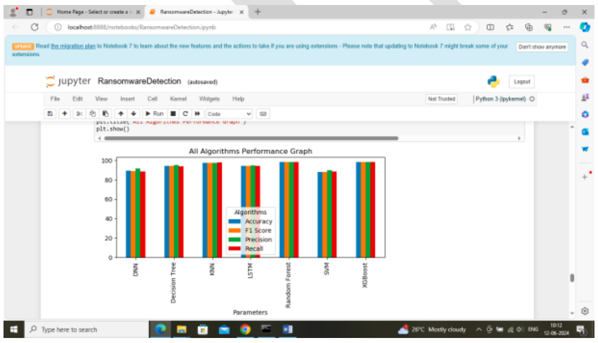
```
In [135]: #now train LSTM algorithm
X_train1 = np.reshape(X_train, (X_train.shape[0], X_train.shape[1], 1))
X_test1 = np.reshape(X_test, (X_test.shape[0], X_test.shape[1], 1))

lstm_model = Sequential()#defining deep learning sequential object
#adding LSTM layer with 100 filters to filter given input X train data to select relevant features
lstm_model.add(LSTM(32,input_shape=(X_train1.shape[1], X_train1.shape[2])))
#adding dropout layer to remove irrelevant features
lstm_model.add(Dropout(0.2))
#adding another layer
lstm_model.add(Dense(32, activation='relu'))
#defining output layer for prediction
lstm_model.add(Dense(y_train1.shape[1], activation='softmax'))
#compile LSTM model
lstm_model.compile(loss='categorical_crossentropy', optimizer='adam', metrics=['accuracy'])
#start training model on train data and perform validation on test data
#train and load the model
if os.path.exists("model/lstm_weights.hdf5") == False:
    model_check_point = ModelCheckpoint(filepath='model/lstm_weights.hdf5', verbose = 1, save_best_only = True)
    hist = lstm_model.fit(X_train1, y_train1, batch_size = 32, epochs = 10, validation_data=(X_test1, y_test1), callbacks=[mode
    f = open('model/lstm_history.pckl', 'wb')
    pickle.dump(hist.history, f)
    f.close()
else:
    lstm_model.load_weights("model/lstm_weights.hdf5")
#perform prediction on test data
predict = lstm_model.predict(X_test1)
predict = np.argmax(predict, axis=1)
testY = np.argmax(y_test1, axis=1)
calculateMetrics("LSTM", predict, testY)#call function to calculate accuracy and other metrics
```

In above screen training LSTM algorithm and after executing above block will get below output

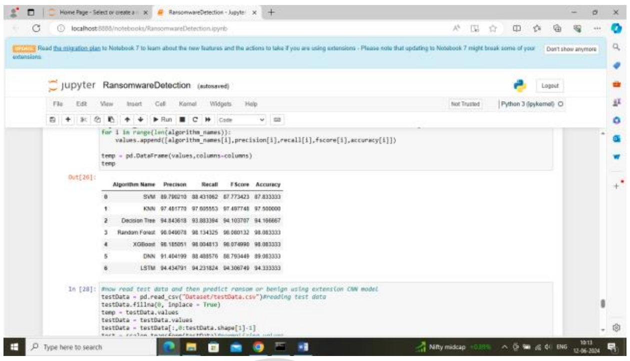


```
calculateMetrics("LSTM", predict, testY)#call function to calculate accuracy and other metrics

LSTM Accuracy  : 94.33333333333334
LSTM Precision  : 94.43479136763114
LSTM Recall     : 94.23182439790662
LSTM FMeasure   : 94.30674901414662
```

In above screen LSTM got 94% accuracy



In above graph x-axis represents algorithm names and y-axis represents accuracy and other metrics in different colour bars and in all algorithms XGBOOST and Random Forest got high accuracy



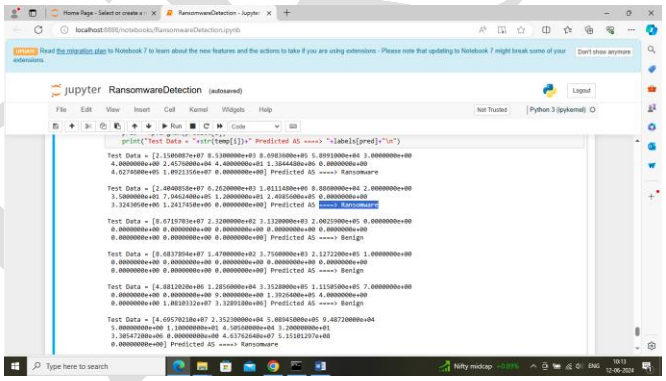
```
for i in range(len(algorithm_names)):
    values.append([algorithm_names[i],precision[i],recall[i],fscore[i],accuracy[i]])

temp = pd.DataFrame(values,columns=columns)
temp
```

Out[26]:

	Algorithm Name	Precison	Recall	F Score	Accuracy
0	SVM	89.790210	88.431062	87.773423	87.833333
1	KNN	97.481770	97.605553	97.497748	97.500000
2	Decision Tree	94.843618	93.883394	94.103707	94.166667
3	Random Forest	98.049078	98.134325	98.080132	98.083333
4	XGBoost	98.185051	98.004813	98.074990	98.083333
5	DNN	91.404199	88.488576	88.793449	89.083333
6	LSTM	94.434791	94.231824	94.306749	94.333333

```
In [28]: #now read test data and then predict ransom or benign using extension CNN model
testData = pd.read_csv("Dataset/testData.csv")#reading test data
testData.fillna(0, inplace = True)
temp = testData.values
testData = testData.values
testData = testData[:,0:testData.shape[1]-1]
```

In above screen displaying all algorithms performance in tabular format



```
    print("Test Data = "+str(temp[i])+" Predicted AS ====> "+labels[pred]+"\n")
Test Data = [2.1506087e+07 8.5300000e+03 8.6983600e+05 5.8991000e+04 3.0000000e+00
 4.0000000e+00 2.4576000e+04 4.4000000e+01 1.3844480e+06 0.0000000e+00
 4.6274600e+05 1.0921356e+07 0.0000000e+00] Predicted AS ====> Ransomware

Test Data = [2.4040858e+07 6.2620000e+03 1.0111480e+06 8.8860000e+04 2.0000000e+00
 3.5000000e+01 7.9462400e+05 1.2000000e+01 2.4985600e+05 0.0000000e+00
 3.3243050e+06 1.2417450e+06 0.0000000e+00] Predicted AS ====> Ransomware

Test Data = [8.6719703e+07 2.3200000e+02 3.1320000e+03 2.0025900e+05 0.0000000e+00
 0.0000000e+00 0.0000000e+00 0.0000000e+00 0.0000000e+00 0.0000000e+00
 0.0000000e+00 0.0000000e+00 0.0000000e+00] Predicted AS ====> Benign

Test Data = [8.6837894e+07 1.4700000e+02 3.7560000e+03 2.1272200e+05 1.0000000e+00
 0.0000000e+00 0.0000000e+00 0.0000000e+00 0.0000000e+00 0.0000000e+00
 0.0000000e+00 0.0000000e+00 0.0000000e+00] Predicted AS ====> Benign

Test Data = [4.8812020e+06 1.2856000e+04 3.3528000e+05 1.1150500e+05 7.0000000e+00
 0.0000000e+00 0.0000000e+00 9.0000000e+00 1.3926400e+05 4.0000000e+00
 0.0000000e+00 1.0810332e+07 3.3289180e+06] Predicted AS ====> Benign

Test Data = [4.69570210e+07 2.35230000e+04 5.08945000e+05 9.48720000e+04
 5.00000000e+00 1.10000000e+01 4.50560000e+04 3.20000000e+01
 3.30547200e+06 0.00000000e+00 4.63762640e+07 5.15101297e+08
 0.00000000e+00] Predicted AS ====> Ransomware
```

In above screen reading test data and then using LSTM Model algorithm object performing prediction on test data and then in output before arrow symbol =➔ we can see Test data values and after arrow symbol we can see predicted values as ‘Ransomware or Benign’.

Conclusion

This study presents a novel and efficient approach for ransomware detection using Hardware Performance Counters (HPCs) and IO Events collected through a virtualized environment. By leveraging machine learning and deep learning algorithms, the system accurately classifies program behavior with minimal impact on system performance. The experimental results, particularly those achieved by Random Forest and XGBoost, demonstrate that low-level hardware and IO features offer significant potential for real-time ransomware detection. The proposed method addresses the shortcomings of traditional antivirus and high-overhead monitoring techniques, offering a lightweight, scalable, and highly accurate solution for modern cybersecurity threats.

References

1. Dinaburg, A., Royal, P., Sharif, M., & Lee, W. (2012). *Ether: Malware analysis via hardware virtualization*. Proceedings of the 15th ACM Conference on Computer and Communications Security.
2. Demme, J., Maycock, M., Schmitz, J., Tang, A., Waksman, A., Sethumadhavan, S., & Stolfo, S. J. (2013). *On the feasibility of online malware detection with performance counters*. ACM SIGARCH Computer Architecture News, 41(3), 559–570.
3. Shafiq, M., Tian, Z., Bashir, A. K., Du, X., & Guizani, M. (2018). *Network traffic classification techniques and comparative analysis using machine learning algorithms*. IEEE Access, 6, 27831–27850.
4. Sgandurra, D., Muñoz-González, L., Mohsen, R., & Lupu, E. C. (2016). *Automated dynamic analysis of ransomware: A survey and new observations*. arXiv preprint arXiv:1609.03020.
5. Mohaisen, A., & Alrawi, O. (2017). *Unveiling Zeus: Automated classification of malware samples*. In Proceedings of the 22nd International Conference on Network Protocols (ICNP).
6. Shijo, P., & Salim, A. (2015). *Integrated static and dynamic analysis for malware detection*. Procedia Computer Science, 46, 804–811.
7. Rehman, M. H. U., Salah, K., Damiani, E., & Svetinovic, D. (2019). *Trust-based secure resource management for edge-cloud computing environments*. Future Generation Computer Systems, 102, 247–261.
8. Kolosnjaji, B., Zarras, A., Webster, G., & Eckert, C. (2016). *Deep learning for classification of malware system call sequences*. In Australasian Joint Conference on Artificial Intelligence (pp. 137–149). Springer.
9. Anderson, B., & McGrew, D. (2016). *Machine learning for encrypted malware traffic classification: Accounting for noisy labels and non-stationarity*. In Proceedings of the 23rd ACM SIGKDD.
10. Nari, S., & Ghorbani, A. A. (2013). *Automated malware classification based on network behavior*. In Proceedings of the International Conference on Computing, Networking and Communications.