IJCRT.ORG ISSN : 2320-2882

INTERNATIONAL JOURNAL OF CREATIVE RESEARCH THOUGHTS (IJCRT)

An International Open Access, Peer-reviewed, Refereed Journal

SCREENING OF RANSOMWARE ATTACKS USING DISK USAGE AND PROCESSOR DATA

[1]Parmati Sreelesh Reddy, [2]Shaik Yaseen Pasha, [3]Kovvuri Abhishek Reddy, [4]Deepa Panse
[1]Student, [2]Student, [3]Student, [4]Assistant Professor
[1]Sreenidhi Institute of Science and Technology,
[2]Sreenidhi Institute of Science and Technology,
[3]Sreenidhi Institute of Science and Technology,
[4]Sreenidhi Institute of Science and Technology

Abstract: This project addresses the challenge of ransomware detection, involving process monitoring and data analysis. The aim is to develop a robust and practical detection method for ransomware executed on a virtualized computer (VM). For the whole VM, data gathering concentrates on particular CPU and disk I/O events.Leveraging machine learning (ML), particularly a random forest (RF), the project aims to create an effective detection model. The proposed method demonstrates resilience to variations in user workloads, overcoming a common challenge in ransomware detection. By avoiding continuous monitoring of every process on the target machine, the model remains adaptable to different user scenarios. The project's effectiveness is measured across various user workloads and 22 ransomware samples. In this project additional enhancements were introduced, incorporating Convolutional Neural Network 2D (CNN2D) and an ensemble model with a voting classifier to further improve ransomware detection accuracy.

Index terms - Among the index phrases are deep learning, virtual machines, ransomware, hardware performance counters, machine learning, and disk statistics.

1. INTRODUCTION

Malware of the ransomware type encrypts files on a target computer or locks it down, rendering the computer and its contents unusable. Ransomware attackers try to take money from their targets. Nation-state actors may use ransomware as a tactic against the essential infrastructure of its adversaries. Data exfiltration is a common tactic used in these attacks to get victims to either pay a ransom or sell their data on the dark web. In 2022, around 70% of businesses were the target of ransomware attacks [1].

It is projected that ransomware will target an organization, an individual, or a device every two seconds by 2031, up from every eleven seconds in 2021. The damage bill reached $20 billion in 2021, and it is predicted to rise to $265 billion. by 2031 [2].Several scholars have recently investigated the process of identifying ransomware attacks. detection using signatures [3], [4] makes use of the hash values generated by antivirus software for known ransomware.looks for files on the target computer that match the hash values. Nevertheless, such signature-based identification can be evaded by polymorphic and metamorphic variants of previously identified ransomware [4], [5]. Special purpose registers called hardware performance counters (HPCs) count processor and system events for each process or for the entire system. Hundreds of processor and system events, such as the quantity of instructions performed, cache misses, and off-chip memory accesses, can be counted by the contemporary processors. System software optimization and performance analysis are common uses for the data gathered with HPCs. Nonetheless, its application for malware detection has been the subject of numerous recent research projects [9], [10], [11], [12], [13], and [14]. Alam et al. [15] made use of the HPC data that was gathered for every system process. But keeping an eye on a lot of processes is not realistic because it can seriously impair the system's performance. The information was gathered at the machine level by Pundir et al. [7]. Their research, however, is restricted to a single Windows virtual machine (VM) workload; altering a VM's workload (changing the amount of programs) could have a substantial effect on the detection accuracy.

2. LITERATURE SURVEY

Malware poses a serious threat to theInternet, including Trojan horses, worms, and spyware. We found that while malware and its variants can differ significantly from content signatures, they share some higher-level behavioral characteristics that are more accurate in exposing the true intentions of malware. [4] This study looks at the process extracting malware behavior, outlines the formal Malware Behavior Feature (MBF) extraction approach [3, 5, 9, 10, 12], and suggests a malware detection algorithm based on MBFs. In the end, we created and executed an MBF-based malware detection system, and the outcomes of our experiments indicate that it is capable of identifying recently discovered, unidentified malware.on this work, we provide a hardware-assisted method for early crypto-ransomware infection detection on commodity processors, named RanStop. RanStop collects information from hardware performance counters that are included into modern processors' performance monitoring units to examine micro-architectural event sets and detect known and undiscovered forms of crypto-ransomware. In this study, we use the long short-term memory (LSTM) [15] model to train a recurrent neural network-based machine learning architecture for evaluating micro-architectural events in the hardware domain during the execution of both benign programs and numerous ransomware variants. Using data from neighboring HPCs, we build timeseries to provide intrinsic statistical features that enhance RanStop's detection accuracy and lower noise through the use of global average pooling and LSTM [15]. RanStop is an early detection approach that uses HPC data collected for 20 timestamps spaced 100us apart to reliably and swiftly identify ransomware within 2ms of the program execution beginning. It is too early in the detecting process for a ransomware to cause much, if any, harm. Furthermore, validation against innocuous programs that exhibit The behavioral resemblances (sub-routine-centric) to a crypto-ransomware indicate that RanStop has an average accuracy of 97% for detecting ransomware over fifty random trials. The use of micro architecture execution patterns in malware software detection has showed promise in recent publications. Because they detect malware by comparing a program's signature—that is, its pattern of program execution—against the signatures of known malware programs, these detectors are referred to as signature-based detectors. Our paper [10] introduces a new class of hardware malware detectors that are anomaly-based and do not require signatures for malware detection [9], [10], [11], [12], [13], [14]. As a result, they are capable of detecting a wider range of malware, including potentially novel ones. We use unsupervised machine learning to generate profiles of normal program execution based on data from performance counters.We then use these profiles to identify notable program behavior anomalies brought on by malware exploitation. We demonstrate that it is almost probable to identify real-world exploitation of widely used Windows/x86 applications, like Internet Explorer and Adobe PDF Reader. We also look at the limitations and difficulties of applying this strategy against a highly skilled adversary trying to avoid anomaly-based detection. The suggested detector can be used in conjunction with earlier suggested signature-based detectors to increase security.

METHODOLOGY

i) Proposed Work:

The proposed system introduces a novel approach to ransomware detection on virtual machines (VMs). It collects particular disk and processor I/O events for the whole virtual machine from the host computer. Specifically, a random forest (RF) classifier is used in machine learning to create a robust detection model. The suggested method quickly and accurately detects both known and unknown ransomware, with the RF classifier outperforming other tested classifiers. In this paper additional enhancements were introduced, incorporating Convolutional Neural Network 2D (CNN2D) and an ensemble model with a voting classifier to further improve ransomware detection accuracy.

ii) System Architecture:

A unique method for ransomware detection on virtual machines (VMs) is introduced by the proposed system. It gathers certain disk and processor I/O events from the host system for the entire virtual machine. Specifically, a random forest (RF) classifier is used in machine learning to create a robust detection model. Both known and undiscovered malware may be quickly and accurately detected by the suggested technique, with the RF classifier beating other evaluated classifiers. To further improve ransomware detection accuracy, new improvements were incorporated in this paper: an ensemble model with a voting classifier and Convolutional Neural Network 2D (CNN2D).

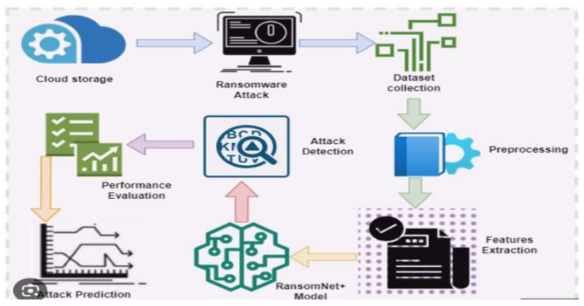
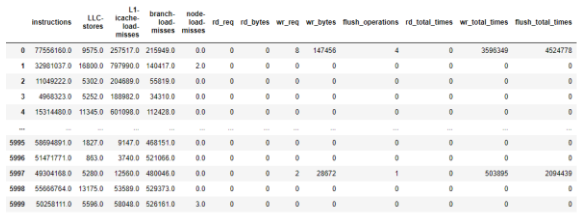


Fig 1 Proposed architecture

iii) Dataset collection:

The records recording processor and disk I/O events during virtual machine execution make up the HPC dataset used in the study. This dataset offers a thorough basis for training and testing ransomware detection models since it has been carefully selected to represent a wide range of system operations. The HPC dataset enables a realistic modeling of probable ransomware behaviors in real-world computing settings by providing both unknown samples for robustness testing and known samples for model calibration [16], [17], [18], and [19].



	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	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
2	11049222.0	5302.0	204689.0	55819.0	0.0	0	0	0	0	0	0	0	0
3	4968323.0	5252.0	188982.0	34310.0	0.0	0	0	0	0	0	0	0	0
4	15314480.0	11345.0	601098.0	112428.0	0.0	0	0	0	0	0	0	0	0
...	...	...	...	...	...	...	...	...	...	...	...	...	...
5995	58694891.0	1827.0	9147.0	468151.0	0.0	0	0	0	0	0	0	0	0
5996	51471771.0	863.0	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	28672	1	0	503895	2094439
5998	55666764.0	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

Fig 2 Dataset

iv) Data Processing:

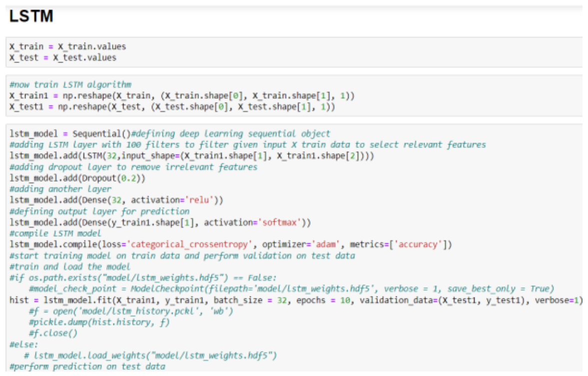
Data processing is the process of turning unprocessed data into information that is useful to organizations. Processed data is typically collected, arranged, cleaned, verified, analyzed, and put into understandable formats like documents or graphs by data scientists. There are three ways to process data: mechanically, electronically, and manually. Enhancing the value of information and making decision-making easier are the goals.

v) Feature selection:

Feature selection is the process of determining which most trustworthy, relevant, and non-redundant features to use while building a model. It isimperative to gradually minimize the size of datasets as their number and diversity rise. One of the fundamental components of feature engineering is feature selection, which is the process of determining which features are most important to incorporate in machine learning algorithms. Feature selection procedures reduce the number of input variables by eliminating superfluous or redundant features and narrowing the set of features to those that are most relevant to the machine learning model.

vi) Algorithms:

Long Short Term Memory (LSTM): Recurrent neural networks (RNNs) of the LSTM type were created to solve the vanishing gradient issue that plagues RNNs in the conventional sense. It is ideal for jobs involving time series or sequential patterns because it adds a memory cell that enables the model to capture long-term dependencies in sequential data. [15] The project probably uses LSTMs because of its capacity to represent and comprehend temporal connections, which is important for ransomware detection because the order in which system events and behaviors occur is important. Over time, LSTMs might pick up subtle trends that improve the model's capacity to identify ransomware activity.



LSTM

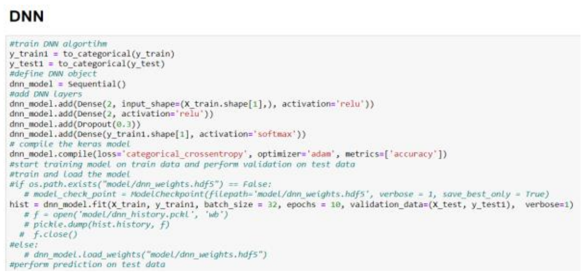
```
X_train = X_train.values
X_test = X_test.values
```

```
#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), verbose=1)
    #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
```

Fig 3 LSTM

Deep Neural Network (DNN An artificial neural network with several hidden layers sandwiched a deep neural network is defined as having layers between the input and output.These networks are appropriate for difficult tasks requiring feature abstraction and representation because they can learn complicated hierarchical representations of data. Because of its ability to discover complex features and relationships within the gathered data, DNNs may be used in the project. DNNs can offer a potent framework for feature extraction and learning high-level representations in ransomware detection, where nuanced and complicated patterns may emerge [8], [13], [14].

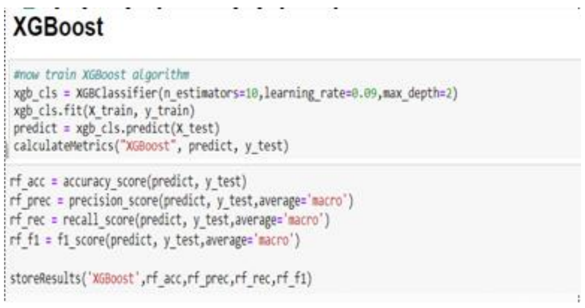


DNN

```
#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),  verbose=1)
   # 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
```

Fig 4 DNN

XGBoost: The Extreme Gradient Boosting (XGBoost) machine learning technique is a member of the gradient boosting technique family. By building an ensemble of weak learners (usually decision trees) one after the other, each tree fixes the mistakes of the one before it, creating a more reliable and accurate model. XGBoost is probably included because of how well it performs on classification tasks and with big datasets. Strong predictive skills are something that XGBoost may provide in the context of ransomware detection, helping to create an accurate detection model by efficiently collecting the various aspects of ransomware activities [8], [13], and [14].



XGBoost

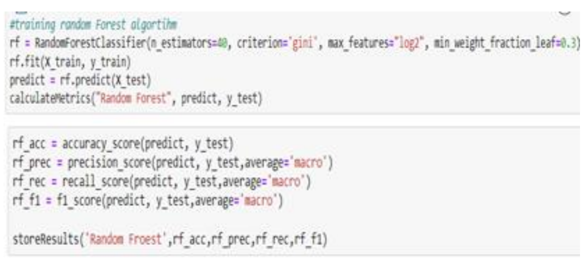
```
#now train XGBoost algorithm
xgb_cls = XGBClassifier(n_estimators=10,learning_rate=0.09,max_depth=2)
xgb_cls.fit(X_train, y_train)
predict = xgb_cls.predict(X_test)
calculateMetrics("XGBoost", predict, y_test)
```

```
rf_acc = accuracy_score(predict, y_test)
rf_prec = precision_score(predict, y_test,average='macro')
rf_rec = recall_score(predict, y_test,average='macro')
rf_f1 = f1_score(predict, y_test,average='macro')

storeResults('XGBoost',rf_acc,rf_prec,rf_rec,rf_f1)
```

Fig 5 Xgboost

Random Forest: During training, the Random Forest algorithm for ensemble learning creates a large number of decision trees. It produces the mean prediction (regression) of each individual tree or the mode of the classes (classification). Use in the Project: Because Random Forest can handle difficult classification problems, it is used in the project. By integrating the advantages of several decision trees, a Random Forest ensemble can improve accuracy in ransomware detection, where there may be a variety of patterns. This results in a strong and dependable model.



```
#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)
```

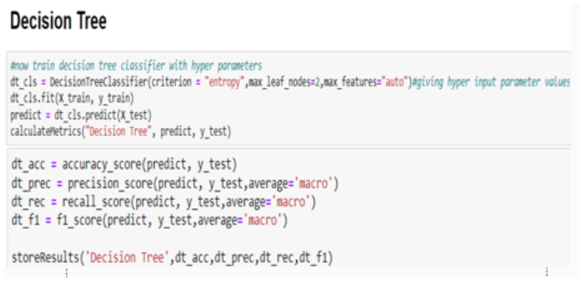
```
rf_acc = accuracy_score(predict, y_test)
rf_prec = precision_score(predict, y_test,average='macro')
rf_rec = recall_score(predict, y_test,average='macro')
rf_f1 = f1_score(predict, y_test,average='macro')

storeResults('Random Froest',rf_acc,rf_prec,rf_rec,rf_f1)
```

Fig 6 Random forest

Decision Tree: This is a model that resembles a tree in which each node denotes a choice made in response to input features. The dataset is recursively divided into subsets, producing terminal nodes that represent the chosen course of action or forecast. Decision trees are used to describe decision-making processes because they are easy to interpret and straightforward. Decision Trees can be used to shed light on the series of actions that lead to a decision in the context of ransomware detection, which can help identify the elements that influence the detection result.

.



Decision Tree

```
#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)
```

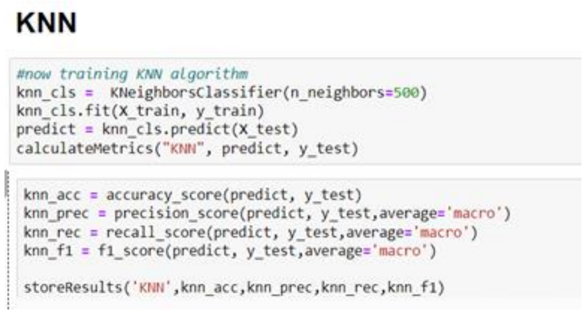
```
dt_acc = accuracy_score(predict, y_test)
dt_prec = precision_score(predict, y_test,average='macro')
dt_rec = recall_score(predict, y_test,average='macro')
dt_f1 = f1_score(predict, y_test,average='macro')

storeResults('Decision Tree',dt_acc,dt_prec,dt_rec,dt_f1)
```

Fig 7 Decision tree

K – Nearest Neighbor (KNN):

KNN is a supervised machine learning method that can be used for classification and regression issues.
A fresh data point is categorized using a majority class vote, or its value is predicted by summing the values of its k closest neighbors in the feature space. KNN is probably employed because of how well it can identify local patterns in the data and how easy it is to use. KNN can offer a versatile method for locating comparable patterns in the dataset when it comes to ransomware detection, where there may be minute differences.



KNN

```
#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)
```

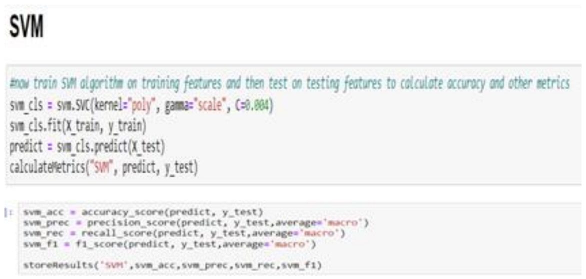
```
knn_acc = accuracy_score(predict, y_test)
knn_prec = precision_score(predict, y_test,average='macro')
knn_rec = recall_score(predict, y_test,average='macro')
knn_f1 = f1_score(predict, y_test,average='macro')

storeResults('KNN',knn_acc,knn_prec,knn_rec,knn_f1)
```

Fig 8 KNN

Support Vector Machine (SVM):

One supervised machine learning method that can be used for both regression and classification issues is support vector machines (SVM). It maximizes the margin between classes by locating a hyperplane that optimally divides data into distinct classes or forecasts a continuous result. Because SVM can handle high-dimensional data and identify ideal decision boundaries, it is used. SVM can offer a reliable technique for efficient classification in ransomware detection, where feature spaces can be complicated, by defining distinct decision limits.



SVM

```
#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)
```

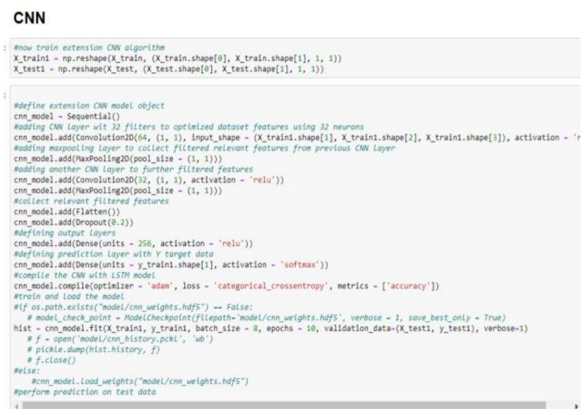
```
svm_acc = accuracy_score(predict, y_test)
svm_prec = precision_score(predict, y_test,average='macro')
svm_rec = recall_score(predict, y_test,average='macro')
svm_f1 = f1_score(predict, y_test,average='macro')

storeResults('SVM',svm_acc,svm_prec,svm_rec,svm_f1)
```

Fig 9 SVM

2D Convolutional Neural Network (CNN2D):

CNN2D is a deep learning technique used mostly for image analysis that is intended to interpret data that resembles a grid. This study, however, adapts CNN2D to handle continuous data by automatically learning hierarchical features and patterns through the use of convolutional layers. Because CNN2D can automatically extract complicated characteristics from continuous data, it is used. CNN2D can capture complex information, which adds to a more successful detection model in the context of ransomware detection, where patterns in the sequential and continuous system operations are vital.



CNN

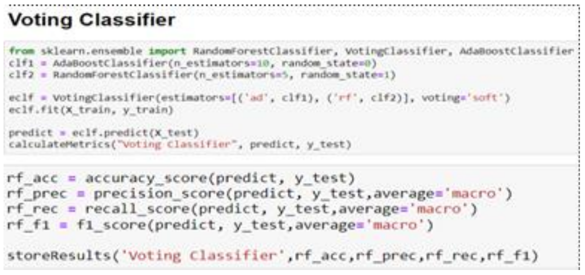
```
#now train extension CNN algorithm
X_train1 = np.reshape(X_train, (X_train.shape[0], X_train.shape[1], 1, 1))
X_test1 = np.reshape(X_test, (X_test.shape[0], X_test.shape[1], 1, 1))
```

```
#define extension CNN model object
cnn_model = Sequential()
#adding CNN layer wit 32 filters to optimized dataset features using 32 neurons
cnn_model.add(Convolution2D(64, (1, 1), input_shape = (X_train1.shape[1], X_train1.shape[2], X_train1.shape[3]), activation = 'r
#adding maxpooling layer to collect filtered relevant features from previous CNN layer
cnn_model.add(MaxPooling2D(pool_size = (1, 1)))
#adding another CNN layer to further filtered features
cnn_model.add(Convolution2D(32, (1, 1), activation = 'relu'))
cnn_model.add(MaxPooling2D(pool_size = (1, 1)))
#collect relevant filtered features
cnn_model.add(Flatten())
cnn_model.add(Dropout(0.2))
#defining output layers
cnn_model.add(Dense(units = 256, activation = 'relu'))
#defining prediction layer with Y target data
cnn_model.add(Dense(units = y_train1.shape[1], activation = 'softmax'))
#compile the CNN with LSTM model
cnn_model.compile(optimizer = 'adam', loss = 'categorical_crossentropy', metrics = ['accuracy'])
#train and load the model
#if os.path.exists("model/cnn_weights.hdf5") == False:
   # model_check_point = ModelCheckpoint(filepath='model/cnn_weights.hdf5', verbose = 1, save_best_only = True)
hist = cnn_model.fit(X_train1, y_train1, batch_size = 8, epochs = 10, validation_data=(X_test1, y_test1), verbose=1)
   # f = open('model/cnn_history.pckl', 'wb')
   # pickle.dump(hist.history, f)
   # f.close()
#else:
    #cnn_model.load_weights("model/cnn_weights.hdf5")
#perform prediction on test data
```

Fig 10 CNN2D

Voting Classifier:

A voting classifier uses either majority voting or average to integrate predictions from several different separate models. It is used to leverage many algorithms in an effort to enhance the model's overall performance. Predictions from different algorithms are combined by the Voting Classifier. By taking into account various viewpoints from several models, this ensemble technique increases the overall robustness and accuracy of the ransomware detection system.



Voting Classifier

```
from sklearn.ensemble import RandomForestClassifier, VotingClassifier, AdaBoostClassifier
clf1 = AdaBoostClassifier(n_estimators=10, random_state=0)
clf2 = RandomForestClassifier(n_estimators=5, random_state=1)

eclf = VotingClassifier(estimators=[('ad', clf1), ('rf', clf2)], voting='soft')
eclf.fit(X_train, y_train)

predict = eclf.predict(X_test)
calculateMetrics("Voting Classifier", predict, y_test)
```

```
rf_acc = accuracy_score(predict, y_test)
rf_prec = precision_score(predict, y_test,average='macro')
rf_rec = recall_score(predict, y_test,average='macro')
rf_f1 = f1_score(predict, y_test,average='macro')

storeResults('Voting Classifier',rf_acc,rf_prec,rf_rec,rf_f1)
```

Fig 11 Voting classifier

1. EXPERIMENTAL RESULTS

Precision: The percentage of accurately classified samples or instances among the positive samples is measured as precision. As a result, the formula to calculate the precision is as follows:

TP/(TP + FP) = True positives/(True positives + False positives)
is the formula for precision.

$$\text{Precision} = \frac{\textit{True Positive}}{\textit{True Positive} + \textit{False Positive}}$$

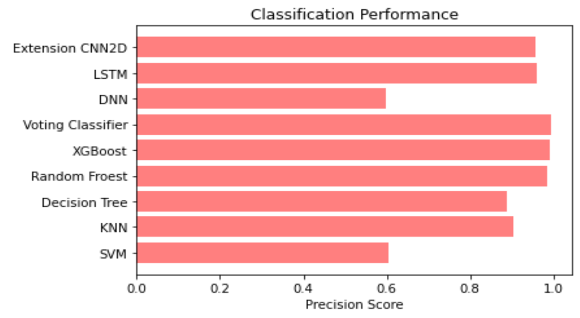


Fig 12 Precision comparison graph

Recall: In machine learning, recall is a statistic that evaluates how well a model can find all relevant examples within a specific class. This ratio, which represents the proportion of accurately predicted positive observations to the total number of genuine positives, indicates how effectively a model represents examples of a given class.

$$Recall = \frac{TP}{TP + FN}$$

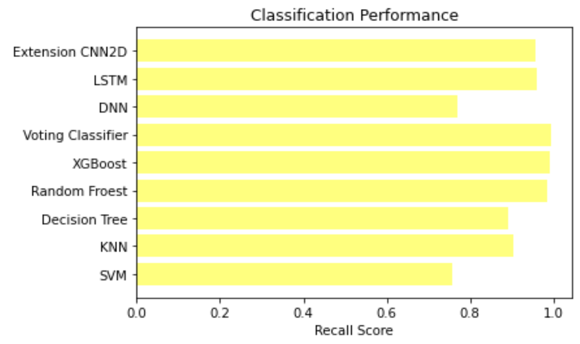


Fig 13 Recall comparison graph

Accuracy: The percentage of right predictions made in a classification exercise is known as accuracy, and it serves as a gauge for how accurate a model is overall.

$$Accuracy = \frac{TP + TN}{TP + FP + TN + FN}$$

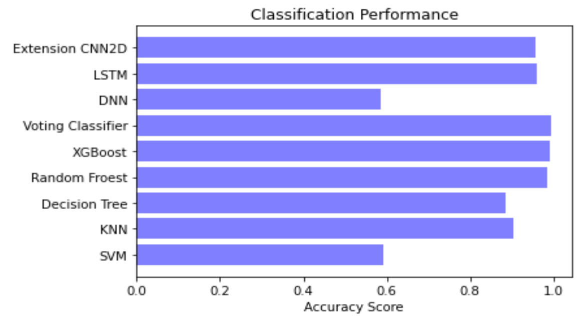


Fig 14 Accuracy graph

F1 Score: Because the F1 Score offers a balanced statistic that accounts for both false positives and false negatives, it is suited for imbalanced datasets. The harmonic mean of recall and precision is used to calculate it.

$$\text{F1 Score} = 2 * \frac{\text{Recall} \times \text{Precision}}{\text{Recall} + \text{Precision}} * 100$$

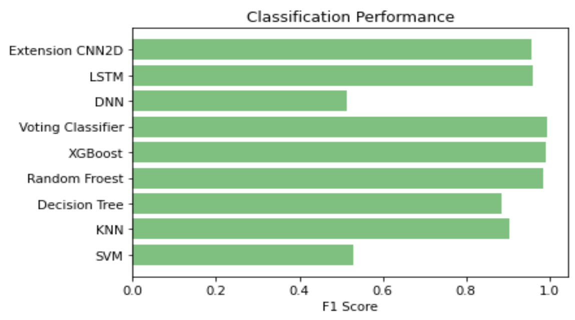
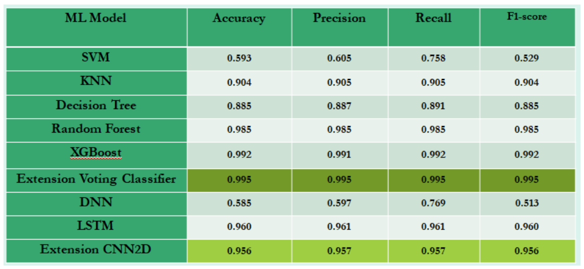


Fig 15 F1Score



ML Model	Accuracy	Precision	Recall	F1-score
SVM	0.593	0.605	0.758	0.529
KNN	0.904	0.905	0.905	0.904
Decision Tree	0.885	0.887	0.891	0.885
Random Forest	0.985	0.985	0.985	0.985
XGBoost	0.992	0.991	0.992	0.992
Extension Voting Classifier	0.995	0.995	0.995	0.995
DNN	0.585	0.597	0.769	0.513
LSTM	0.960	0.961	0.961	0.960
Extension CNN2D	0.956	0.957	0.957	0.956

Fig 16 Performance Evaluation

2. CONCLUSION

Through the use of virtualization technology, hardware performance counters, and IO event data, the project effectively presents a novel way to ransomware detection [9], [10], [11], [12], [13], and [14]. This strategy improves accuracy while minimizing the impact on system performance. The study assesses a number various machine learning algorithms with extensive testing, including SVM, KNN, Random Forest, Decision Tree, XGBOOST, DNN, and LSTM. The findings demonstrate that XGBOOST and Random Forest consistently forecast ransomware activity with excellent accuracy. The study looks into how successful deep learning models are, particularly LSTM and DNN, and provides insightful information about how well they perform relative to conventional machine learning algorithms, expanding the range of predictive techniques.

3. FUTURE SCOPE

Further investigation could delve into assessing the efficacy of the proposed method in identifying novel and emerging ransomware variants, given that the present study concentrated on a mix of both known and unknown ransomware. Extending the project to evaluate how diverse user workloads impact ransomware detection [7] would provide valuable insights, building on the study's demonstrated adaptability to varying workloads. Voting classifier which is extension to the project, has demonstrated exceptional performance with a 99% accuracy in ransomware detection. Rigorous testing on the front end using feature values substantiates its robustness and effectiveness in reliably identifying and mitigating ransomware threats. Implementing and testing the proposed approach in real-world scenarios would offer practical insights into its effectiveness in identifying ransomware attacks within live production environments. The project has the potential to enhance its ransomware detection capabilities by investigating the integration of additional data sources or features into the machine learning model.