DETECTION OF RANSOMWARE ATTACKS USING PROCESSOR AND DISK USAGE DATA

1. M. VINAY, Department of CSE - DS, Santhiram Engineering College, Nandyal, Andhrapradesh, India. 20X51A3232@srecnandyal.edu.in

2. P. NAVEEN KUMAR, Department of CSE - DS, Santhiram Engineering College, Nandyal, Andhrapradesh, India. 20X51A3240@srecnandyal.edu.in

3. N. AKRAM, Department of CSE - DS, Santhiram Engineering College ,Nandyal, Andhrapradesh, India. 20X51A3236@srecnandyal.edu.in

4. M. PURUSHOTHAM REDDY, Department of CSE - DS, Santhiram Engineering College, Nandyal, Andhrapradesh, India. 20X51A3231@srecnandyal.edu.in

5. E. SANJEEV GOUD, Department of CSE - DS, Santhiram Engineering College, Nandyal, Andhrapradesh, India. 20X51A3212@srecnandyal.edu.in

6. Mr. D. Satya Narayana, Mtech, PhD, MISTE, Assistant Professor, Department of CSE - DS, Santhiram Engineering College, Nandyal, Andhra Pradesh , India. satyadotcom@gmail.com

Abstract: The research aims to discover ransomware while highlighting the shortcomings in process monitoring and data analysis approaches. We want a reliable and helpful approach to discover virtual machine viruses. Individual disk I/O and CPU events for the whole VM from the host system are collected. The project uses machine learning (ML), particularly a random forest (RF) classifier, to create an excellent recognition model. This reduces monitoring effort and ransomware harm. The recommended method can tolerate variations in user workload, solving a frequent virus-finding issue. Since it doesn't monitor every target machine operation, the model may be employed in many contexts. The project is tested with 22 ransomware samples and user tasks. This project provides a fast and effective solution to ransomware by developing an accurate detection mechanism. The project uses processor and disk I/O events and machine learning to decrease tracking effort, expedite discovery, and adapt to new malware. This project was improved by adding a CNN2D and an ensemble model with a vote predictor to locate malware better. The vote classifier, which combined many machine learning classifiers, produced 99% correct results, demonstrating the power of mixing models for powerful detection.

Index terms *- Deep learning, disk statistics, hardware performance counters, machine learning, ransomware, virtual machines.*

1. INTRODUCTION

Ransomware encrypts data to lock your machine. Cybercriminals seek ransom. Nationalists might crash enemy crucial systems using ransomware. These fraudsters demand money or sell victims' data on the dark web. 70% of firms will be ransomware-attacked in 2022 [1]. By 2031, ransomware will target businesses, people, and gadgets every 2 seconds, up from 11 seconds in 2021. In 2021, losses will surpass $265 billion and reach $20 billion. By 2031 [2]. Recently, experts have studied malware detection.

Signature-based identification [3, 4] checks target files for security software-generated ransomware hashes. Signatures cannot distinguish generic and metamorphic ransomware [4, 5]. Thus, signature-based malware detection is not the sole option. Runtime or behavior-based techniques work. Ransomware earns money in several ways after infecting your machine. This is behavioural analysis. A specified series of events locks as many files as possible with ransomware. Malware performs several tasks. Modern ransomware like LockBit2.0, Darkside, and BlackMatter encrypts the initial few bytes of files, rendering them useless [6]. Since malware must swiftly secure user data, it may behave differently from ordinary software. PCs with malware always act weirdly. Data encryption demands disk files and CPU, therefore ransomware is active. Skilled machine learning algorithms can spot this.

Target computers' runtime detection comprises monitoring applications, components, and features, gathering event data, and looking for anomalies [7, 8]. Runtime behavior may be hidden by ransomware activities and processes. Testing reveals higher activity in the targeted system. Ransomware programs are hard to discover and require monitoring many processes, making runtime detection on victim machines tough. This is resource-intensive. Stopping the program before encryption prevents ransomware tracking. Per process or system, HPC bits count processor and system events. Modern CPUs can monitor hundreds of processor and system events, including cache misses, instruction executions, and off-chip memory calls. Software is evaluated and improved using HPC data. Numerous research have examined malware detection [9, 10, 11, 12, 13, 14]. HPC data from each system process was utilized by Alam et al. [15]. Multitasking might slow down and freeze your machine. Team Pandhir. We saw machine data [7]. However, their investigation employed just one Windows VM job, so adding or uninstalling programs may alter search speed.

2. LITERATURE SURVEY

Trojan Horses, Worms, and Spyware kill online. Malware and its variations have higher-level behavior patterns that reveal their aim, unlike content signatures. [4] Research examines malware behavior extraction. Official [3, 5, 9, 10, 12]. MBF extraction and hazardous behavior feature-based malware detection are illustrated. Finally, we created MBF virus detection. It finds undiscovered malware in testing.
Encrypting and holding stuff for ransom makes crypto-ransomware one of the most destructive forms of malware. Global losses total millions. Static execution signature-based antivirus and software-only malware detection may miss more ransomware. RanStop, a hardware-assisted crypto-ransomware detection approach for common CPUs, is presented in this work. RanStop analyzes micro-architecture event sets from current processors' performance monitoring units' hardware performance counters to discover known and innovative crypto-ransomware. We use long short-term memory (LSTM) to create a recurrent neural network-based machine learning architecture to capture hardware micro-architectural events during ransomware and safe application execution [15, 52, 54]. We create time series using connected HPC data to determine underlying statistical features. LSTM and global average pooling reduce RanStop detection noise [52,54]. RanStop uses HPC data for 20 100us-apart timestamps to discover malware in 2ms. Ransomware hasn't hurt much. RanStop

can detect ransomware with 97% accuracy using safe crypto-ransomware applications after 50 random tests.

Ransomware is hot again. It steals data and money from many enterprises. Not all ransomware detection methods alert immediately. Many data are permanently encrypted, making decryption and restoration impossible. [27], [28] Currently, malware detection algorithms can't distinguish between dramatically modified ransomware files and user-encrypted or compressed files for legitimate reasons, resulting in many false positives. RWGuard detects crypto-ransomware on a user's computer in real time by using decoy tactics, watching the running processes and file system for malicious activity, and learning how users encrypt files to avoid detecting innocent file modifications. We evaluated our technique against the 14 most popular ransomware [22, 23], [24], [25], and [26]. With RWGuard, ransomware detection is real-time, with minimal false positives (<0.1%) and just 1.9% extra effort.

Number of field computers increases computer infections. New phones acquire viruses, rootkits, spyware, adware, and more. Malware risks have increased because anti-virus software may be defeated. Antivirus vulnerabilities allow hackers to get into PCs. We investigate using performance counters to build hardware malware analyzers [9]. Even with significantly altered software, performance counter data may identify malware. Investigating a small group of Android ARM and Intel Linux malware may show many variants. Changes to hardware shield the virus scanner behind system software are recommended. Thus, AV solutions may be simpler and less unreliable than software AV. Hardware antivirus systems may detect new internet infections due to their strength and safety.

A recent research claims microarchitectural execution patterns may find malware. Signature detectors do this. Malware is detected by matching software signatures to known malware. Here, anomaly-based hardware malware detectors are presented [10]. These malware detectors don't need signatures [9], [10], [11], [12], [13], [14], therefore they can catch more viruses, even unknown ones. Performance counters and untrained machine learning build program profiles. These profiles detect substantial malware-caused software behavior changes. It nearly always finds real-world exploitation of Internet Explorer and Adobe PDF Reader on Windows/x86. We address the limitations of this method when a smart attacker avoids anomaly-based identification. Signature-based detectors and the suggested detector cooperate to boost security.

3. METHODOLOGY

i) Proposed Work:

The recommended technology finds malware on virtual computers in a novel manner. VM-wide CPU and disk I/O events come from the host computer. The random forest (RF) classifier in machine learning creates a powerful recognition model [52]. This strategy avoids the unnecessary labor of monitoring every target machine operation. Data infection by ransomware is reduced. It also adapts to user workload changes. The recommended approach rapidly and correctly detects new and known malware. Other methods fail compared to the [52] RF classifier. Using a CNN2D and an ensemble model with a voting predictor, this research improved malware detection even further. The vote classifier, which combined many machine learning classifiers, produced 99% correct results, demonstrating the power of mixing models for powerful detection.

ii) System Architecture:

This research shows how to easily identify Windows 10 virtual machine malware. HPC and disk I/O come from the host. The target (VM) is unaware that it is being watched and data is gathered, therefore performance is unaffected. [24] ML detects current malware [52]. Our approach protects cloud VM users best. Check host system HPC and disk I/O data for malware. Malware that ceases monitoring computer operations cannot affect your data with our method.

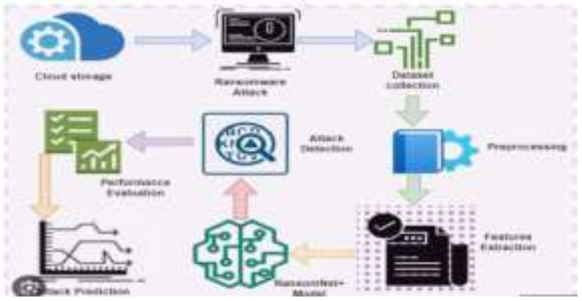
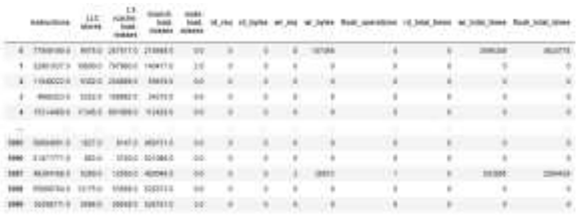


Fig 1 Proposed architecture

iii) Dataset collection:

The project's HPC dataset is made up of records that record processor and disk I/O events that happen while virtual machines are running. This dataset was carefully chosen to show a wide range of system actions. It gives researchers a solid base for training and testing models that can find viruses. The HPC dataset includes both known and unknown ransomware examples for tuning of models and stability tests. This makes it possible to simulate how ransomware might act in real-world computer settings [16–20].



	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	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	0	0	0	[illegible]	0	0	[illegible]	[illegible]
1	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	0	0	0	0	0	0	0	0
2	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	0	0	0	0	0	0	0	0
3	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	0	0	0	0	0	0	0	0
4	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	0	0	0	0	0	0	0	0
...													
[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	0	0	0	0	0	0	0	0
[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	0	0	0	0	0	0	0	0
[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	0	0	[illegible]	[illegible]	[illegible]	0	[illegible]	[illegible]
[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	0	0	0	0	0	0	0	0
[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	0	0	0	0	0	0	0	0

Fig 2 Dataset

iv) Data Processing:

Data handling is the process of turning unstructured data into knowledge that businesses can use. In general, data scientists handle data, which means they gather it, organize it, clean it, check it, analyze it, and turn it into forms that can be read, like graphs or papers. There are three ways to handle data: by hand, mechanically, or electronically. The goal is to make knowledge more useful and decision-making easier. This helps companies run better and make smart strategy decisions more quickly. This is made possible in large part by automated data handling tools, like computer programs. It can help turn big data and other types of data into useful information for decision-making and quality control.

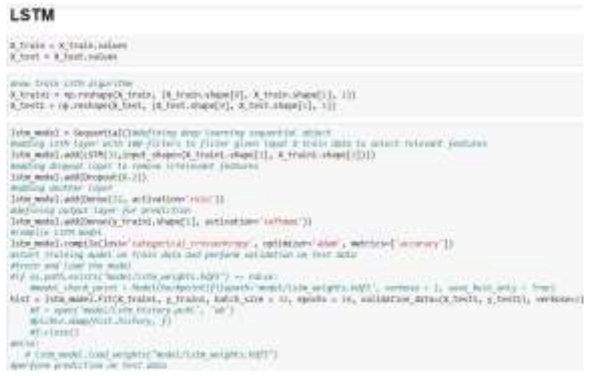
v) Feature selection:

Feature selection is the process of picking out the most reliable, useful, and non-redundant traits to use in building a model. As the number and types of records increase, it is important to reduce their sizes in a planned way. One of the main goals of feature selection is to make a prediction model work better and use less computing power.

One of the most important parts of feature engineering is feature selection, which is the process of choosing the most important features to feed into algorithms for machine learning. Feature selection methods get rid of unnecessary or useless features and only keep the ones that are most important to the machine learning model. This lowers the number of input factors. If you choose which traits are most important ahead of time instead of letting the machine learning model do it, here are the major benefits.

vi) Algorithms:

Long Short Term Memory (LSTM): LSTM is a recurrent neural network (RNN) that was created to fix the problem of disappearing gradients that happens in regular RNNs. The new memory cell lets the model understand long-term relationships in sequential data. This makes it perfect for jobs that involve time series or sequential patterns. [15] LSTMs are likely used in the project because they can model and understand how events and behaviors depend on time. This is very important for finding ransomware because the order of events and behaviors plays a big part. [45] Over time, LSTMs can pick up on subtle trends, which makes the model better at finding malicious actions.


LSTM

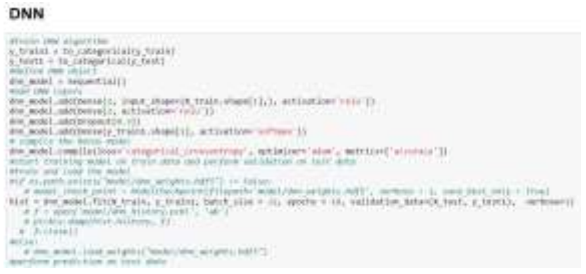
```
X_train = X_train.values
X_test = X_test.values

#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

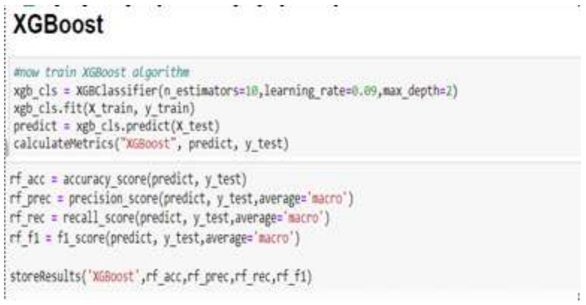
Deep Neural Network (DNN): A Deep Neural Network is a fancy name for an artificial neural network that has many buried layers between the input and output levels. Because these networks can learn complex hierarchical representations of data, they can be used for hard tasks that need to abstract and describe features. DNNs could be used in the project because they can learn complex traits and connections in the data that is collected. There may be subtle and complicated trends in ransomware detection, and DNNs can be a very useful tool for learning high-level models and extracting features [8, 13, 14].


DNN

```
#train DNN algorithm
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: Extreme Gradient Boosting, or XGBoost, is a machine learning method that is in the family of techniques called gradient boosting. It creates a group of weak learners, which are usually decision trees, one after the other. Each tree fixes the mistakes made by the trees that came before it, creating a strong and correct model. XGBoost is probably used because it is good at both classification jobs and working with big datasets. Regarding finding ransomware, XGBoost can be very good at making predictions, able to accurately capture the different ways that ransomware acts and help make a good detection model [8, 13, 14].


XGBoost

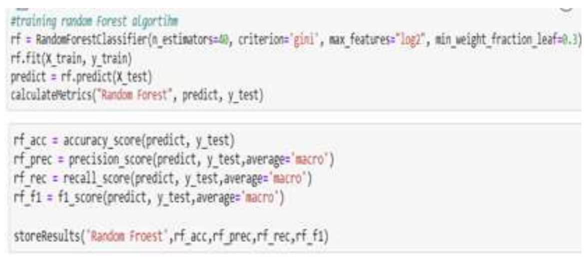
```
#now train XGBoost algorithm
xgb_cls = XGBClassifier(n_estimators=10,learning_rate=0.09,max_depth=2)
xgb_cls.fit(X_train, y_train)
predict = xgb_cls.predict(X_test)
calculateMetrics("XGBoost", predict, y_test)

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: Random Forest is an ensemble learning method that builds many decision trees while training. It gives you the mode of the classes (classification) or the average forecast (regression) of each tree. Use in the Project: Random Forest is used because it can handle difficult classification jobs. When looking for malware, where different trends may exist, a Random Forest ensemble can improve accuracy by combining the best features of several decision trees into a single, strong model.



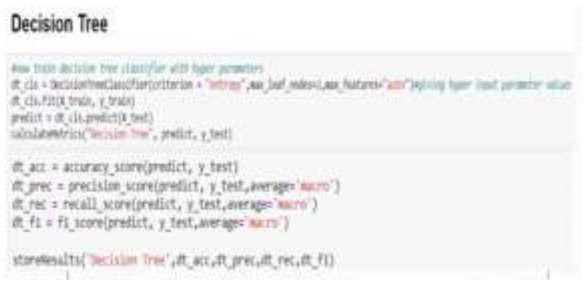
```
#training random Forest algortihm
rf = RandomForestClassifier(n_estimators=40, criterion='gini', max_features="log2", min_weight_fraction_leaf=0.3)
rf.fit(X_train, y_train)
predict = rf.predict(X_test)
calculateMetrics("Random Forest", predict, y_test)
```

```
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: A decision tree is a model that looks like a tree, and each point is a choice that is made based on the traits that are given. In a looping process, it divides the information into smaller groups, ending with nodes that represent the final guess or choice. Decision trees are used to show how decisions are made because they are easy to understand and use. When it comes to finding ransomware, Decision Trees can help you understand the steps that go into making a choice, which can help you figure out what factors affect the result of the discovery [52].



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)
```

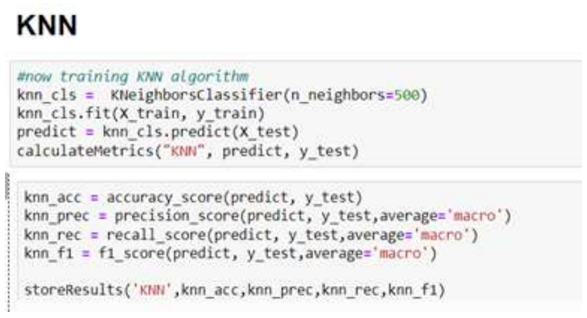
```
dt_acc = accuracy_score(predict, y_test)
dt_prec = precision_score(predict, y_test,average='macro')
dt_rec = recall_score(predict, y_test,average='macro')
dt_f1 = f1_score(predict, y_test,average='macro')

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

Fig 7 Decision tree

K – Nearest Neighbor (KNN):

KNN is a guided machine learning method that is used to decide what to classify and what to predict. It either guesses a new data point's value by taking the average of its k close neighbors' values in the feature space or puts it into a class based on the majority vote. KNN is probably used because it is easy to use and good at finding local trends in data. When looking for malware, where small differences may exist, KNN can be a creative way to find patterns in the information that are similar.



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)
```

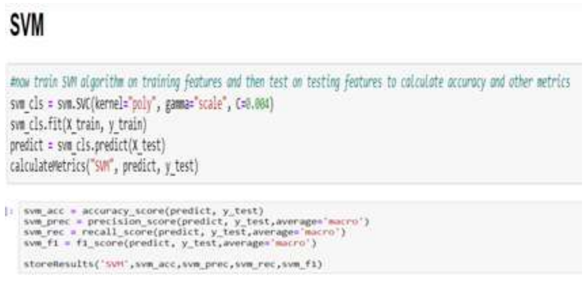
```
knn_acc = accuracy_score(predict, y_test)
knn_prec = precision_score(predict, y_test,average='macro')
knn_rec = recall_score(predict, y_test,average='macro')
knn_f1 = f1_score(predict, y_test,average='macro')

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

Fig 8 KNN

Support Vector Machine (SVM):

The SVM method is a type of guided machine learning that is used for jobs like regression and classification. It finds a hyperplane that best divides data into groups or guesses a continuous result, with the most space between groups. SVM is used because it can deal with data with many dimensions and find the best choice limits. In finding malware, where feature spaces can be complicated, SVM can be a reliable way to classify files by making clear decision lines [52].



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)
```

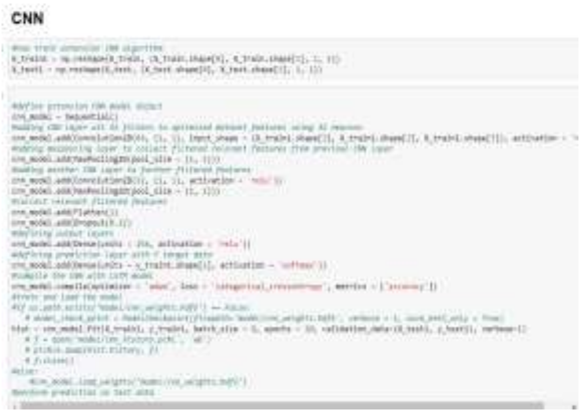
```
svm_acc = accuracy_score(predict, y_test)
svm_prec = precision_score(predict, y_test,average='macro')
svm_rec = recall_score(predict, y_test,average='macro')
svm_f1 = f1_score(predict, y_test,average='macro')

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

Fig 9 SVM

2D Convolutional Neural Network (CNN2D):

CNN2D is a deep learning method that works with grid-like data and is often used to analyze images. CNN2D is changed to work with continuous data in this project, though, by using convolutional layers to naturally learn features and patterns that are arranged in a hierarchy. CNN2D is used because it can easily pull out complicated features from continuous data. When looking for malware, where trends in the system's ongoing and sequential actions are very important, CNN2D can pick up on small details that help make the recognition model more accurate.



CNN

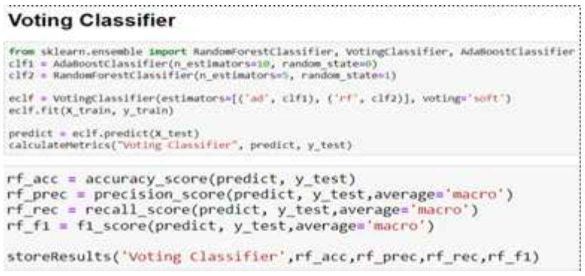
```
#now train extension CNN algorithm
X_train1 = np.reshape(X_train, (X_train.shape[0], X_train.shape[1], 1, 1))
X_test1 = np.reshape(X_test, (X_test.shape[0], X_test.shape[1], 1, 1))

#define extension CNN model object
cnn_model = Sequential()
#adding CNN layer with 32 filters to optimized dataset features using AI neurons
cnn_model.add(Convolution2D(32, (1, 1), input_shape = (X_train1.shape[1], X_train1.shape[2], X_train1.shape[3]), activation = '
#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 majority voting or average to mix results from several separate models. By using different methods, it is used to improve the general performance of the model. Predictions from different algorithms are put together by the Voting Classifier. This ensemble method takes into account different points of view from different models, which makes the general malware detection system more reliable and accurate.



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

4. EXPERIMENTAL RESULTS

Precision: Precision is the percentage of correctly classified events or samples that are among the hits. So, the following method can be used to figure out the accuracy:

Precision = True positives/ (True positives + False positives) = TP/ (TP + FP)

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

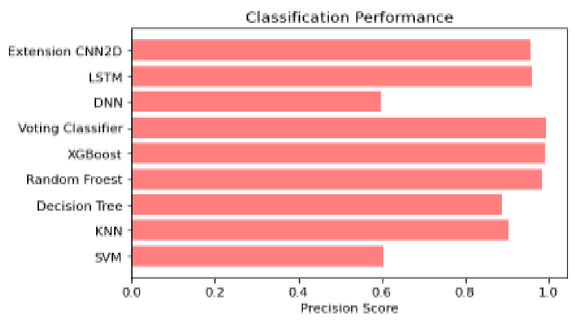


Fig 12 Precision comparison graph

Recall: Recall is a machine learning variable that measures how well a model can recognize all relevant examples of a certain class. It's the percentage of expected positive feelings that turn out to be real positive feelings.

This tells us how well a model can catch instances of a certain class.

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

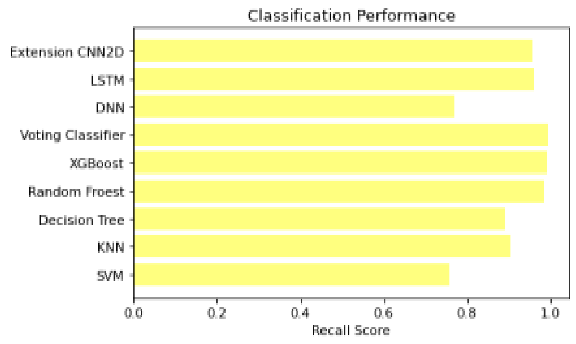


Fig 13 Recall comparison graph

Accuracy: Accuracy is the percentage of right guesses in a classification job. It shows how accurate a model's forecasts are generally.

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

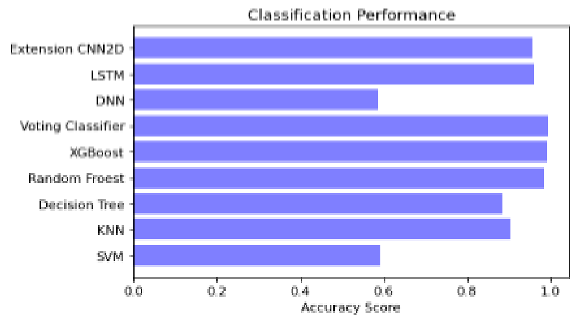


Fig 14 Accuracy graph

F1 Score: There is a machine learning rating tool called the F1 score that measures how accurate a model is. It adds up the accuracy and review scores of a model. The accuracy measurement figures out how often, across the whole collection, a model correctly predicted what would happen.

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

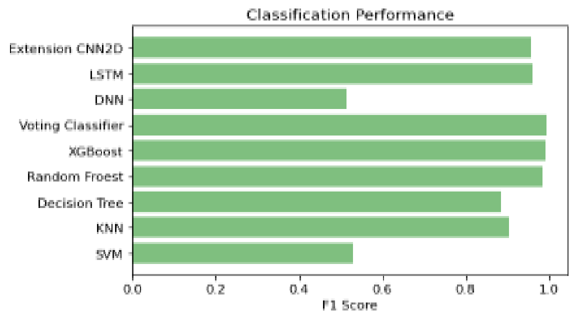
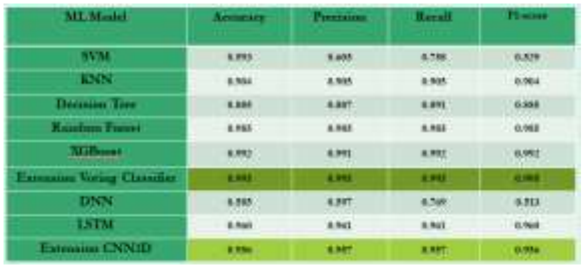


Fig 15 F1Score



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	0.886	0.887	0.891	0.888
Random Forest	0.985	0.982	0.982	0.982
XGBoost	0.992	0.991	0.992	0.992
Extension Voting Classifier	0.993	0.993	0.993	0.993
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

Fig 17 Home page

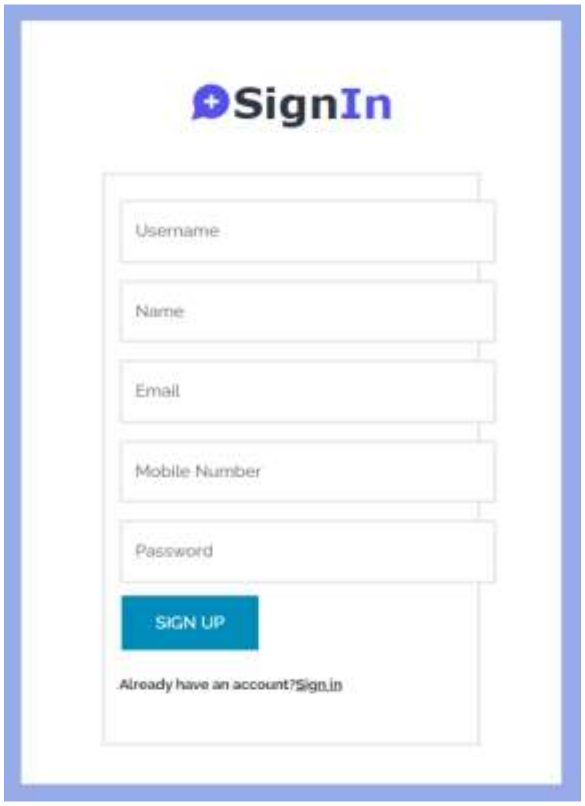


Fig 18 Signin page

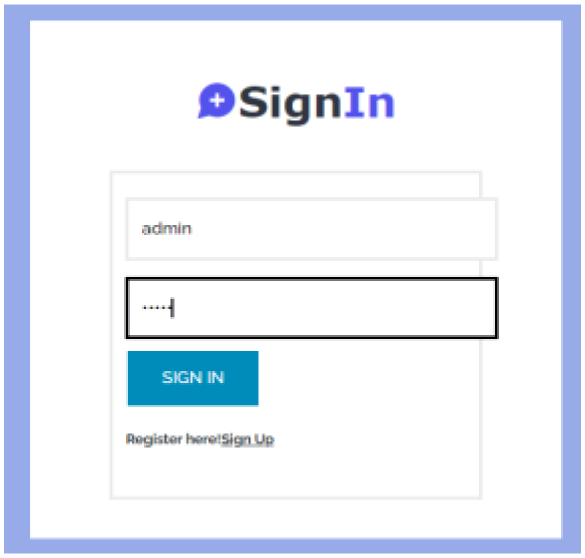


Fig 19 Login page

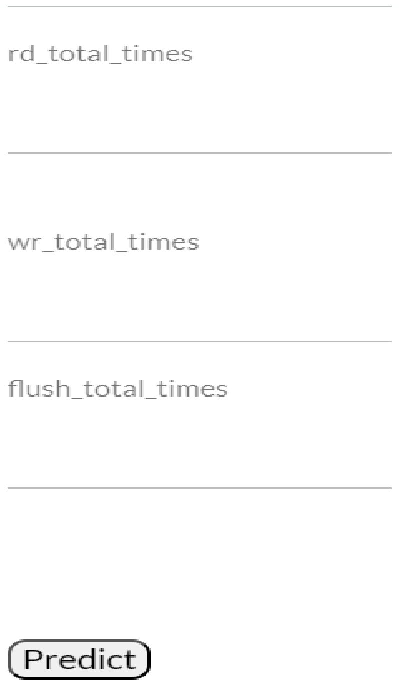


Fig 20 User input

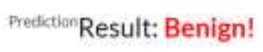


Fig 21 Predict result for given input

5. CONCLUSION

The project successfully provides a new way to find ransomware [9, 10, 11, 12, 13, 14]. It does this by using virtualization technology, hardware performance counters, and IO events data to improve accuracy while reducing the impact on system speed. The project does a lot of testing on different machine learning algorithms, such as SVM, KNN, Decision Tree, Random Forest, XGBOOST, DNN, and LSTM. It turns out that Random Forest and XGBOOST [52] are the best at predicting what ransomware will do. The project looks into how well deep learning models, especially DNN and LSTM, work. It gives useful information about how they compare to traditional machine learning methods, which adds to the variety of ways to make predictions. The project helps the defense community by releasing a dataset that comes from a number of different programs. This makes it easier for people to work together and gives experts a way to compare their ransomware detection models. The project uses Flask for the web framework and SQLite for user registration and identification. It has an easy-to-use interface where users can enter data, have it preprocessed, and get results from the learned model, which makes it more useful in real life.

6. FUTURE SCOPE

Even though this study only looked at a mix of known and unknown ransomware, more research could be done to see how well the proposed method works at finding new and different types of ransomware. Adding another part to the project to look at how different user workloads affect ransomware spotting [7] would add to the study's useful results by showing that it can handle different workloads. The voting algorithm, which is an add-on to the project, has done a great job of finding malware (99% of the time). Its reliability and ability to consistently find and stop ransomware threats is proven by rigorous testing on the front end using feature values. Putting the suggested method to the test in real-life situations would give useful information about how well it finds ransomware threats in live production settings. It is possible for the project to improve its ability to find viruses [24, 25, 34] by looking into adding more data sources or traits to the machine learning model. Working together with cybersecurity experts and groups gives us a chance to make sure our results are correct and to improve our suggested method using their knowledge and experience from the real world.

REFERENCES

[1] SR Department. (2022). Ransomware victimization rate 2022. Accessed: Apr. 6, 2022. [Online]. Available: https://www.statista. com/statistics/204457/businesses-ransomware-attack-rate/

[2] D. Braue. (2022). Ransomware Damage Costs. Accessed: Sep. 16, 2022. [Online]. Available: https://cybersecurityventures.com/globalransomware-damage-costs-predicted-to-reach-250-billion-usd-by-2031/

[3] Logix Consulting. (2020). What is Signature Based Malware Detection. Accessed: Apr. 3, 2023. [Online]. Available: https://www.logixconsulting. com/2020/12/15/what-is-signature-based-malware-detection/

[4] W. Liu, P. Ren, K. Liu, and H.-X. Duan, ''Behavior-based malware analysis and detection,'' in Proc. 1st Int. Workshop Complex. Data Mining, Sep. 2011, pp. 39–42.

[5] (2021). Polymorphic Malware. Accessed: Apr. 3, 2023. [Online]. Available: https://www.thesslstore.com/blog/polymorphic-malware-andmetamorphic-malware-what-you-need-to-know/

[6] M. Loman. (2021). Lockfile Ransomware's Box of Tricks: Intermittent Encryption and Evasion. Accessed: Nov. 16, 2021. [Online]. Available: https://news.sophos.com/en-us/2021/08/27/lockfile-ransomwares-box-oftricks-intermittent-encryption-and-evasion/

[7] N. Pundir, M. Tehranipoor, and F. Rahman, ''RanStop: A hardwareassisted runtime crypto-ransomware detection technique,'' 2020, arXiv:2011.12248.

[8] S. Mehnaz, A. Mudgerikar, and E. Bertino, ''RWGuard: A real-time detection system against cryptographic ransomware,'' in Proc. Int. Symp. Res.

Attacks, Intrusions, Defenses. Cham, Switzerland: Springer, 2018, pp. 114–136.

[9] J. Demme, M. Maycock, J. Schmitz, A. Tang, A. Waksman, S. Sethumadhavan, and S. Stolfo, ''On the feasibility of online malware detection with performance counters,'' ACM SIGARCH Comput. Archit. News, vol. 41, no. 3, pp. 559–570, Jun. 2013.

[10] A. Tang, S. Sethumadhavan, and S. J. Stolfo, ''Unsupervised anomalybased malware detection using hardware features,'' in Proc. Int. Workshop Recent Adv. Intrusion Detection. Cham, Switzerland: Springer, 2014, pp. 109–129.

[11] S. Das, J. Werner, M. Antonakakis, M. Polychronakis, and F. Monrose, ''SoK: The challenges, pitfalls, and perils of using hardware performance counters for security,'' in Proc. IEEE Symp. Secur. Privacy (SP), May 2019, pp. 20–38.

[12] S. P. Kadiyala, P. Jadhav, S.-K. Lam, and T. Srikanthan, ''Hardware performance counter-based fine-grained malware detection,'' ACM Trans. Embedded Comput. Syst., vol. 19, no. 5, pp. 1–17, Sep. 2020.

[13] B. Zhou, A. Gupta, R. Jahanshahi, M. Egele, and A. Joshi, ''Hardware performance counters can detect malware: Myth or fact?'' in Proc. Asia Conf. Comput. Commun. Secur., May 2018, pp. 457–468.

[14] S. Aurangzeb, R. N. B. Rais, M. Aleem, M. A. Islam, and M. A. Iqbal, ''On the classification of microsoft-windows ransomware using hardware profile,'' PeerJ Comput. Sci., vol. 7, p. e361, Feb. 2021.

[15] M. Alam, S. Bhattacharya, S. Dutta, S. Sinha, D. Mukhopadhyay, and A. Chattopadhyay, ''RATAFIA: Ransomware analysis using time and frequency informed autoencoders,'' in Proc. IEEE Int. Symp. Hardw. Oriented Secur. Trust (HOST), May 2019, pp. 218–227.

[16] K. Thummapudi, R. Boppana, and P. Lama, ''HPC 41 events 5 rounds,'' Harvard Dataverse, 2022, doi: 10.7910/DVN/MA5UPP.

[17] K. Thummapudi, R. Boppana, and P. Lama, ''IO 41 events 5 rounds,'' Harvard Dataverse, 2022, doi: 10.7910/DVN/GHJFUT.

[18] K. Thummapudi, R. Boppana, and P. Lama, ''HPC 5 events 7 rounds,'' Harvard Dataverse, 2022, doi: 10.7910/DVN/YAYW0J.

[19] K. Thummapudi, R. Boppana, and P. Lama, ''Io 5 events 7 rounds,'' Harvard Dataverse, 2022, doi: 10.7910/DVN/R9FYPL.

[20] K. Thummapudi, R. Boppana, and P. Lama, ''Scripts to reproduce results,'' Harvard Dataverse, 2023, doi: 10.7910/DVN/HSX6CS.

[21] M. Rhode, P. Burnap, and A. Wedgbury, ''Real-time malware process detection and automated process killing,'' Secur. Commun. Netw., vol. 2021, pp. 1–23, Dec. 2021.

[22] A. Kharraz and E. Kirda, ''Redemption: Real-time protection against ransomware at end-hosts,'' in Proc. Int. Symp. Res. Attacks, Intrusions, Defenses. Cham, Switzerland: Springer, 2017, pp. 98–119.

[23] F. Mbol, J.-M. Robert, and A. Sadighian, ''An efficient approach to detect torrentlocker ransomware in computer systems,'' in Proc. Int. Conf. Cryptol. Netw. Secur. Springer, 2016, pp. 532–541.

[24] K. Lee, S. Lee, and K. Yim, ''Machine learning based file entropy analysis for ransomware detection in

backup systems,'' IEEE Access, vol. 7, pp. 110205–110215, 2019.

[25] C. J. Chew and V. Kumar, ''Behaviour based ransomware detection,'' in Proc. Int. Conf. Comput. Their Appl., in EPiC Series in Computing, vol. 58. 2019, pp. 127–136.

[26] S. Homayoun, A. Dehghantanha, M. Ahmadzadeh, S. Hashemi, and R. Khayami, ''Know abnormal, find evil: Frequent pattern mining for ransomware threat hunting and intelligence,'' IEEE Trans. Emerg. Topics Comput., vol. 8, no. 2, pp. 341–351, Apr. 2020.

[27] A. Kharaz, S. Arshad, C. Mulliner, W. Robertson, and E. Kirda, ''UNVEIL: A large-scale, automated approach to detecting ransomware (keynote),'' in Proc. IEEE 24th Int. Conf. Softw. Anal., Evol. Reengineering (SANER), Feb. 2017, pp. 757–772.

[28] A. Kharraz, W. Robertson, D. Balzarotti, L. Bilge, and E. Kirda, ''Cutting the gordian knot: A look under the hood of ransomware attacks,'' in Proc. Int. Conf. Detection Intrusions Malware, Vulnerability Assessment. Cham, Switzerland: Springer, 2015, pp. 3–24.

[29] A. Continella, A. Guagnelli, G. Zingaro, G. De Pasquale, A. Barenghi, S. Zanero, and F. Maggi, ''ShieldFS: A self-healing, ransomware-aware filesystem,'' in Proc. 32nd Annu. Conf. Comput. Secur. Appl., Dec. 2016, pp. 336–347.

[30] M. Shukla, S. Mondal, and S. Lodha, ''POSTER: Locally virtualized environment for mitigating ransomware threat,'' in Proc. ACM SIGSAC Conf. Comput. Commun. Secur., Oct. 2016, pp. 1784–1786.

[31] N. Scaife, H. Carter, P. Traynor, and K. R. B. Butler, ''CryptoLock (and drop it): Stopping ransomware attacks on user data,'' in Proc. IEEE 36th Int. Conf. Distrib. Comput. Syst. (ICDCS), Jun. 2016, pp. 303–312.

[32] D. Sgandurra, L. Muñoz-González, R. Mohsen, and E. C. Lupu, ''Automated dynamic analysis of ransomware: Benefits, limitations and use for detection,'' 2016, arXiv:1609.03020.

[33] P. Zavarsky and D. Lindskog, ''Experimental analysis of ransomware on windows and Android platforms: Evolution and characterization,'' Proc. Comput. Sci., vol. 94, pp. 465–472, Jan. 2016.

[34] T. McIntosh, J. Jang-Jaccard, P. Watters, and T. Susnjak, ''The inadequacy of entropy-based ransomware detection,'' in Proc. Int. Conf. Neural Inf. Process. Cham, Switzerland: Springer, 2019, pp. 181–189.

[35] Z. A. Genc, G. Lenzini, and D. Sgandurra, ''On deception-based protection against cryptographic ransomware,'' in Proc. Int. Conf. Detection Intrusions Malware, Vulnerability Assessment Cham, Switzerland: Springer, 2019, pp. 219–239.

[36] S. Song, B. Kim, and S. Lee, ''The effective ransomware prevention technique using process monitoring on Android platform,'' Mobile Inf. Syst., vol. 2016, pp. 1–9, Mar. 2016.

[37] K. Cabaj, M. Gregorczyk, and W. Mazurczyk, ''Software-defined networking-based crypto ransomware detection using HTTP traffic characteristics,'' Comput. Electr. Eng., vol. 66, pp. 353–368, Feb. 2018.

[38] R. Moussaileb, B. Bouget, A. Palisse, H. Le Bouder, N. Cuppens, and J.-L. Lanet, ''Ransomware's early mitigation mechanisms,'' in Proc. 13th Int. Conf. Availability, Rel. Secur., 2018, pp. 1–10.

[39] Z. A. Genc, G. Lenzini, and P. Y. Ryan, ''No random, no ransom: A key to stop cryptographic ransomware,'' in Proc. Int. Conf. Detection Intrusions Malware, Vulnerability Assessment. Cham, Switzerland: Springer, 2018, pp. 234–255.

[40] M. M. Ahmadian, H. R. Shahriari, and S. M. Ghaffarian, ''Connectionmonitor & connection-breaker: A novel approach for prevention and detection of high survivable ransomwares,'' in Proc. 12th Int. Iranian Soc. Cryptol. Conf. Inf. Secur. Cryptol. (ISCISC), Sep. 2015, pp. 79–84.

[41] E. Kolodenker, W. Koch, G. Stringhini, and M. Egele, ''PayBreak: Defense against cryptographic ransomware,'' in Proc. ACM Asia Conf. Comput. Commun. Secur., Apr. 2017, pp. 599–611.

[42] M. S. Kiraz, Z. A. Genc, and E. Ozturk, ''Detecting large integer arithmetic for defense against crypto ransomware,'' Cryptology ePrint Arch., Rep., vol. 558, p. 2017, Jan. 2017.

[43] (2022). What Systems Have You Seen Infected by Ransomware? Accessed: Apr. 3, 2023. [Online]. Available: https://www.statista.com/ statistics/701020/major-operating-systems-targeted-by-ransomware/

[44] (2022). Linux Profiling With Performance Counters. Accessed: Apr. 3, 2023. [Online]. Available: https://perf.wiki.kernel.org/ index.php/MainPage

[45] (2022). Likwid Performance Tools. Accessed: Apr. 3, 2023. [Online]. Available: https://hpc.fau.de/research/tools/likwid/

[46] K. Keahey, J. Anderson, Z. Zhen, P. Riteau, P. Ruth, D. Stanzione, M. Cevik, J. Colleran, H. S. Gunawi, C. Hammock, J. Mambretti, A. Barnes, F. Halbach, A. Rocha, and J. Stubbs, ''Lessons learned from the chameleon testbed,'' in Proc. USENIX Annu. Tech. Conf., Jul. 2020, pp. 219–233.

[47] Alexa Top Websites. Accessed: Sep. 13, 2021. [Online]. Available: https://www.alexa.com/topsites

[48] Ninite. Accessed: Apr. 3, 2023. [Online]. Available: https://ninite.com

[49] API. (2023). Libvirt: Virsh Tool Manual. Accessed: Apr. 3, 2023. [Online]. Available: https://libvirt.org/manpages/virsh.html

[50] (2021). Virusshare. Accessed: Nov. 16, 2021. [Online]. Available: https://virusshare.com

[51] Intel. (2023). System Programming Guide. Accessed: Apr. 3, 2023. [Online]. Available: https://www.intel.com/content/dam/www/ public/us/en/documents/manuals/64-ia-32-architectures-softwaredeveloper-vol-3b-part-2-manual.pdf

[52] F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, J. Vanderplas, A. Passos, D. Cournapeau, M. Brucher, M. Perrot, and E. Duchesnay, ''Scikit-learn: Machine learning in Python,'' J. Mach. Learn. Res., vol. 12, pp. 2825–2830, Oct. 2011.

[53] J. Benesty, J. Chen, Y. Huang, and I. Cohen, ''Pearson correlation coefficient,'' in Noise Reduction in Speech Processing. Berlin, Germany: Springer, 2009, pp. 1–4.

[54] H. Jin, Q. Song, and X. Hu, ''Auto-Keras: An efficient neural architecture search system,'' in Proc. 25th ACM SIGKDD Int. Conf. Knowl. Discovery Data Mining, Jul. 2019, pp. 1946–1956.

[55] Wikipedia. (2021). Sensitivity and Specificity. Accessed: Apr. 3, 2023. [Online]. Available: https://en.wikipedia.org/wiki/ Sensitivityandspecificity

[56] Hat Enterprise. (2023). Block I/O Tuning. [Online]. Available: https://access.redhat.com/documentation/en-us/redhatenterpriselinux/ 7/html/virtualizationtuningandoptimizationguide/sectvirtualizationtuningoptimizationguide-blockio-techniques