International Journal of Computer Trends and Technology
ISSN: 2231 – 2803 / https://doi.org/10.14445/22312803/IJCTT-V69I4P106

Volume 69 Issue 4, 23-33, April, 2021


*Original Article*

# Variational Autoencoder based Data Augmentation & Corroboration

Atharva Bankar[1], Chanavi Singh[2], Lakshanya Shinde[3], Pallavi Udatewar[4]

[1,2,3,4] *Student at School of Computer Engineering and Technology, MIT World Peace University, Pune, India*

Received Date: 05 March 2021
Revised Date: 15 April 2021
Accepted Date: 17 April 2021

***Abstract*** *- Cybersecurity attacks spanning countries and organizations are triggered by networks that are compromised with cryptographic ransomware, which results in the loss of millions of dollars in the form of extortion amount. By encrypting the user files, this type of malicious software takes them hostage and demands a large ransom payment in exchange for the decryption key. In most cases, cryptocurrency is used as a method of payment.*

*The combination of efficient and well-implemented cryptographic methods to take the data hostage, the Tor protocol for anonymous correspondence, and the use of a cryptocurrency to collect unmediated payments give ransomware attackers a high degree of impunity. Every year, a number of ransomware attacks on various institutions compel them to keep a huge chunk of money aside to pay the ransom in order to access their files quickly. This calls for a need to address this issue.*

*In this paper, we propose the use of Autoencoders (AE) and Variational Autoencoders (VAE) to augment the data consisting of ransomware properties with two techniques: AE and VAE on the entire test set and on each ransomware independently. This verifies the robustness of Machine Learning models- Extra Tree Classifier, XGBoost Classifier, and Random Forest Classifier. The metrics used to judge the classification of the ransomware verifies if the data generated is in accordance with the dataset used.*

***Keywords*** *– Cryptocurrency, Bitcoin, Ransomware, Machine Learning, Autoencoder, Variational Autoencoders.*

## I. INTRODUCTION

Interest in blockchain technology, specifically Bitcoin, is on the rise since January 2018. Bitcoin is a cryptocurrency that came into use in 2009. It uses blockchain technology, much like the other cryptocurrencies that followed it. Some estimate that if they were to integrate Bitcoin into Apple Pay, Apple could produce $100 billion in shareholder value [1].

It is possible to generate Bitcoin transactions anonymously, and it does not require any kind of identity verification. Furthermore, anonymous networks are used to request payment from a sender using a public Bitcoin address. Because of the availability and ease of usage provided by bitcoin transactions, Bitcoin has been a target for many illicit users.

Ransomware is malicious software that holds your data captive in demand for money. One of the most frequent forms of cyber-attacks this year has been ransomware attacks. In a recent webinar, Greg Foss claimed that the world would see more ransomware by 2021, which will be purposely re-factored and converted into purely destructive attacks [2]. Attacks on large databases have taken place where everything is wiped clean and replaced with fake data. Because of the increasing quantity and severity of ransomware attacks, now more than ever, there is a pressing need for better protection measures against such threats. Our approach suggests using industry-standard machine learning algorithms to tackle this and check their resilience by testing them against augmented data.

Properties of Bitcoin addresses have been used to train machine learning models. Income, neighbors, weight, length, count, and loop are the properties of the ransomware. Income is the total amount of coins produced to the address. Neighbors of an address are the number of transactions having the same output address. The weight of an address is the fractional sum of coins reaching the address. The total number of non-starter transactions on its longest chain is called the length of an address. A chain of an address is known as an acyclic path that originates from any starter transaction ending with the address. The count of an address is the number of starter transactions that are linked through the chain to the address. The loop of an address is the number of starter transactions connected to the address having more than one directed path [3].

## II. LITERATURE SURVEY

Bitcoin is a peer-to-peer based system of online payment, which relies on trust rather than a mediating financial institution. The original paper does mention the possible pitfalls in case of an attacker trying to forge and create coins out of thin air or the attacker trying to steal back their payment [5]. However, what they couldn't foresee was the misuse of anonymity to demand a ransom via bitcoins in return for the victim's highly valuable data. Bitcoin being a public ledger makes the payment traceable, but the parties between which the transaction takes place can still be subject to anonymity and obscurity.

As Paquet-Clouston et al. state [6], the combination of efficient and well-implemented cryptographic methods to take hostage data, the Tor protocol to Anonymous correspondence, and the use of a cryptocurrency to accumulate unmediated payments give ransomware attackers a high degree of freedom. Montreal [6], Princeton [7], and Padua [8] studies have studied cryptocurrency ransomware networks in ransomware research and found that hacker activity can help us identify undisclosed ransomware payments. There are publicly available databases of all three studies. The authors of this paper have used these three datasets to study and understand the dependence of various features on their addresses and to help track and predict malicious transactions.

Bitcoin is a currency based on Proof-of-Work that enables users to create digital coins by computing. Proof of work is one CPU-one vote. The higher the computation time associated with a certain transaction, the more are the chances of it being an honest transaction. Through digitally signing their transactions, Bitcoin users execute transfers and are prevented from double-spending their coins via a distributed time-stamping program. This service functions on top of the Bitcoin peer-to-peer network and guarantees that all transactions and their execution order are accessible to the public. In general, each user has hundreds of different Bitcoin addresses, all of which are stored by their client and handled transparently [9].

There have been studies to prevent ransomware attacks at their early stages through their delivery channels using supervised machine learning algorithms [10], as well as preventing ransomware attacks using a layered defense system called RansomWall [11]. Gonzalez-Hayajneh, discuss the various crypto-ransomware types, their typical behavior, and recommend methods of prevention. They explain the various sources of malware and their type. Further, they elaborate on the general tactics used by the ransomware to infiltrate and the strategies used. They conclude by describing various methods to help prevent and safeguard oneself from such attacks [12]. One ransomware detection study done on CryptoLocker [13] used Kolmogorov-Smirnov tests to gain insight into CryptoLocker's targets and the variations in their targets by statistically determining various distributions throughout the day when different ransom types were being paid.

Chen et al. evaluate the resilience of ML algorithms in areas where security is a concern. To do so, they make use of the generative adversarial network (GAN) to generate generalized malicious samples and explore reasons why such samples cause the output of a certain class of ransomware classifiers to degrade. They aim to make the use of ML algorithms more robust for security-related concerns [14]. Another paper by Mahmudha Fasheem et al. elaborates on the ransomware attacks. They propose automatic test packet generation (ATPG) to achieve the same goals as the aforementioned paper [15].

A method of detecting ransomware using autoencoders has already been proposed [17]. They used a two-step framework consisting of LSTM-based autoencoders to detect ransomware. With the aid of high-performance computing, they could perform anomaly detection with almost zero false positives in a matter of 5 seconds. However, data augmentation has not been performed in this paper to verify the resilience of their architecture.

Akcora et al., in their work of data analysis for ransomware detection [3], identify Bitcoin addresses that are used to store and trade Bitcoins gained through ransomware activities. Kingma-Welling introduces and implements a stochastic variational inference and learning algorithm that scales to large datasets and even works under some mild conditions of differentiability [16].

We use various algorithms including Random Forest Classifier [18], XGBoost Classifier [19], and Extra Trees Classifier [20] to discover the trends observed in the aforementioned properties of a malicious address, namely Income, Neighbours, Weight, Length, Count and Loop and to discover and improve the possibility of identifying an attack. In this paper, we will be using Autoencoders and Variational Autoencoders to check the robustness of Machine Learning models.

## III. DATASET

This paper uses the dataset from the UCI repository [4]. Features for our dataset are ransomware address, year, day, length, weight, count, looped, neighbors, income, and label. The label is the category of the ransomware family, i.e., DMALocker, WannaCry, CryptXXX, etc. The label "white" indicates that it is not ransomware. Fig. I, II, and III show the data distribution of ransomware, and Fig. IV showcases the correlation matrix for the properties of ransomware payment.

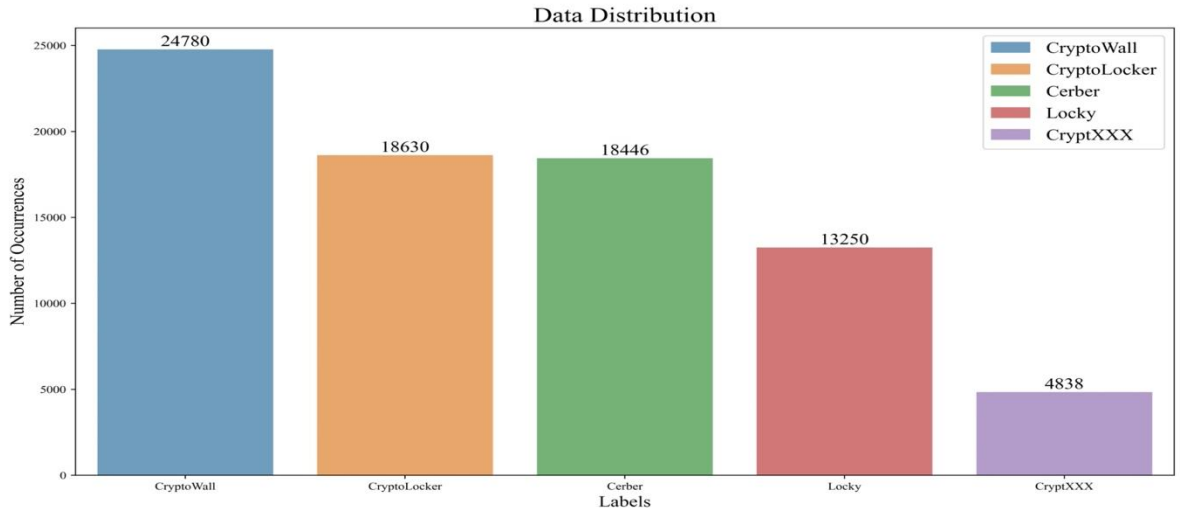


**Fig. 1 Data Distribution - 1**

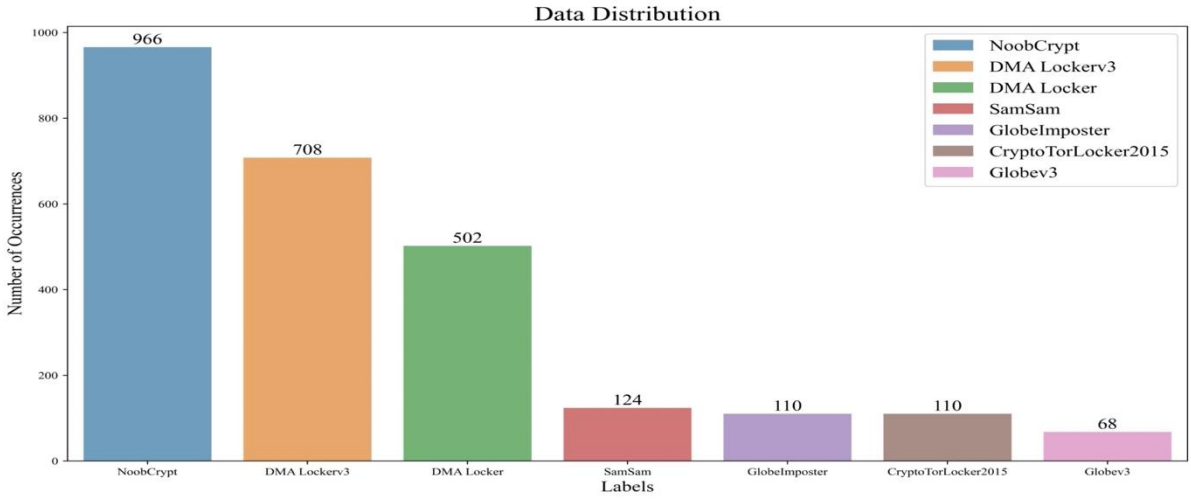


**Fig. 2 Data Distribution - 2**

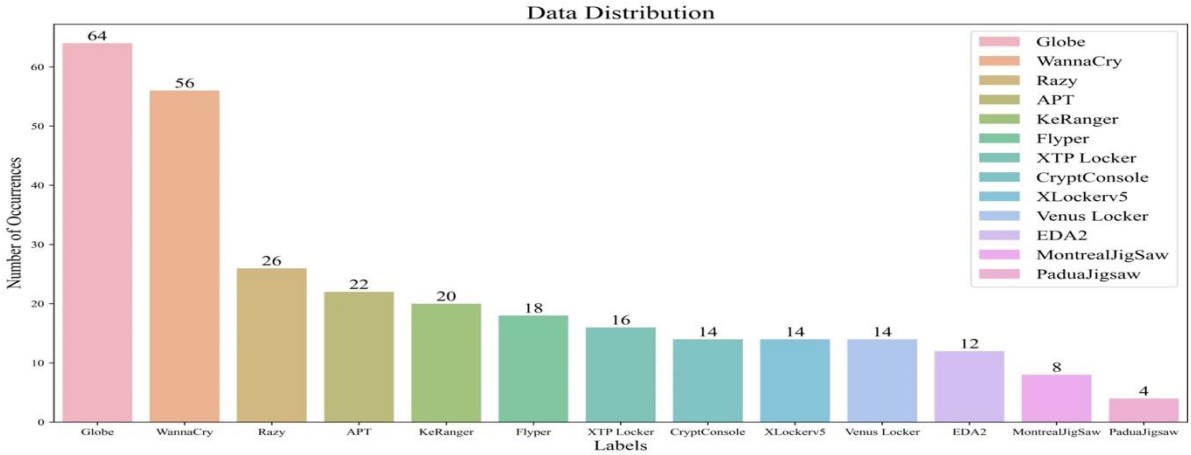


**Fig. 3 Data Distribution - 3**

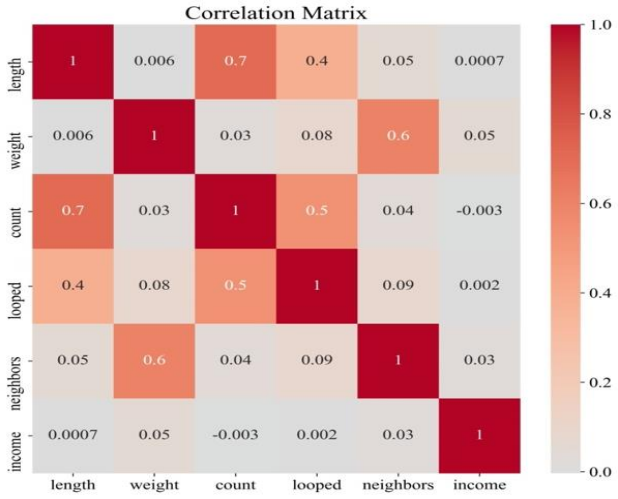


**Fig. 4 Correlation Matrix for properties of ransomware payment**

## IV. METHODOLOGY

### A. *Data Reduction*

For non-ransomware target label white, all repetitive instances of data have been removed.

### B. *Data Preprocessing*

In order to train the Machine Learning algorithms, the data is required to be scaled to maintain the stability of the model against colossal values in the data. Tree-based algorithms like Decision Trees and Random Forest remain justly insensitive to scaling. However, it becomes less cumbersome to visualize the data and interpret the model for comparing it with distance-based algorithms like Support Vector Machines and K-Nearest Neighbours. This study uses the following data pre-processing techniques:

#### *a) Log Transformation*

Data transformation technique where each variable var is replaced with log(var). Then the continuous data does not obey the bell-shaped curve, and log transformation makes this data normal provided that they obey a log-normal distribution. This diminishes the data skewness, which results in improving the linearity between our target and observed variables. For instances with var=0, log(var+0.001) has been calculated to avoid undefined instances of data.

#### *b) Data Normalization*

It is pre-processing of data such that its range is between 0 and 1. It is a robust way of dealing with outliers as compared to normalizing the data with respect to its standard variance. This technique makes it easier to verify whether the data augmented with autoencoders is in accordance with the dataset.

### C. *Algorithms*

#### *a) Random Forest*

**Procedure**: $RandomForest(D, E, p)$
**Data**: Training set $D$, size of the ensemble $E$, the number of instances per sub-sample $p$
**Result**: Tree ensemble
**Begin**

$Trees \leftarrow \emptyset$
**For** $i \in \{1, \ldots, E\}$ **do**
$D_i \leftarrow \emptyset$;
**while** $|Q_{Di}| < p$ **do**
$q \leftarrow selectRandom\,(Q_D \setminus Q_{Di})$;
$D_i.add(\,(x_{q,j}, l_{q,j})_{j=1}^{n_q})$;
**end**
$Trees.add(ConstructTrees(D_i$ ));
**end**
return $Trees$;
**end**

An object is uniformly selected from the set A by the function $selectRandom(A)$.

**Working:**

**Step 1** − First, begin by choosing random samples from a given dataset.

**Step 2** − Next, for every sample, this algorithm will construct a decision tree. Then, from every decision tree, it will get the prediction result.

**Step 3** − In this step, for every predicted outcome, voting will be carried out.

**Step 4** −Finally, for the final prediction result, select the most voted prediction outcome.

#### *b) XGBoost*

**Data**: Dataset and hyperparameters
Define $f_0(x)$;
**for** $k = 1,2, \ldots, M$ do
Evaluate $g_k = \frac{\partial L(y,f)}{\partial f}$;
Evaluate $h_k = \frac{\partial^2 \partial L(y,f)}{\partial f}$;
By selecting splits with maximized gain, determine the structure
A = $\frac{1}{2}\left[\frac{G_L^2}{H_L} + \frac{G_R^2}{H_R} + \frac{G^2}{H}\right]$;
Find out the left weights $w^* = \frac{-G}{H}$;
Find out the base learner $\hat{b}(x) = \Sigma_{j=1}^{T} wI$
Add trees $f_k(x) = f_{k-1}(x) + \hat{b}(x)$;
**end**
**Result**: $f(x) = \Sigma_{k=0}^{M} f_k(x)$

**Working**

**Step 1** – XGBoost applies the machine learning algorithm under the gradient Boosting model.

**Step 2 -** The Gradient Boosting algorithm's initial thought is to *estimate the mean value of the target* y.
**Step 3 –** Calculation pseudo-residuals.
**Step 4 -** Prediction of the pseudo-residuals.
**Step 5 –** Fit the model on the loss that is generated from the preceding step.
**Step 6 –** Making a prediction and measuring the residuals.

***c) Extra Trees***

**Data**: Observation $O$
**Result**: Resemblance matrix S
$D \leftarrow addCl3(O)$;
$T \leftarrow$
$Build\_an\_extra\_tree\_emsemble(D, k)$

S = $0_{n_{obs}, n_{obs}}$
**for** $d_i \in D$ do
  **for** $d_j \in D$ do
    $S_{i,j}$ = the no. of times sample $d_i$ and $d_j$ fall into the same leaf node in each of the trees;
  **end**
**end**
$S_{i,j} = \frac{s_{i,j}}{M}$;

**Working**
**Step 1 -** The algorithm operates by generating a variety of unpruned decision trees from the training dataset.
**Step 2 -** By majority voting, the predictions are grouped to make the final prediction.
**Step 3 –** Similar to Random Forest, at each split point of a decision tree, the algorithm will randomly sample the features.

**Step 4 –** The algorithm chooses an arbitrary split point not similar to the random forest, which uses a greedy algorithm to pick an optimal split point.

***D. Hyper-Parameter Optimization***

For the Machine Learning model to generalize well and give accurate results on unseen data, hyper-parameter optimization is a crucial step. An exhaustive search is performed for the best set of manually tuned hyper-parameter values such that the model outputs the best accuracy. This has been achieved by dividing the dataset into K folds in a stratified way such that each iteration of model training is performed on different instances of data but with the same proportion of target variables in the train and test dataset.

***E Autoencoder***

An autoencoder is a special form of neural network which tries to match its input to its output in an unsupervised manner. It is possible to interpret the network as consisting of three components: A function $h = f(x)$ encoder is used for compressing the input data into an encoded representation, a reconstruction function $\mathrm{r} = \mathrm{g(h)}$ decoder used to reconstruct the output from the encoded representation to be identical to the input provided, and a loss function used for comparing the output with the target. The autoencoder also has a hidden layer internally called as code layer or compression layer, which describes the encoded representation of the input which is fed into the decoder.

**INPUT**: A collection of samples of pieces of training $\{X, Y\}_{i=1}^{N}$, instances of testing, trade-off parameters $\alpha, \beta, \gamma,$, the number of k hidden layer nodes, and the number of labels for the class.
Initialize: the weight matrices W = $\{W\}_{c=1}^{2(C+1)}$ and the bias vectors $B = \{B_c\}_{c=1}^{2(c+1}$

**For** each epoch, **do**
**For** samples $\{X_i, Y_i\}$ **do**
  **For** j =1, 2…, k
  $X_i \leftarrow [X_i]$;
    **While** the algorithm coverage ( $L < 10^{-6}$ or the no. of the iterations is less than 300);
      Evaluate the partial derivatives of all variables;
      Update the weight matrices iteratively $W = \{W_c\}_{c=1}^{2(C+1)}$ and $B = \{B_c\}_{c=1}^{2(c+1}$;
      (Adam update of $W = \{W_c\}_{c=1}^{2(C+1)}$and $B = \{B_c\}_{c=1}^{2(c+1)}$);
      Evaluate the hidden layer and output layer;
      Determine the value $\bar{y}_{i,j}$ in coordination with the output of the autoencoder classifier;
**OUTPUT**: Predict the values of $\bar{Y}_i$

***F. Variational Autoencoder***

In order to thwart overfitting, a variational autoencoder can be exemplified as an autoencoder whose training is standardized in order to warrant that the latent space has good properties that allow generative processes. The difference between variational autoencoders and standard autoencoders is that in variational autoencoders, we encode it as distribution through the latent space instead of encoding an input as a single point. First, the input is encoded as the latent space distribution. From that distribution, a sample point from the latent space is obtained. This point is reconstructed, and the error of reconstruction can be calculated. The error in reconstruction is backpropagated through the network.

$i \leftarrow 0$
**While** not converged, **do**
  Sample $\{x^{(1)}, \ldots., x^{(m)}\}$ from the data distribution. $p_D(x)$

Sample $\{z^{(1)}, \ldots, z^{(m)}\}$ from prior $p(z)$

Sample $\{\epsilon^{(1)}, \ldots, \epsilon^{(m)}\}$ from $N(0,1)$

Evaluate $\theta$-gradient:

$$g_\theta \leftarrow \frac{1}{m}\Sigma_{k=1}^{m} \nabla_\theta \log p_\theta(x^{(k)} | z_\phi(x^{(k)}, \epsilon^{(k)}))$$

Evaluate $\emptyset$ —gradient (eq. 3.7):

$$g_\phi \leftarrow \frac{1}{m}\Sigma_{k=1}^{m}[-T_\psi\left(x^{(k)} \middle| z_\phi(x^{(k)}, \epsilon^{(k)})\right) + \log p_\theta(x^{(k)} | z_\phi(x^{(k)}, \epsilon^{(k)}))$$

Evaluate $\psi$- gradient (eq.3.3) :

$$g_\Psi \leftarrow \frac{1}{m}\Sigma_{k=1}^{m}\nabla_\theta[\log(\sigma(T_\psi\left(x^{(k)} \middle| z_\phi(x^{(k)}, \epsilon^{(k)})\right) + \log(1 - \sigma(T_\psi (x^{(k)}, \epsilon^{(k)}))]$$

Perform Adam-updates for $\theta, \emptyset \text{ and } \psi$:

$$\theta \leftarrow \theta + h_i g_\theta,\ \emptyset \leftarrow \emptyset + h_i g_\emptyset, \Psi \leftarrow \Psi + h_i g_\Psi$$

$$i \leftarrow i + 1$$

**End while**

Fig. 5 shows the general autoencoder architecture comprising of the input layer, compression layer, and output layer.

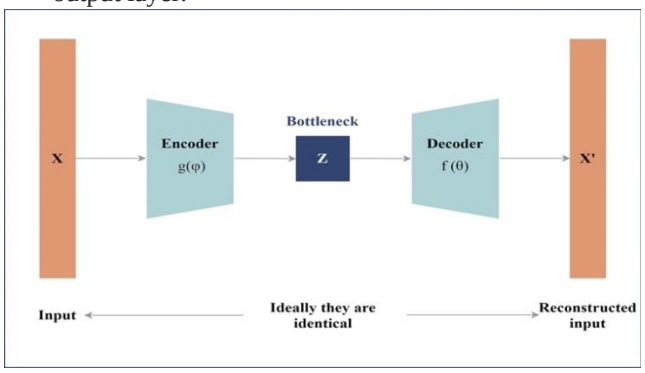


**Fig. 5 Autoencoder Architecture**

## V. RESULTS

### A. Metrics

***a) Accuracy***

Accuracy is the set of data points accurately predi cted from all the data points. More precisely, it is calculated by the set of true positives and true negatives divided by the set of true positives, true negatives, false positives, and false negatives.

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

***b) Precision***

Precision is defined as the ratio of a set of true positives and the sum of the set of true positives and false positives.

$$Precision = \frac{TP}{TP + FP}$$

***c) Recall***

The recall is defined as the ratio of a set of true positives and the sum of the set of true positives and false negatives.

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

***d) F-1 Score***

The F1 score can be described as precision and recall harmonic average, where an F1 score achieves its best value at 1 and its worst score at 0.

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

***e) Mean Absolute Error (MAE)***

The mean absolute error of a model in relation to a test set is the mean of the absolute values of the individual errors of estimation over all the test set instances. For instance, each prediction error is the difference between the true value and the expected value.

$$mae = \frac{\Sigma_{i=1}^{n} abs(y_i - \lambda(x_i))}{n}$$

### B. Feature Importance

To provide useful insights into the data, the tree-based models designate a score to all the independent variables in a relative manner depending on how effective they are at predicting a dependent variable. In addition to this, it shows how various models give importance to the independent variables. Dimensionality reduction and feature selection can be prompted with feature importance by discarding the less important attributes in the dataset and thereby improving the performance of the machine learning model.

Feature importance is measured as a decrease in node impurity weighted by the chance of touching the node. The probability of the node is obtained after dividing the number of occurrences that reach the node by the total number of occurrences. Higher is the relative score of the attribute, and more the target variable can be described with the help of that attribute. Fig. 6, 7, and 8 show the feature importance for Random Forest, XGBoost, and Extra Trees algorithm, respectively. Fig. 9 shows the weighted featured importance of all three aforementioned algorithms.

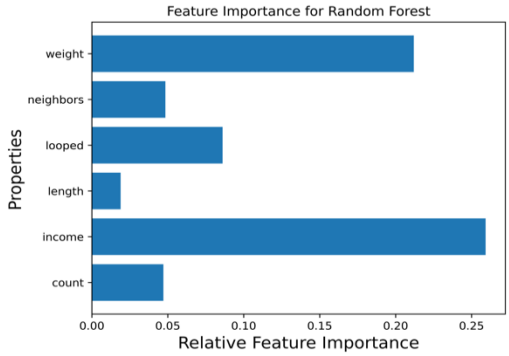


**Fig. 6 Relative Feature Importance of properties of ransomware – Random Forest**

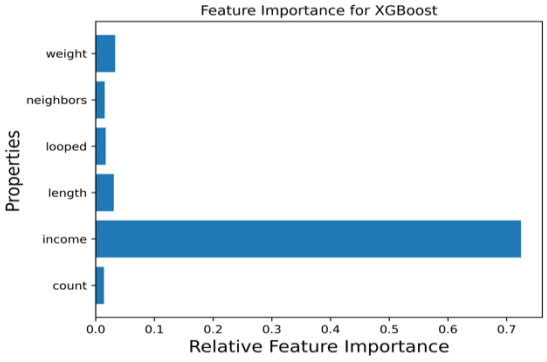


**Fig. 7 Relative Feature Importance of properties of ransomware – XGBoost**

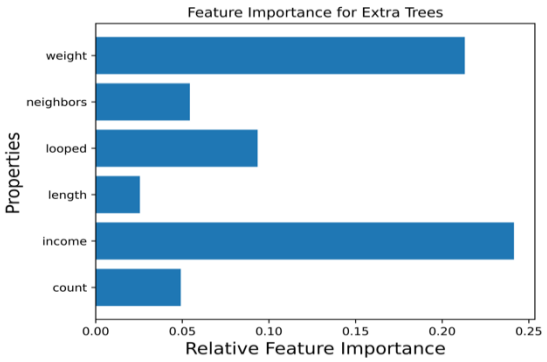


**Fig. 8 Relative Feature Importance of properties of ransomware – Extra Trees**

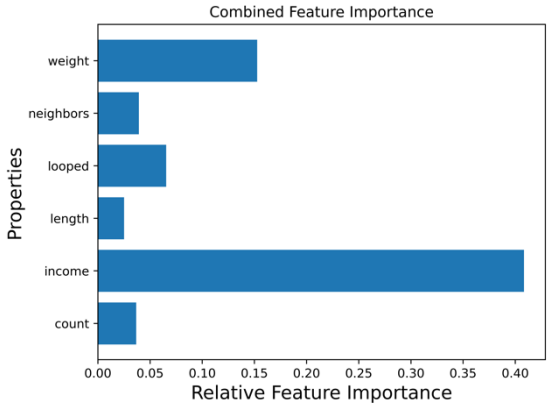


**Fig. 9 Combined Feature Importance**

### *B. Performance of Machine Learning Models*

In order to make the predictions stronger, hyperparameter tuning was used. As shown in Table I, XGBoost outputs the highest accuracy with the number of trees equal to 1850, maximum depth of the tree to 9, and learning rate to 0.1. The learning rate determines the pace at which the model adapts to the problem while making sure that the model doesn't overfit the training set. In doing so, one performs a trade-off between computation time and the best possible outcome.

On similar lines, Extra Tree uses maximum features equal to $log_2(n)$ where n is the number of properties of ransomware, and the minimum number of samples needed to slice an internal node is 3. Random forest uses the number of trees equal to 250. A train-test split of 80-20 was created in a stratified fashion to achieve the results mentioned in table I.

**Table 1. Metrics (Precision, Recall & F-1 Score are weighted)**

| Algorithm | Accuracy | Precision | Recall | F-1 Score | MAE | Computation Time (seconds) |
|---|---|---|---|---|---|---|
| Random Forest | 0.9511 | 0.9529 | 0.9511 | 0.9516 | 0.8486 | 91.37 |
| XGBoost | 0.9523 | 0.9538 | 0.9523 | 0.9527 | 0.5864 | 568.42 |
| Extra Trees | 0.9501 | 0.9514 | 0.9501 | 0.9504 | 0.8563 | 23.12 |

### *C. Classification Report*

The dataset used in this paper is heavily imbalanced, as shown in Fig. 1, Fig. 2, and Fig. 3. which makes it imperative to check how the Machine Learning model performs on each ransomware. The quality of predictions of an algorithm is computed using a classification report.

Table II shows the precision, recall, and F-1 score of all three models for each ransomware. It can be deduced that the classification for ransomware having a smaller number of instances is accurate in spite of the imbalanced dataset.

**Table 2. Classification Report**

| Target Variable | Precision | | | Recall | | | F-1 Score | | |
|---|---|---|---|---|---|---|---|---|---|
| | RF | XGB | ET | RF | XGB | ET | RF | XGB | ET |
| APT | 1.0000 | 0.9422 | 1.0000 | 0.5000 | 0.9838 | 0.5000 | 0.6666 | 0.9625 | 0.6666 |
| Cerber | 0.8996 | 1.0000 | 0.8969 | 0.9449 | 1.0000 | 0.9414 | 0.9217 | 1.0000 | 0.9186 |
| CryptConsole | 1.0000 | 1.0000 | 1.0000 | 1.0000 | 1.0000 | 1.0000 | 1.0000 | 1.0000 | 1.0000 |
| CryptXXX | 0.9683 | 0.8382 | 0.9570 | 0.9783 | 0.8717 | 0.9679 | 0.9732 | 0.8546 | 0.9625 |
| CryptoLocker | 0.8422 | 0.9599 | 0.8568 | 0.8872 | 0.9897 | 0.8789 | 0.8642 | 0.9746 | 0.8677 |
| CryptoTorLocker | 1.0000 | 1.0000 | 1.0000 | 0.8181 | 1.0000 | 0.8181 | 0.9000 | 1.0000 | 0.9000 |
| CryptoWall | 0.8606 | 0.9000 | 0.8650 | 0.9572 | 0.8182 | 0.9471 | 0.9063 | 0.8571 | 0.9042 |
| DMA Locker | 0.9220 | 0.9167 | 0.9220 | 0.7100 | 0.7857 | 0.7100 | 0.8022 | 0.8462 | 0.8022 |
| DMA Locker v3 | 0.8870 | 0.8571 | 0.9000 | 0.7746 | 0.8028 | 0.7605 | 0.8270 | 0.8291 | 0.8244 |
| EDA2 | 1.0000 | 0.8114 | 1.0000 | 1.0000 | 0.7358 | 1.0000 | 1.0000 | 0.7717 | 1.0000 |
| Flyper | 1.0000 | 0.9000 | 1.0000 | 0.5000 | 0.6923 | 0.5000 | 0.6666 | 0.7826 | 0.6666 |
| Globe | 1.0000 | 1.0000 | 1.0000 | 0.8461 | 0.5000 | 0.8461 | 0.9166 | 0.6667 | 0.9166 |
| GlobeImposter | 1.0000 | 0.9140 | 0.9473 | 0.8181 | 0.9536 | 0.8181 | 0.9000 | 0.9334 | 0.8780 |
| Globev3 | 1.0000 | 1.0000 | 1.0000 | 0.7857 | 1.0000 | 0.7857 | 0.8800 | 1.0000 | 0.8800 |
| KeRanger | 1.0000 | 0.8000 | 0.6666 | 0.5000 | 0.8000 | 0.5000 | 0.6666 | 0.8000 | 0.5714 |
| Locky | 0.9271 | 0.7500 | 0.9184 | 0.9694 | 1.0000 | 0.9603 | 0.9477 | 0.8571 | 0.9389 |
| Montreal Jigsaw | 1.0000 | 1.0000 | 1.0000 | 1.0000 | 0.3333 | 1.0000 | 1.0000 | 0.5000 | 1.0000 |
| NoobCrypt | 0.8644 | 0.8421 | 0.8388 | 0.7927 | 0.7273 | 0.7823 | 0.8270 | 0.7805 | 0.8096 |
| Padua – Jigsaw | 1.0000 | 1.0000 | 1.0000 | 1.0000 | 1.0000 | 1.0000 | 1.0000 | 1.0000 | 1.0000 |
| Razy | 1.0000 | 1.0000 | 1.0000 | 0.6000 | 1.0000 | 0.6000 | 0.7500 | 1.0000 | 0.7500 |
| SamSam | 0.8846 | 0.8000 | 0.8846 | 0.9200 | 1.0000 | 0.9200 | 0.9019 | 0.8889 | 0.9019 |
| Venus Locker | 1.0000 | 0.8750 | 1.0000 | 1.0000 | 0.8400 | 1.0000 | 1.0000 | 0.8571 | 1.0000 |
| WannaCry | 1.0000 | 0.9000 | 1.0000 | 0.6363 | 0.8100 | 0.6363 | 0.7777 | 0.8526 | 0.7777 |
| White | 0.9775 | 0.9781 | 0.9749 | 0.9567 | 0.9583 | 0.9581 | 0.9670 | 0.9681 | 0.9664 |
| XLockerv5 | 1.0000 | 0.8592 | 1.0000 | 1.0000 | 0.9520 | 1.0000 | 1.0000 | 0.9032 | 1.0000 |
| XTP Locker | 1.0000 | 1.0000 | 1.0000 | 1.0000 | 0.6667 | 1.0000 | 1.0000 | 0.8000 | 1.0000 |

### *D. Classification Analysis*

As mentioned in Table I, XGBoost gives the best accuracy of 95.23%. Fig.10. shows the confusion matrix, which is normalized over the predicted labels for better interpretation of classification analysis.

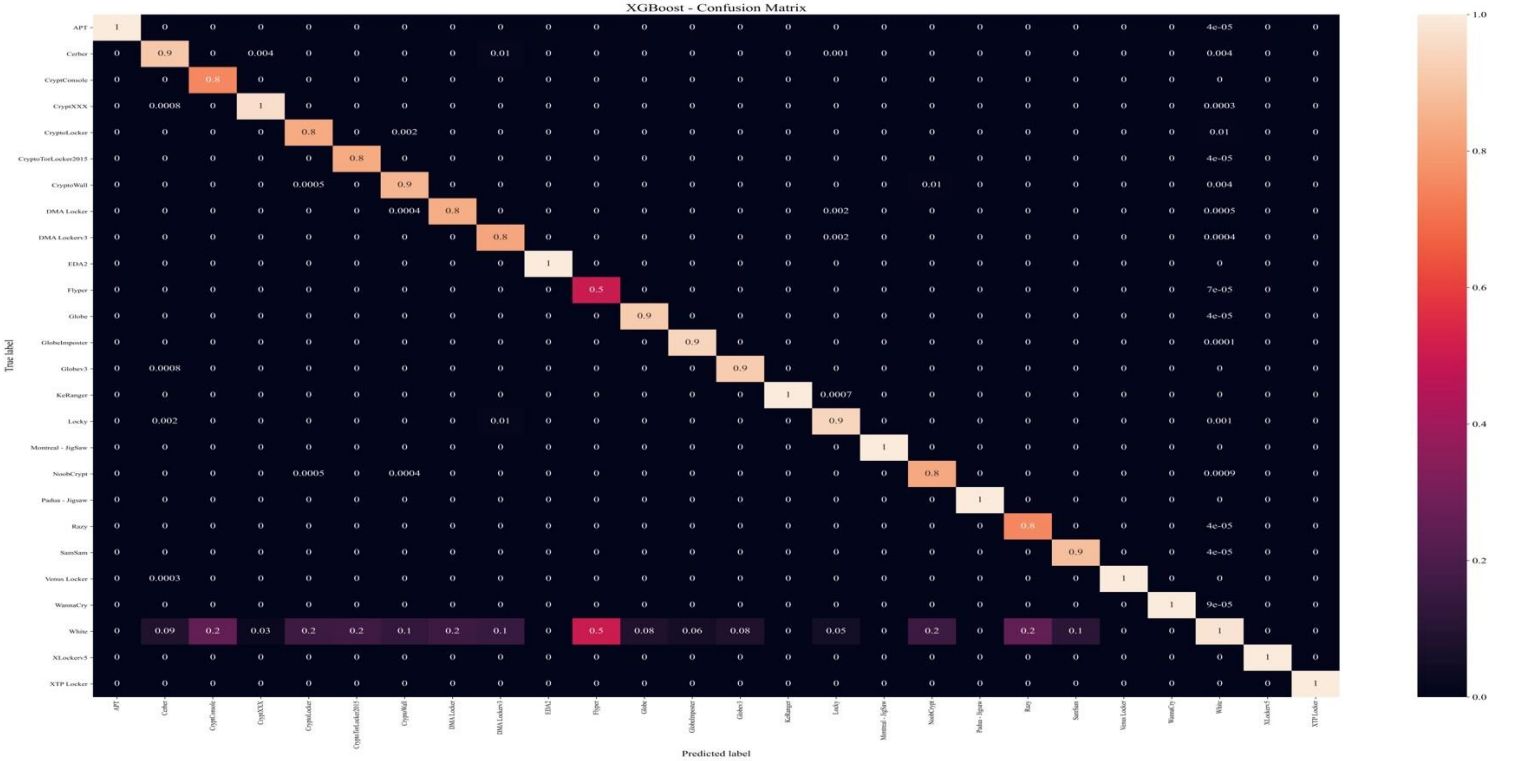


**Fig. 10 Confusion Matrix - XGBoost**

### *E. Training of Autoencoder (AE) & Variational Autoencoder (VAE)*

The paper uses two techniques - AE & VAE in order to generate new data for assessing the numeric data that is generated on the test set. The architecture for AE and VAE has a compression layer of 4 units which is retracted back to properties of ransomware in the output layer of the decoder neural network. This, in turn, checks the robustness of the Machine Learning models. These two techniques have been applied to the test set in order to augment the data in two ways each:

#### *a) Entire Test set:*

AE & VAE has been applied on the entire test set such that it contains all the instances of various ransomware, i.e., one AE & VAE architecture where each architecture has all ransomware.

#### *b) With respect to Ransomware:*

AE & VAE has been applied on all ransomware independently present in the test set, i.e., a total of twenty-six AE & VAE architectures where each architecture has only one specific ransomware. This augmented data is then combined.

Fig. 11. shows the latent space comprising 4 units of the neural network, which is the compression layer.

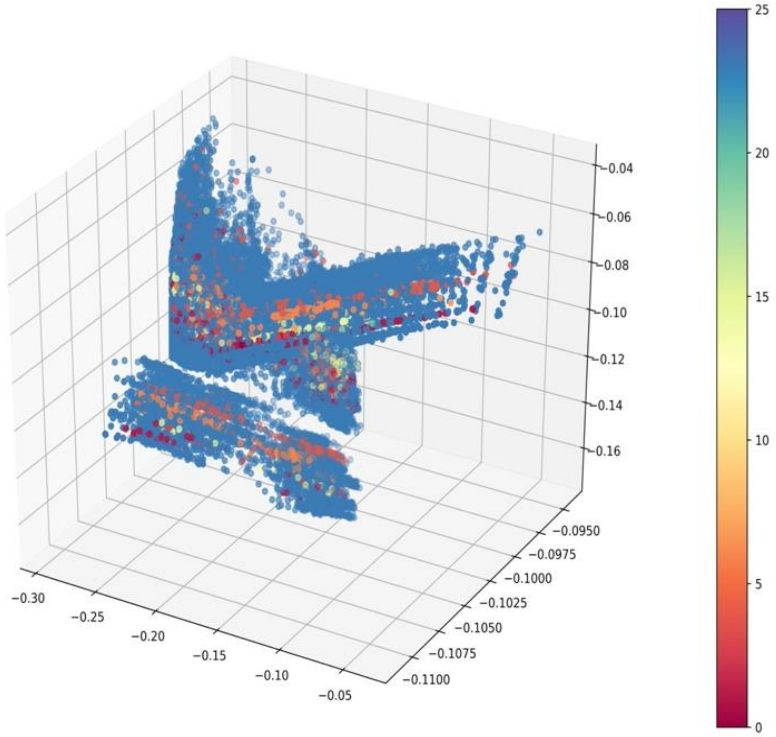


**Fig. 11 Latent Space for VAE (Colour bar represents the ransomware index from 0 to 25)**

**Table 3. Entire test set (Acc: accuracy, Prec: precision, Rec: recall, MAE: mean absolute error)**

| ML Algorithm | AE | | | | | VAE | | | | |
|---|---|---|---|---|---|---|---|---|---|---|
| | Acc. | Prec. | Rec. | F-1 Score | MAE | Acc. | Prec. | Rec. | F-1 Score | MAE |
| Random Forest | 0.8446 | 0.8437 | 0.8446 | 0.8308 | 2.6555 | 0.8340 | 0.8314 | 0.8340 | 0.8194 | 2.8041 |
| XGBoost | 0.8356 | 0.8373 | 0.8356 | 0.8239 | 2.7925 | 0.8180 | 0.8256 | 0.8180 | 0.8092 | 2.9665 |
| Extra Trees | 0.8454 | 0.8489 | 0.8454 | 0.8290 | 2.6398 | 0.8397 | 0.8418 | 0.8397 | 0.8228 | 2.7156 |

**Table 4. With respect to ransomware (Acc: accuracy, Prec: precision, Rec: recall, MAE: mean absolute error)**

| ML Algorithm | AE | | | | | VAE | | | | |
|---|---|---|---|---|---|---|---|---|---|---|
| | Acc. | Prec. | Rec. | F-1 Score | MAE | Acc. | Prec. | Rec. | F-1 Score | MAE |
| Random Forest | 0.8889 | 0.8872 | 0.8889 | 0.8808 | 1.8277 | 0.8894 | 0.8870 | 0.8894 | 0.8820 | 1.8313 |
| XGBoost | 0.8439 | 0.8466 | 0.8439 | 0.8346 | 2.6223 | 0.8595 | 0.8592 | 0.8595 | 0.8492 | 2.3602 |
| Extra Trees | 0.8752 | 0.8759 | 0.8752 | 0.8645 | 2.0865 | 0.8666 | 0.8678 | 0.8666 | 0.8547 | 2.2337 |

Table III represents the performance of data that is augmented using the entire test set on the models, and

Table IV represents the performance of data that is augmented with respect to ransomware on the models. From table III and IV, it can be concluded that the Random Forest algorithm gives the best accuracy of 88.94% for the data that is trained with VAE with respect to ransomware. Table V gives a detailed classification report for the best result from table III and IV.

**Table 5. Classification report of Random Forest for the data augmented using VAE with respect to ransomware**

| Target Variable | Precision | Recall | F-1 Score |
|---|---|---|---|
| APT | 1.0000 | 0.2222 | 0.3636 |
| Cerber | 0.8496 | 0.7106 | 0.7739 |
| CryptConsole | 1.0000 | 0.5000 | 0.6666 |
| CryptXXX | 0.9033 | 0.5263 | 0.6651 |
| CryptoLocker | 0.8112 | 0.4993 | 0.6181 |
| CryptoTorLocker | 1.0000 | 0.4545 | 0.6250 |
| CryptoWall | 0.8689 | 0.8468 | 0.8577 |
| DMA Locker | 0.8488 | 0.3631 | 0.5087 |
| DMA Locker v3 | 0.8888 | 0.3943 | 0.5463 |
| EDA2 | 1.0000 | 0.4000 | 0.5714 |
| Flyper | 1.0000 | 0.2500 | 0.4000 |
| Globe | 1.0000 | 0.4230 | 0.5945 |
| GlobeImposter | 1.0000 | 0.4090 | 0.5806 |
| Globev3 | 1.0000 | 0.3928 | 0.5641 |
| KeRanger | 1.0000 | 0.2500 | 0.4000 |
| Locky | 0.9115 | 0.6316 | 0.7462 |
| Montreal – Jigsaw | 1.0000 | 0.3333 | 0.5000 |
| NoobCrypt | 0.7794 | 0.4108 | 0.5380 |
| Padua – Jigsaw | 1.0000 | 0.5000 | 0.6666 |
| Razy | 1.0000 | 0.2727 | 0.4285 |
| SamSam | 0.9200 | 0.4893 | 0.6388 |
| Venus Locker | 1.0000 | 0.5000 | 0.6666 |
| WannaCry | 1.0000 | 0.3043 | 0.4666 |
| White | 0.8965 | 0.9677 | 0.9308 |
| XLockerv5 | 1.0000 | 0.5000 | 0.6666 |
| XTP Locker | 1.0000 | 0.4285 | 0.6000 |

## VI. CONCLUSION

Many who use the internet have had their personal data thieved or compromised. The criminals are rarely punished because most cybercriminals operate outside the jurisdictions of the Law Enforcing Agencies. Although large enterprises can certainly pay the ransom, small businesses cannot afford to shell out huge amounts of their earnings frequently.

This study uses tree-based Machine Learning algorithms- Random Forest, XGBoost, and Extra Trees to identify the ransomware. With XGBoost attaining the best accuracy of 95.23%, the performance of the aforementioned algorithms is similar in terms of the metrics used. In order to check the reliability of the three algorithms, this paper generates new data using Autoencoder and Variational Autoencoder. It can be deduced that training the AE & VAE with respect to the target variable results in more accurate data augmentation as compared to training the AE & VAE on the entire test set at once. Furthermore, VAE being generative in nature, is able to generate new examples of the data from the latent space that can test the Machine Learning models' ability to work on unprecedented data. An accuracy of 88.94% is attained with Random Forest on the newly generated data using VAE. The techniques used in this paper can be utilized with appropriate hyper-parameter tuning across multiple domains where Machine Learning is applied in order to check the efficiency of the models on unseen data.

From the results, it can be established that the recall obtained for ransomware with a smaller number of instances is lower as compared to others. The future scope can include using different architectures of AE or VAE to generate more plausible and realistic instances of imbalanced data and thereby increasing the recall.

## REFERENCES


[1] IPWatchdog., The Blockchain Patent Landscape Shows Accelerating Growth. 10(2020) Retrieved from. https://www.ipwatchdog.com/2020/12/04/the-blockchain-patent-landscape-shows-accelerating-growth/id=127922/

[2] Infosecurity., Ransomware set for Evolution in Attack Capabilities in (2021). Retrieved from https://www.infosecurity-magazine.com/news/ransomware-evolution-capabilities/

[3] Cuneyt G. Akcora, Yitao Li, Yulia R. Gel, Murat Kantarcioglu, BitcoinHeist: Topological Data Analysis for Ransomware Prediction on the Bitcoin BlockchaiN., Proceedings of the Twenty-Ninth International Joint Conference on Artificial Intelligence Special Track on AI in FinTech. Pages (2020) 4439-4445. https://doi.org/10.24963/ijcai.2020/612

[4] [dataset] Akcora, C., Li, Y., Gel, Y., & Kantarcioglu, M., BitcoinHeist: Topological Data Analysis for Ransomware Detection on the Bitcoin Blockchain. arXiv preprint arXiv:1906.07852 (2019).

[5] S. Nakamoto. Bitcoin: A peer-to-peer electronic cash system (2008) https://bitcoin.org/bitcoin.pdf.

[6] Masarah Paquet-Clouston, Bernhard Haslhofer, Benoît Dupont, Ransomware payments in the Bitcoin ecosystem, Journal of Cybersecurity, 5(1)(2019) tyz003, https://doi.org/10.1093/cybsec/tyz003

[7] Huang, D. Y., Aliapoulios, M. M., Li, V. G., Invernizzi, L., Bursztein, E., McRoberts, K., Levin, J., Levchenko, K., Snoeren, A. C., & McCoy, D., Tracking Ransomware End-to-end. In Proceedings - 2018 IEEE Symposium on Security and Privacy, SP (2018) 618-631. [8418627] (Proceedings - IEEE Symposium on Security and Privacy; Institute of Electrical and Electronics Engineers Inc. (2018). https://doi.org/10.1109/SP.2018.00047

[8] Conti, Mauro & Gangwal, Ankit & Ruj, Sushmita. On the Economic Significance of Ransomware Campaigns: A Bitcoin Transactions Perspective. Computers & Security. 79(2018) 162-189. https://doi.org/10.1016/j.cose.2018.08.008

[9] Androulaki E., Karame G.O., Roeschlin M., Scherer T., Capkun S., Evaluating User Privacy in Bitcoin. In: Sadeghi AR. (eds) Financial Cryptography and Data Security. FC. Lecture Notes in Computer Science,7859Springer, Berlin, Heidelberg. (2013). https://doi.org/10.1007/978-3-642-39884-1_4

[10] Gangwar, Keertika & Mohanty, Subhranshu & Mohapatra, A., Analysis, and Detection of Ransomware Through Its Delivery Methods. (2018).https://doi.org/10.1007/978-981-10-8527-7_29

[11] S. K. Shaukat and V. J. Ribeiro, RansomWall: A layered defense system against cryptographic ransomware attacks using machine learning.,10th International Conference on Communication Systems & Networks (COMSNETS), Bengaluru, (2018) 356-363, https://doi.org/10.1109/COMSNETS.2018.8328219

[12] D. Gonzalez and T. Hayajneh., Detection and prevention of crypto-ransomware., IEEE 8th Annual Ubiquitous Computing, Electronics and Mobile Communication Conference (UEMCON), New York, NY, (2017) 472-478, https://doi.org/10.1109/UEMCON.2017.8249052

[13] K. Liao, Z. Zhao, A. Doupe and G. Ahn, Behind closed doors: measurement and analysis of CryptoLocker ransoms in Bitcoin, APWG Symposium on Electronic Crime Research (eCrime), Toronto, ON, (2016) 1-13. https://doi.org/10.1109/ECRIME.2016.7487938

[14] Li Chen, Chih-Yuan Yang, Anindya Paul, Ravi Sahita.Towards resilient machine learning for ransomware detection., In KDD Workshop. ACM, New York, NY, USA, 10 pages. arXiv preprint arXiv:1812.09400 ., (2019)

[15] S.Mahmudha Fasheem, P.Kanimozhi, B.Akora Murthy. Detection and Avoidance of Ransomware IJEDR 5(1)(2017) ISSN: 2321-9939, https://www.ijedr.org/papers/IJEDR1701090.pdf

[16] Diederik P Kingma, Max Welling Auto-Encoding Variational Bayes., arXiv preprint arXiv:1312.6114v10., (2014).

[17] M. Alam, S. Bhattacharya, S. Dutta, S. Sinha, D. Mukhopadhyay, and A. Chattopadhyay., RATAFIA: Ransomware Analysis using Time And Frequency Informed Autoencoders., IEEE International Symposium on Hardware Oriented Security and Trust (HOST), McLean, VA, USA, (2019)218-227. https://doi.org/10.1109/HST.2019.8740837

[18] Breiman, L. Random Forests. Machine Learning 45(2001) 5–32. https://doi.org/10.1023/A:1010933404324

[19] Tianqi Chen and Carlos Guestrin. XGBoost: A Scalable Tree Boosting System. In Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining (KDD '16). Association for Computing Machinery, New York, NY, USA, (2016)785–794. https://doi.org/10.1145/2939672.2939785

[20] Geurts, P., Ernst, D. & Wehenkel, L., Extremely randomized trees. Mach Learn 63(2006) 3–4. https://doi.org/10.1007/s10994-006-6226-1