Deep Dive into Ransomware Threat Hunting and Intelligence at Fog Layer

Sajad Homayoun[1], Ali Dehghantanha[2], Marzieh Ahmadzadeh[1], Sattar Hashemi[3], Raouf Khayami[1], Kim-Kwang Raymond Choo[4], David Ellis Newton[5]

Abstract

Ransomware, a malware designed to encrypt data for ransom payments, is a threat to fog layer nodes as such nodes typically contain considerably amount of sensitive data. The capability to efficiently hunt abnormalities relating to ransomware activities is crucial in timely detection of ransomware. In this paper, we present our *Deep Ransomware Threat Hunting and Intelligence System (DRTHIS)* to distinguish ransomware from goodware and identify their families. Specifically, *DRTHIS* utilizes *Long Short-Term Memory (LSTM)* and *Convolutional Neural Network (CNN)*, two deep learning techniques, for classification using the softmax algorithm. We then use 220 *Locky*, 220 *Cerber* and 220 *TeslaCrypt* ransomware samples, and 219 goodware samples, to train *DRTHIS*. Findings from our evaluations demonstrate that the proposed system achieves an F-measure of 99.6% with a true positive rate of 97.2% in the classification of ransomware instances. Additionally, we demonstrate that *DRTHIS* is capable of detecting previously unseen ransomware samples from new ransomware families in a timely and accurate manner using ransomware from the *CryptoWall*, *TorrentLocker* and *Sage* families. The findings show that 99% of *CryptoWall* samples, 75% of *TorrentLocker* samples and 92% of *Sage* samples are correctly classified.

Keywords: Crypto-ransomware, ransomware detection, ransomware family detection, deep learning, Long Short-Term Memory, Convolutional Neural Network.

1. Introduction

Ransomware is a recent threat that has affected a number of industries and countries [1], and is reportedly the fastest growing malware type [2, 3]. Today's ransomware is a sophisticated threat affecting users all around the world. The first wave of misleading applications began to appear in 2005 where performance enhancement tools or fake spyware removal tools such as RegistryCare, PerformanceOptimizer and SpySherriff mainly affected Windows computers. These tools claimed that there is a critical performance/security issue in victim's computer and suggested to buy an extra program for removing the problem. Afterwards, a more disruptive form of extortion emerged that disabled access and control of the computer by locking up the computer from use. Because of effective recovery solutions such as re-installing the Operating System or un-installing fake softwares, cyber criminals shifted to cryptography based ransomware. In the literature there are two main types of ransomwares namely *Locker* and *Crypto* ransomwares. Lockers denies users' access without making any changes to the data stored on the system while the crypto-ransomware encrypts all or selected data based on a predefined file formats such as *.pdf, *.doc etc. usually using a strong cryptography algorithm such as AES or RSA[4]. After encryption, the victim is presented with the ransom payment instructions with possibility of recovering ransomed data. Unsurprisingly, the ransomware topic has also attracted the attention of security researchers and practitioners. For example, *ransomwaretracker.abuse.ch*[6] tracks major ransomware families, such as *Locky, Cerber, TeslaCrypt, CryptoWall, TorrentLocker* and *Sage*. *Locky* ransomware is usually distributed via phishing e-mails that contain Microsoft Word Office documents with embedded malicious macros, which will subsequently result in the download of the ransomware [5]. *Cerber* ransomware is often distributed via exploit kits [6], and has the capability to encrypt the victims data without connecting to a command and control (C2) server. *TeslaCrypt* is another major family of ransomware distributed using exploit kits and capable of encrypting all user contents including

Email addresses: S.Homayoun@sutech.ac.ir (Sajad Homayoun), A.Dehghantanha@sheffield.ac.uk (Ali Dehghantanha), Ahmadzadeh@sutech.ac.ir (Marzieh Ahmadzadeh), S_Hashemi@shirazu.ac.ir (Sattar Hashemi), Khayami@sutech.ac.ir (Raouf Khayami), Raymond.Choo@fulbrightmail.org (Kim-Kwang Raymond Choo), D.E.Newton@salford.ac.uk (David Ellis Newton)

[1]Department of Computer Engineering and Information Technology, Shiraz University of Technology, Shiraz, Iran.

[2]Department of Computer Science, University of Sheffield, Sheffield, U.K.

[3]Department of Computer Engineering, Shiraz University, Shiraz, Iran.

[4]Department of Information Systems and Cyber Security and Department of Electrical and Computer Engineering, The University of Texas at San Antonio, San Antonio, TX 78249, USA.

[5]Department of Computer Science, School of Computing, Science and Engineering, University of Salford, Salford, U.K.

[6]https://ransomwaretracker.abuse.ch/tracker/

network mapped drives [7]. *CryptoWall* ransomware first appeared in early 2014s [7], and is widely distributed using web exploit kits, phishing emails, and corrupted attachments (e.g. PDF files). *TorrentLocker* ransomware is distributed via emails that deceive victims into downloading the ransomware by sending emails purporting to be shipping notifications, driving violations, or other corporate/government correspondence. *Sage* is a more recent ransomware, which is distributed via Microsoft Office documents and is capable of encrypting user data without the need to contact a C2 server.

With the popularity of *Internet of Things (IoT)* devices and these devices being placed at the edge/fog layer [8, 9, 10], such devices can also be targeted by ransomware attackers [11, 12]. The naive solution of encrypting the fog layer of an IoT network (also known as data historian nodes) would impact on data collection from IoT devices (e.g. sensors being deployed in the field) as well as affecting the quality of the architecture due to the exacting computational requirements.

While user training may minimize the success rate of ransomware attack campaigns [13, 14, 15], such solutions are not likely to work for fog layer nodes since these nodes are not being in direct contact with end users. Existing detection approaches include signature-based and behavioral-based ransomware detection techniques. A detection system may consider static features (e.g. entropy of bytes, Program Executable - PE - imports, and ASCII printable strings) to distinguish malware, and a system with dynamic analysis usually focuses on the application's Windows API calls[16, 17] or network behavior [18]. Although static features can be very useful for characterizing malware samples, attackers can easily obfuscate the malware code to complicate static analysis. Also, most ransomware behavior detection solutions rely on filesystem [19, 20, 21] and registry events [22] to identify malicious behavior.

The use of machine learning to facilitate ransomware detection is becoming popular, for example in static analysis [23], dynamic analysis [16, 17] or hybrid analysis [24] of malware and normal applications. However, using deep learning algorithms to detect ransomware applications appear to be an understudied topic, despite its potential to extract useful features based on ransomware activities. This allows one to detect previously unseen ransomware samples.

Since system calls are of great importance in tracing events for determining malware behavior [17], we focus on detecting ransomware samples and characterizing their families by analyzing the sequence of actions taken by an application. We implement a *many to one* classifier by considering sequences of actions performed by goodware or ransomware samples as the inputs for predicting whether the sample is a ransomware and predicting their families accurately. We use *Long Short-Term Memory (LSTM)* and *Convolutional Neural Network (CNN)* in our proposed *Deep Ransomware Threat Hunting and Intelligence System (DRTHIS)*.

We implement the proposed system using Python2.7 and Python3.5, and using *Keras2.0.5* which provides fast experimentation with its high-level neural networks API. *Keras* [25] has the capability of running on top of *TensorFlow, CNTK* and *Theano* to make programming easier for deep learning researchers and developers. We use *Multi layer Perceptron (MLP)* with 3 layers of input, hidden and output in our comparative analysis. We set the input neurons to be equal to the padded length of sequences to feed one sequence at a time to the network. We also ensure that the number of hidden neurons is not more than twice the number of input neurons [26]. We then train and evaluate the performance of our proposed system using a dataset of 220 *Locky* samples, 220 *Cerber* samples, 220 *TeslaCrypt*, as well as benign samples from our previous research [27] augmented by 99 *CryptoWall* samples, 28 *TorrentLocker* samples and 77 *Sage*. To avoid overfitting our system, we use *Dropout* [28] as a regularization technique to prevent complex co-adaptations on the training data [29]. *Dropout* ignores randomly selected neurons during training in forward pass. In other words, the dropped out neurons are temporarily removed from the forward pass and any weight updates are not applied to the neuron on the backward pass.

1.1. Evaluation Metrics

We perform 10-fold cross validation to evaluate the proposed system. We achieve a mean F-Measure of 99.6% in the detection of ransomware samples and a mean true positive rate of 97.2% in the detection of their families in less than 10 seconds of launching an application. Moreover, it is shown that *DRTHIS* has the capability of identifying families of unseen ransomware samples. For testing samples from unseen families, *DRTHIS* correctly classifies 99% of *CryptoWall* samples, 75% of *TorrentLocker* samples and 92% of *Sage* samples as a new family of ransomware. We use True Positive (TP), False Positive (FP), True Negative (TN), and False Negative [30, 31] as the evaluation metrics. TP indicates the total of the samples that are correctly identified as a positive label, and FP shows the total negative samples incorrectly classified as positive. TN denotes the number of correctly rejected samples, while FN refers to incorrectly rejected samples. *Precision* (see below) reflects the positive predictive value by dividing TP by the total of FP and TP predicted by a classifier as shown in Equation (1). *Recall* shows the rate of positive samples that are correctly identified which is calculated by dividing TP by the total of TP and FN as shown in Equation (2). F-measure considers both *Precision* and *Recall* of test set in harmonic mean of precision and recall as shown in Equation (3).

$$Precision = \frac{TP}{TP + FP} \tag{1}$$

$$Recall = \frac{TP}{TP + FN} \tag{2}$$

$$F - measure = 2 \times \frac{Precision \times Recall}{Precision + Recall} \quad (3)$$

Although *Area Under the Curve (AUC)* is a measure of how well a parameter can be used to distinguish between two classes, there is no explicit formula to compute AUC [32]. *Matthews Correlation Coefficient (MCC)* [33] was first introduced to assess the performance of prediction in bio-informatics and provides a measure of quality to compare different classifiers [34]. The possible value of MCC is in $[-1, +1]$, where $+1$ indicates perfect prediction. In binary classifiers with total disagreement, the MCC value will be -1 while value of 0 shows random classification. MCC is also robust to imbalanced data [32]. While *Precision, Recall or F-measure* values in a random guessing would be higher than *0.5*, the MCC value would be around 0 for random guessing. Therefore, for making sure that our classifiers are far from random classifiers, we will compute MCC values for each classifier. These values can be computed using Equation (4).

$$MCC = \frac{TP \times TN - FP \times FN}{\sqrt{(TP + FP)(TP + FN)(TN + FP)(TN + FN)}} \quad (4)$$

The remainder of this paper is structured as follows. In Section 2, we review related research in ransomware detection. Section 3 presents our proposed *DRTHIS*. Section 4 describes how *DRTHIS* trains best binary classifier for threat hunting, and Section 5 describes the training of one class classifiers as well as deep feature extractor for threat intelligence. Section 6 presents the findings of the evaluation. Finally, we conclude the paper in Section 7.

2. Related Work

In recent years, ransomware has drawn the attention of information security researchers and practitioners as it is reportedly becoming a dominant tool for cybercriminals [35, 36].

Most ransomware samples detection solutions rely on the dynamic behavior of applications such as registry changes [22], and filesystem activities [19, 20, 21] to identify malicious applications. A study using 1359 ransomware samples revealed that the majority of ransomware samples are making similar API calls and similar filesystem activities [20]. For instance, *UNVEIL* [20] utilizes filesystem activities to classify ransomware from other types of malware with a 96.3% detection rate. Detecting abnormalities in filesystem operations is considered in *CloudRPS* [37] using leveraged anomalies in file system activities such as conversion of large quantities of files in a short interval to detect crypto-ransomware samples. *EldeRan* [38] considered the presence of activities in a sequence to build a matrix of activities to detect ransomware samples within 30 seconds of their execution with an AUC of 0.995.

Ahmadian et al., [22] proposed a *Bayesian Network* based model to identify ransomware samples from goodware with an *F-Measure* of 0.93 by using 20 different types of filesystem and registry events. Homayoun et al., [27] developed a ransomware detection system based on sequential pattern activities and achieved 99% accuracy. However, the suggested system was not capable of detecting previously unseen ransomware.

DL4MD [16] is a malware detection method, which uses a deep Stacked Autoencoders (SAEs) network for unsupervised feature learning that is followed by supervised parameter fine-tuning process to classify malware and goodware. The approach reportedly achieves an accuracy of 95% with a false positive of 2.3%. A deep learning approach [17] was proposed to benefit system calls for detecting malware with a precision rate of 85.6% and a recall rate of 89.4%.

Similar to [27], this paper attempts to classify a given sample as a ransomware/goodware in its first stage of infection. Specifically, we utilize a range of features for hunting ransomware even if they are from new unseen families. Moreover, identifying the family of a given ransomware is considered as a threat intelligence task in our work. Determining the family of a ransomware in a reasonable time would assist practitioners in following the threat profile for a given target.

3. Deep Ransomware Threat Hunting and Intelligence System (DRTHIS)

The proposed *DRTHIS* utilizes a binary classifier, a *Deep Feature Extractor (DFE)*, and a *One Class Classifier (OCCs)*, for hunting ransomware samples and identifying their families based on the application sequence of activities as shown in Figure 1.

When a user launches an application, the system records all executed events that within the first 10 seconds of application execution, and transforms the captured sequence to detect if a given sample is actually a ransomware (we refer to this stage as *Threat Hunting*). Afterward, identified ransomware samples are forwarded to the system to detect the ransomware family. During *Threat Intelligence*, we take advantage of the *DFE* component which uses a pre-trained deep model (*LSTM or CNN*) in its first stage to extract a vector to feed to the *OCCs*; thus, the capability to identify the family of the given ransomware sample.

DRTHIS performs a *Data Transformation* task to transform textual sequences of events into a numerical form. Then, the *Combining and Labeling* component combines input datasets into one integrated dataset suitable for our deep learning tasks. It is notable that *Combining and Labeling* creates two separate datasets with the same samples but different class labels. $TD_{Total(binary)}$ has two class labels of *Ransomware* and *Goodware*, and samples of $TD_{Total(family)}$ are labeled into four classes namely *Locky, Cerber, TeslaCrypt* and *Goodware*.

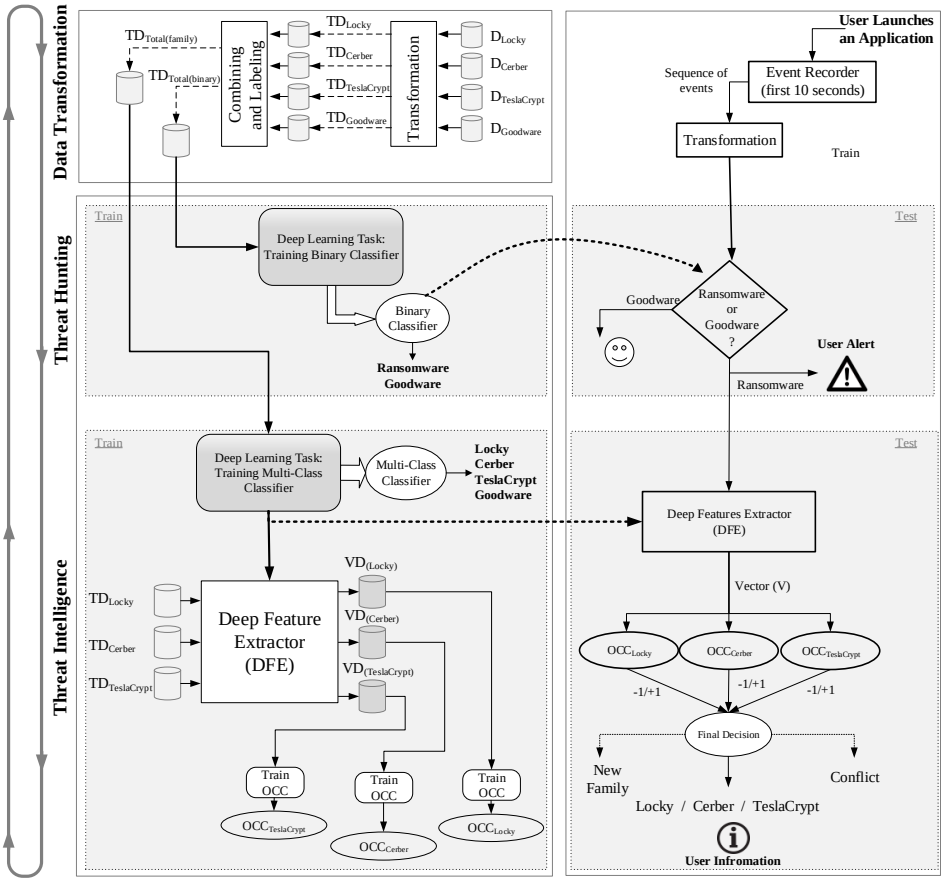


Figure 1: Proposed Deep Ransomware Threat hunting and Intelligence System (DRTHIS)

There are two *Deep Learning Tasks* (see Figure 1), which train a binary classifier for threat hunting and a multi-class classifier for using *DFE* component. The former uses $TD_{Total(binary)}$ to create a classifier to detect ransomware samples while the latter creates a classifier for separating different classes (families) of ransomware. Unlike *Threat Hunting* which directly benefits the binary classifier to detect ransomware, *Threat Intelligence* does not use the multi-class classifier to classify instances directly. In other words, *DFE* benefits trained *LSTM/CNN* to extract a vector of deep features from the given sequence. We will give the relevant details of *DFE* in Section 5.

Figure 2 shows the DRTHIS procedure. *Train, Test* and *EventRecorder* are the three main modules of *DRTHIS*. *Train* module (see Figure 3) returns the *Best Binary Classifier (BBC), Best Multiclass Classifier (BMC)*, and the list of trained one class classifiers *OCCs*. *Test* module in line 5 of Figure 2 returns the final decision on S, which is sequence of all captured events during the first 10 seconds of launching an application. Figure 3 describes the training procedure, where datasets DS are transformed in line 4. Both *BBC* and *BMC* are trained using deep learning tasks on $TD_{Total(binary)}$ and $TD_{Total(family)}$, respectively. *DFE* extracts vectored version of dataset D using trained multi class classifier (BMC). *Train* procedure creates one *OCC* for each *Vectored Dataset (VDS)* and returns *BBC, BMC* and list of *OCCList* as shown in line 2 of Figure 2.

Figure 4 presents the *DRTHIS* algorithm when it is hunting and capturing any intelligence of the new sequence of S of events executed by an application in its first 10 seconds of execution. *Test* procedure transforms the sequence using *SeqTransformation(S)*, which transforms a single sequence in comparison to the *Transformation(Dataset D)* that returns the transformed version of the dataset D. *Test* procedure alerts users when a ransomware is detected (hunting) and extracts its deep representation using *DFE(S)* to specify the family of S. *Final Decision* uses the output of the *OCCs* (R) to return F to show the gathered intelligence information.

DRTHIS utilizes the sequence (S) of events to differentiate ransomware from goodware. In this paper, an activity (A) on an argument ($argA$) is called an event. A set of sequential activities (A) on an argument ($argA$) by an application creates a sequence S in the form of $S_i = \{A_{1,i}(argA_1), A_{2,i}(argA_2), ..., A_{m,i}(argA_m)\}$, where S_i refers to the sequence of actions executed by the i^{th} sample, and $A_{x,y}(argA_x)$ represents the activity x for an application y, and $argA_x$ shows the argument passed to the

procedure DRTHIS(Dataset[] DS)
  $[BBC, BMC, OCCList] = Train(DS)$
  for each application started **do**
    $S = EventRecorder()$
    $Decision = Test(S, [BBC, BMC, OCCList])$
    print $Decision$
  end for
end procedure

Figure 2: DRTHIS for training threat hunting and intelligence components

procedure TRAIN(Dataset[] DS)
  $TDS = \{\}$ ▷ transformed datasets
  for all $D \in DS$ **do**
    $TD = Transfomration(D)$
    $TDS.add(TD)$
  end for
  $[TD_{Total(binary)}, TD_{Total(family)}] = Combine(TDS)$
  $BBC = TrainBinaryClassifier(TD_{Total(binary)})$
  $BMC = TrainMultiClassifier(TD_{Total(family)})$
  $VDS = \{\}$ ▷ vectored datasets
  for all $D \in DS$ **do**
    $VD = DFE(D, BMC)$
    $VDS.add(VD)$
  end for
  $OCCList = \{\}$ ▷ one class classifiers
  for all $VD \in VDS$ **do**
    $OCC = TrainOCC(VD)$
    $OCCList.add(OCC)$
  end for
  return $[BBC, BMC, OCCList]$
end procedure

Figure 3: Training components of DRTHIS

activity A_x. Therefore, we have $S_i = \{e_{i,1}, e_{i,2}, ..., e_{i,m}\}$, where m is the length of the sequence.

Our datasets are in the form of $D = \{S_1, S_2, ..., S_n\}$, where each S_i represents the sequence of events for the i^{th} sample. We create datasets listed in Table 1 to train and evaluate our models. D_{Locky}, D_{Cerber}, $D_{TeslaCrypt}$ and $D_{Goodware}$ refer to the dataset of sequences generated by *Locky*, *Cerber*, *TeslaCrypt* and benign applications, respectively.

We also create one dataset for each of ($D_{Test(Locky)}, D_{Test(Cerber)}$ and $D_{Test(TeslaCrypt)}$) ransomware families to evaluate the performance of our system in detecting unseen as well as over-fitting tests. It is notable that $D_{Test(x)}$ datasets are very new for our trained model because we will not use them in any steps related to training classifiers such as mapping and padding operations. We separated $D_{Test(x)}$ from the main dataset D_x because the *Mapping* process in the early stage of our model works based on frequency of each element in the dataset. If we don't consider separated

procedure TEST(Sequence S,BinaryClassifier BBC,MultiClassifier BMC,OneClassClassifier[] OCCList)
  $TS = SeqTransformation(S)$
  if $BBC(TS) == Goodware$ **then**
    return *Goodware*
  else
    Alarm(*"RansomwareAttack"*)
    $V = DFE(TS, BMC)$
    $R = \{\}$
    $R.add(OCCList_{Locky}(V))$
    $R.add(OCCList_{Cerber}(V))$
    $R.add(OCCList_{TeslaCrypt}(V))$
    $F = FinalDecision(R)$
    return F
  end if
end procedure

Figure 4: Testing DRTHIS for threat hunting and intelligence

Test dataset from the beginning stage of our model, test samples will be seen during *Mapping* process while we are going to test *DRTHIS* against real unseen samples. We use $D_{CryptoWall}, D_{TorrentLocker}$ and D_{Sage} to show each dataset of sequences for new families.

Table 1: Datasets of sequences for each family

Dataset	Number of Sequences
D_{Locky}	154
D_{Cerber}	154
$D_{TeslaCrypt}$	154
$D_{Goodware}$	153
$D_{Test(Locky)}$	66
$D_{Test(Cerber)}$	66
$D_{Test(TeslaCrypt)}$	66
$D_{Test(Goodware)}$	66
$D_{CryptoWall}$	99
$D_{TorrentLocker}$	28
D_{Sage}	77

As neural networks require numerical features, we have to pre-process and convert sequences (S_i) of events ($e_{i,j}$) into equivalent sequences of scalar values (MS_i) with mapped events ($me_{i,j}$)(see Figure 5). Since our sequences consist of text words as elements (malware activities), we need a way to convert these text elements into continuous values to feed our deep network.

Bag of Words, one of the most common techniques for converting text to numerical vectors, often results in sparse vectors [39]. The dimensionality of the vectors is equal to the size of the supported vocabulary, where there exists an element for each possible word. It is obvious that in most

cases there exists a huge word vocabulary; therefore, this vector has a long length in order to cover all words in the dictionary. The problem of long one-hot vector also exists for ransomware sequences, where many possible words (events) are available by combining activities and paths.

We first combine samples of D_{Locky}, D_{Cerber} and $D_{TeslaCrypt}$ into $D_{Ransomware}$, and rank all events by their frequency of occurrence in $D_{Ransomware}$ for mapping events into numerical representations for each sequence. Figure 5 depicts our steps in transforming textual events into numerically coded sequences. We create a dictionary containing all the events sorted in descending order according to their frequency of occurrence indexing from 1 to v, where v is the total number of words. Events with a frequency of 1 are removed from the dataset as they are not good for ransomware detection. *Mapping(S)* in line 4 of Figure 5 uses the dictionary of events and returns the mapped version of the input sequence S within the *MappedDataset*.

1: **procedure** TRANSFORMATION(Dataset D)
2: $MappedDataset = \{\}$
3: **for all** $S \in D$ **do**
4: $MS = Mapping(S)$
5: $MappedDataset.add(MS)$
6: **end for**
7: $TransformedDataset = \{\}$
8: **for all** $MS \in MappedDataset$ **do**
9: $PS = Padding(MS)$
10: $TransformedDataset.add(PS)$
11: **end for**
12: **return** $TransformedDataset$
13: **end procedure**

Figure 5: Transforming of all sequences of dataset D.

LSTM and *CNN* are two common techniques of deep learning among researchers. It is proved in the literature that *CNN* is very suitable for image machine learning tasks. However, CNN has the capability of working on sequential data because it employs sliding window (sliding vector), and a CNN with one or more 1D sliding window can extract precious features from sequences[40, 41, 42]. *LSTM* naturally works on sequential datasets[43, 44] and is very suitable for our project as we are considering sequences of activities for detecting ransomware samples. In fact we are testing both *LSTM* and *CNN* in our project to find the best one suitable for *DRTHIS*.

LSTM network is usually applied in time-series analysis [43] or sequences of data [45], such as city traffic forecasting [43], and driver distraction detection [46]. The historical events of ransomware can be viewed as a priori knowledge, so we can use *LSTM* to follow the temporal structure (sequence) of ransomware events. Figure 6 illustrates the steps taken to create a classifier to distinguish ransomware samples from benign applications.

Since we use the *Keras* deep learning framework that

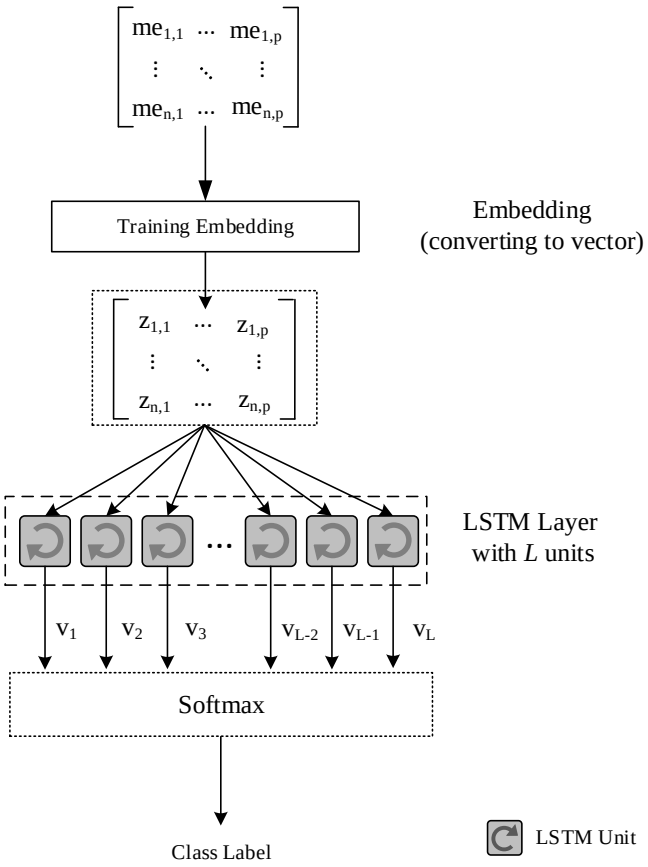


Figure 6: *LSTM* model for detecting ransomware samples

works based on *TensorFlow* for implementing our models, we perform a padding operation (*Padding(MS)*) in line 9 of Figure 5 to make all the mapped sequence (MS) with the same length into a padded sequence (PS). In the padding operation, any sequence longer than the padding size (p) is truncated, while shorter sequences are padded with zeroes to reach to the desired size. In theory, padding is not a must in *LSTMs* but it is in *CNNs*. As *LSTM* naturally works on sequential datasets, it has the capability of considering sequences with various lengths. However, in many deep learning implementation framework, padding is necessary for *LSTMs*. As we were going to use *Keras* framework for implementing our models, we had to use padding in our model. *CNNs* essentially requires a sample as a matrix (or vector for 1D convolution) to extract similar number of features for each sample.

After transformation, our data is in the form of an $(n \times p)$ matrix (dataset) for the training operation, where n is the total number of sequences and p is the padding size. From here on, we will use TD_x to refer to the transformed version of datasets, e.g. TD_{Locky} refers to the transformed version of D_{Locky}.

Combining and Labelling Data Transformation creates

$TD_{Total(binary)}$, which contains labeled samples as *Ransomware* and *Goodware*, and $TD_{Total(family)}$. These include all samples with *Locky, Cerber, TeslaCrypt* and *Goodware* class labels.

We use the *Embedding* technique to build a new vectored representation of numerical events in *Transformed-Dataset* that reflects the relationships between different events. *Embedding* provides contextual similarity; in other words, words that regularly occur nearby in text will also be in close proximity in the vectored space. *Embedding* complements the approach in *Word2Vec* [47] to learn a new representation of sequence values by randomly initializing, and training during back-propagation. If $S = \{me_1, me_2, ..., me_p\}$ is the input sequence, then the output of embedding with d dimensions is in the form of $\{Z_1, ..., Z_p\}$, where $Z_i = [z_{i,1}, z_{i,2}, ..., z_{i,d}]$ is the embedded vector of me_i and d is the embedding size.

Although the quality of word embedding increases with higher dimensionality, the gain will diminish after reaching some point [48, 49]. Therefore, we use embedding with size 8 in this paper. We use the median of the length of all sequences (5470) in our ransomware dataset ($D_{Ransomware}$) as the padding size. In the transformation phase of preprocessing, we only consider events with a total occurrence of more than one because there were too many events with a frequency of one (which does not contribute any useful information to this research). Therefore, in the mapping process, we map events with a frequency of one to zero in our sequences. We eventually have 23,372 unique events with a frequency of more than one in our dictionary of events.

The *LSTM* layer in Figure 6 consists of several *LSTM* units, which build dependencies between current and past events of a sequence. In fact, *LSTM* learns u best features that represent the sequence in a vectored manner, where u is the number of *LSTM* units in the *LSTM* layer.

Softmax [50] function is common as the final layer of neural networks. Softmax assigns a probability to each class to classify the instances, where it exponentially scales the values and normalizes them to sum up to 1. The class with the highest probability will be predicted as the final decision for any given data sample.

Since our focus is on detecting ransomware samples based on sequences of events, *CNN* can be a good solution for creating a high quality classifier. In other words, the relationship between different events of sequences is important in our approach, and *CNN* has the capability to extract these relationships into one or more new feature maps. Therefore, we use *CNN* over captured sequences of performed actions (events) of ransomware samples to extract valuable features for classification.

Figure 7 depicts our architecture used to train a classifier with the capability of detecting ransomware samples. We use *CNN* to extract relevant features into the *Fully Connected* layer for training our final classifier. Z_i represents the sequence of embedded events presented in Figure 7, and *Filter Size* is the size of the current filter that is convolving to make a new feature map. Since we have only one dimensional sequential, we use *1D-Convolution* that uses a vector (1D matrix) as its filter.

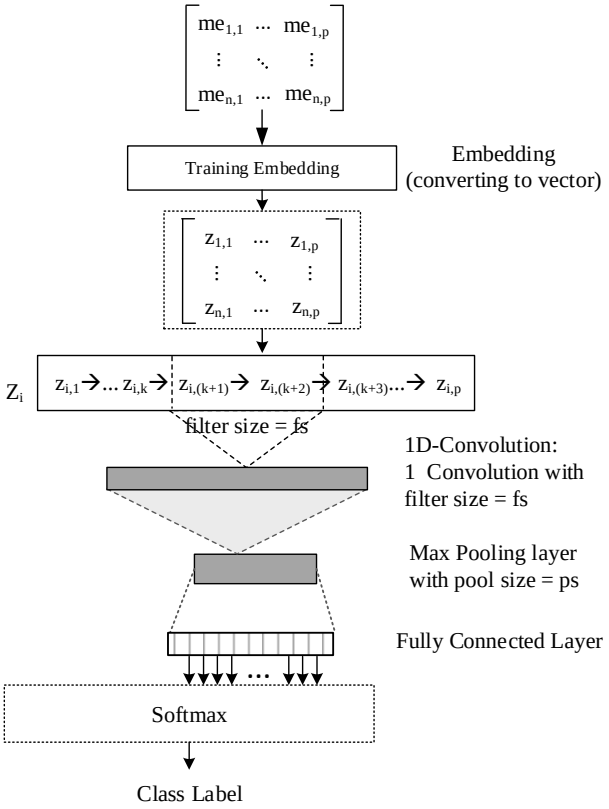


Figure 7: *CNN* model for detecting ransomware

4. Threat Hunting

We apply *LSTM* and *CNN* methods introduced in Figures 6 and 7 to train final binary classifiers for detecting (hunting) ransomware samples. We create $TD_{Total(binary)}$ dataset by combining $TD_{Ransomware}$ and $TD_{Goodware}$ to train our models for the binary classification of ransomware and goodware. It worth noting that we have a separate $TD_{Test(binary)}$ dataset for testing our final model, and $TD_{Test(binary)}$ does not contribute in the ranking and *Embedding* processes.

At first, we train our *LSTM* model shown in Figure 6 using only one unit in its hidden layer with $TD_{Total(bibary)}$. We also train *CNN* shown in Figure 7 with only one convolution filter of size 3 to obtain the final Softmax classifiers. Table 2 shows the performance of *LSTM* and *CNN*, where all values are rounded to three decimal places. As shown in Table 2, the trained model using *LSTM* with one *LSTM* unit has the potential to be under-fitting although it achieves high *TPR* (of 0.951). This is because, it generates a high false positive (of 0.324) as well. *CNN* achieves lower *FPR* (0.036) in comparison with *LSTM*, while its

MCC value suggests that it is trained as a good predictor for separating ransomware from benign applications.

Table 2: Performance results on dataset $TD_{Test(binary)}$ for *LSTM* with one *LSTM* unit in hidden layer and *CNN* with only one convolution filter

	F-Measure	**TPR**	**FPR**	**MCC**
LSTM	0.951	1	0.324	0.717
CNN	0.985	0.982	0.036	0.945

Higher performance metrics for *CNN* are not unexpected because the Softmax layer of Figure 7 trains the final classifier based on 2,735 features output from a fully connected layer that extracted by *CNN*. On the other hand, the Softmax layer of Figure 6 with one *LSTM* unit is trained with only one feature. In other words, all events of a sequence are fed to the *LSTM* unit, and after feeding the last event, the value of this single unit is forwarded to the next layer for training in the Softmax layer. Therefore, we train our *LSTM* model with more *LSTM* units.

Figures 8 and 9 depict the performance of *LSTM* networks with varying numbers of *LSTM* units from 2 to 20 increasing by 2. As Figures 8 and 9 show, *LSTM* achieves better results when it uses more units of *LSTM* in its hidden layer. The rate of false positives is reduced to 0.16 for *LSTM* with 2 units in comparison with a *LSTM* with one unit only. We observe a significant reduction in *FPR* for a trained model with 4 units of *LSTM*, and it reduces to a zero percent false positive rate over TD_{Test}. Figure 8 also shows that having too many *LSTM* units may over-fit the final classifier as the performance metrics decrease by increasing the number of *LSTM* units.

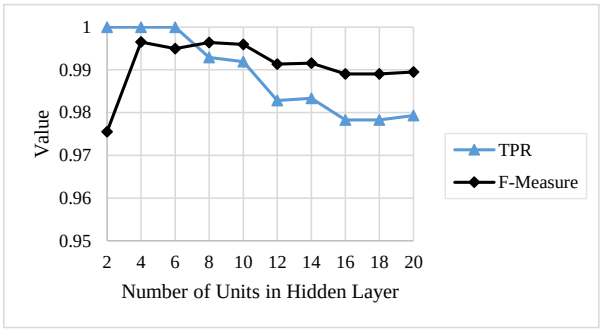


Figure 8: Performance of *LSTM* model on TD_{Test} for different numbers of *LSTM* units in hidden layer

We then design a *MLP* with an input size of 5470 (size of padded sequence) to compare the results of simple *MLP* with deep architecture in detecting ransomware from goodware. To determine the most appropriate numbers of hidden units in the *MLP*, we evaluate the network with neurons from 1000 to 10000, and findings suggest that *MLP* with 4000 hidden neurons results in the best *F-Measure*.

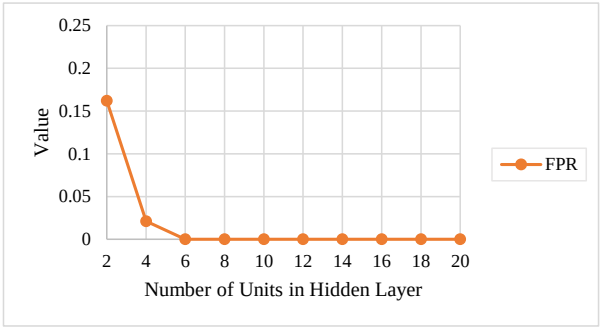


Figure 9: False Positive Rate on TD_{Test} for different numbers of *LSTM* units in hidden layer

As expected (see Table 3), *LSTM* with 8 units in the hidden layer outperforms both *CNN* and *MLP*. This is mainly because an *LSTM* with more units may extract more features to create a higher quality classifier. Therefore, *DRTHIS* uses a classifier trained by an *LSTM* network with 8 units of *LSTM*. Moreover, the high false positive rate of *MLP* (11.3%) may reflect the fact that *MLP* is not suitable for detecting sequential relations between different sequences of events.

Table 3: Performance of our models on TD_{Test}

	F-Measure	**TPR**	**FPR**	**MCC**
LSTM	0.996	0.992	0	0.986
CNN	0.985	0.982	0.036	0.945
MLP	0.958	0.956	0.113	0.837

5. Threat Intelligence

Unlike *Threat Hunting* which separates ransomware from goodware, the deep learning task of *Threat Intelligence* trains a Softmax classifier to separate samples from 4 class labels of $D_{Total(family)}$. In fact, the objective of the deep learning task is to create the best classifier for identifying known families, while *DFE* extracts features from a trained *LSTM* into a vector before feeding to Softmax for the final classification ($[v_1, v_2, v_3, ..., v_{L-2}, v_{L-1}, v_L]$, as shown in Figure 6). We simply use classification performance metrics for Softmax predictions to evaluate and find the best *LSTM* model for extracting features, while *DRTHIS* does not use the created multi-class classifier for threat intelligence.

We apply the same architecture as shown in Figures 6 and 7 on $D_{Total(family)}$ with 4 class labels to find the best

LSTM and *CNN* models respectively. Table 4 presents the findings after training *LSTM* with one hidden unit and *CNN* with one convolution filter with size 3. *LSTM* displays poor outcomes in terms of identifying families of ransomware while the created classifier based on *CNN* extracted features works much better than *LSTM* with one hidden unit.

Table 4: Performance of LSTM and CNN for identifying ransomware family $TD_{Test(family)}$

	TPR	**FPR**
LSTM	0.501	0.498
CNN	0.899	0.100

We increase the number of hidden units in *LSTM* layer and calculate *TPR, FPR* and *F-Measure* in Figure 10. The figure shows the highest performance for an *LSTM* with 18 units.

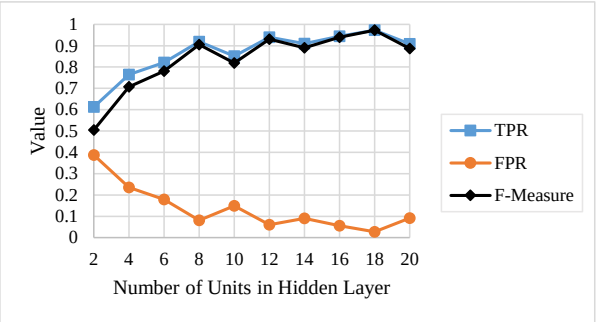


Figure 10: Performance of our models on TD_{Test}

Table 5 represents the best *F-Measure* achieved with *LSTM, CNN* and *MLP* on $TD_{Test(family)}$. We determine that *MLP* with 2000 hidden neurons offers the best performance, while *MLP* still suffers from a high false positive rate. The latter only identifies the family of 66.8% ransomware correctly. This could be because of the inability of *MLP* to consider sequential relations between different events of sequences.

LSTM with multiple units may find more discriminative features and hence results in the highest *TPR* with lowest *FPR*. Table 5 and Figure 10 show that the best model for *DFE* component is a trained *LSTM* with 18 units.

We feed every sample of each family to a *DFE* component, and store an equivalent *Vector (V)* containing values of units after feeding the last event of the given sequence. We extract *Vs* of D_{Locky}, D_{Cerber}, $D_{TeslaCrypt}$, $D_{Test(Locky)}$, $D_{Test(Cerber)}$ and $D_{(Test)TeslaCrypt}$ to create *Vectored Datasets (VD)* of VD_{Locky},VD_{Cerber}, $VD_{TeslaCrypt}$, $VD_{Test(Locky)}$, $VD_{Test(Cerber)}$ and $VD_{Test(TeslaCrypt)}$.

Table 5: Performance of our models for identifying ransomware family

	TPR	**FPR**
LSTM	0.972	0.027
CNN	0.899	0.100
MLP	0.668	0.331

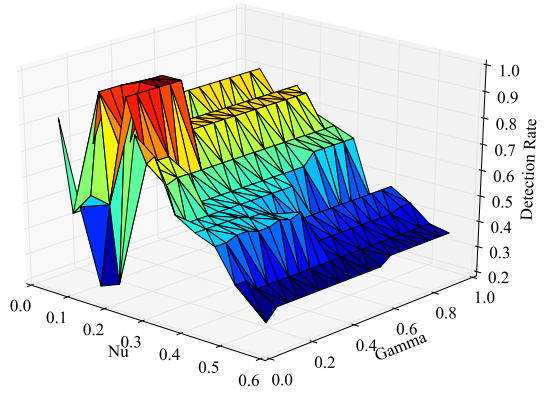
DRTHIS attempts to identify ransomware families using 3 trained one class classifiers (*OCCs*). We use *One Class Support Vector Machine (OCSVM)* for training *OCCs* as a powerful one class classifier algorithm [51]. We create a separate classifier for each ransomware family using a dataset $VD_{(x)}$, each of which has the capability of determining whether a sample belongs to a trained family class x.

SVMs [52] are supervised learning models that can be used for both classification and regression tasks. The *SVM* algorithm represents each sample as a point in space and tries to find a separator with a gap to separate samples from different classes. *OCSVM* may be viewed as a regular two-class *SVM*, where all training data samples are in the first class, and the origin is taken as the only member of the second class [53].

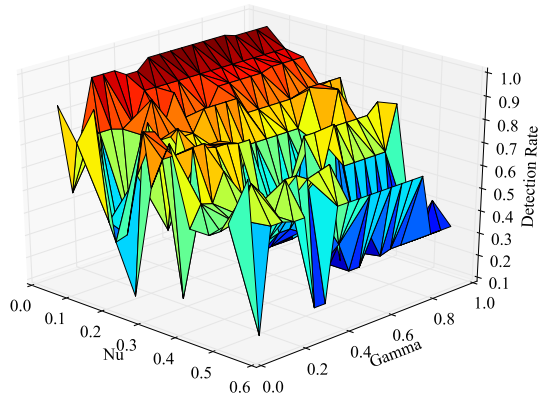
We use 3 one class classifiers for the 3 ransomware families (i.e. *Locky, Cerber* and *TeslaCrypt*). OCC_{Locky} in Figure 1 presents the *OCC* classifier trained using $VD_{(Locky)}$ to detect *Locky* samples, OCC_{Cerber} shows *OCC* classifier trained by samples from $VD_{(Cerber)}$ to identify *Cerber* samples, and $OCC_{TeslaCrypt}$ represents *OCC* classifier trained from $VD_{(TeslaCrypt)}$ to detect *TeslaCrypt* samples. Output of OCC_x is +1 if the sample is classified as x; otherwise, it returns -1.

Nu and *Gamma* are two important parameters for executing *OCSVM*. *Nu* can be between $(0, 1]$; that is, an upper bound on the fraction of margin errors and a lower bound of the fraction of support vectors relative to the total number of training examples. *Gamma* is the Kernel coefficient for *RBF* kernel using *SVM*. As *Nu* and *Gamma* affect the performance of created classifiers [53], we train *OCSVM* with different values of *Nu* and *Gamma* to find the best classifiers for *DRTHIS*. Figure 11 depicts the detection ratio for each family on classifiers trained by different values of *Nu* and *Gamma*. Blue colors in Figure 11 represent low detection rates while red surfaces indicate higher detection rates. As shown in Figure 11a, the highest detection rate for *Locky* samples is at *Nu=0.15* and *Gamma=0.3*. Both Figures 11b and 11c suggest that *Nu=0.05* and *Gamma=0.55* are the best values to achieve highest possible detection rates.

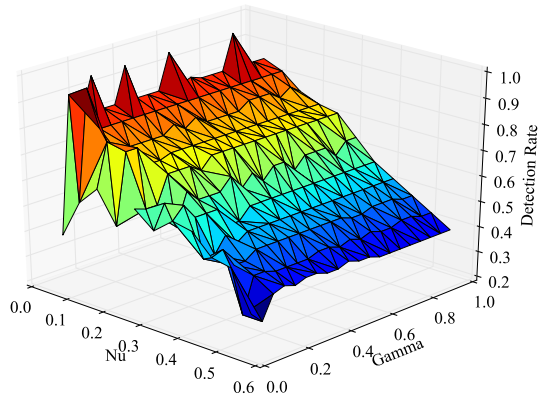
Equation 5 shows *Final Decision* after receiving outputs from *OCCs*, where F is the final decision based on the input set R containing outputs of *OCCs*. *Final Decision* determines a class label (i.e. family of given ransomware) or specifies it as a new family of ransomware. In the case



(a) Detection Rate for Locky test samples on $TD_{Test(Locky)}$



(b) Detection Rate for Cerber test samples on $TD_{Test(Cerber)}$



(c) Detection Rate for TeslaCrypt test samples $TD_{Test(TeslaCrypt)}$

Figure 11: Detection Ratio for different values of Nu and Gamma of OCSVM over $TD_{Test(x)}$

where a sample is a candidate for more than one family, *OCSVM* (more than one +1 in R), *Final Decision* returns a *Conflict*. Hence, if and only if one of *Final Decision* is +1 for a sample, then the output will be the class label of the sample family ($Family(x)$ in Equation 5). If all of the inputs of the *Final Decision* component are -1 for a given sample, then it will be classified as a sample belonging to a new family.

$$F = \begin{cases} Family(x), & \exists_{x \in R} \forall_{y \in R \wedge y \neq x} (x = +1 \wedge y = -1) \\ New, & \forall_{x \in R} (x = -1) \\ Conflict, & \exists_{x \in R} \exists_{y \in R \wedge y \neq x} (x = +1 \wedge y = +1 \end{cases} \quad (5)$$

6. Evaluation and Discussion

We evaluate the performance of *DRTHIS* using both ransomware samples from new families and unforeseen benign applications.

Table 6 presents the classification result after applying *DRTHIS* on $TD_{Goodware}$, TD_{Locky}, TD_{Cerber} and $TD_{TeslaCrypt}$. In the table, the *New* column represents samples that do not belong to any family (ransomware belonging to unseen families) while the number of samples detected in more than one family are listed in the *Conflict* column. Only two *Cerber* samples encounter conflicts after applying *DRTHIS* on $TD_{Test(Locky)}$, $TD_{Test(Cerber)}$ and $TD_{Test(TeslaCrypt)}$.

Classifying 16 out of 66 *Locky* samples as *Goodware* means that our created *One Class* classifier generated 24% wrong prediction over $TD_{Test(Locky)}$. Moreover, one *Cerber* sample and two *TeslaCrypt* samples are wrongly classified as a new family of ransomware.

Table 6: Confusion matrix calculated after applying DRTHIS on samples from TD_{Test}
LC:Locky,CB:Cerber,TC:TeslaCrypt,GW:Goodware

	LC	CB	TC	GW	New	Conflicts	Total
LC	50	0	0	16	0	0	66
CB	0	60	0	3	1	2	66
TC	0	0	64	0	2	0	66
GW	0	0	0	66	0	0	66

DRTHIS takes the advantage of *One Class Classifier* to determine if a sample belongs to a known family of ransomware or whether it belongs to a new family. We use samples from *CryptoWall*, *TorrentLocker* and *Sage* families for evaluating the performance of our system against samples from unforeseen families.

Table 7 presents a confusion matrix after applying *DRTHIS* over $TD_{CryptoWall}$, $TD_{TorrenLocker}$ and TD_{Sage}. It is clear that *DRTHIS* identifies these three families as a new family without any conflict with the trained families. 99% of *CryptoWall*, 75% of *TorrentLocker* and 92% of *Sage* samples are correctly detected as samples from a new

family of ransomware. *DRTHIS* wrongly classifies 1 *CryptoWall* sample (1%), 4 *TorrentLocker* samples (14.2%), and 1 *Sage* sample (1.2%) as *Goodware*. *DRTHIS* identifies 2 *TorrentLocker* samples (7%) and 1 *Sage* sample (1.2%) as *Cerber* samples. Three samples (3.8%) of *Sage* and 1 sample (3.5%) of *TorrentLocker* are also detected as *TeslaCrypt*.

Table 7: Confusion matrix calculated after applying our detection system on samples from studied families
LC:Locky,CB:Cerber,TC:TeslaCrypt,GW:Goodware, CW:CryptoWall, TL:TorrentLocker,SG:Sage

	LC	**CB**	**TC**	**GW**	**New**	**Conflicts**	**Total**
CW	0	0	0	1	98	0	99
TL	0	2	1	4	21	0	28
SG	1	1	3	1	71	0	77

7. Conclusion and Future Work

Fog computing will be increasingly commonplace and with fog nodes having more computational and storage capabilities (e.g. due to advances in technologies), fog nodes will be an attractive target for ransomware (and other attacks).

In this paper, we presented our proposed *DRTHIS* designed to detect ransomware and identify the family of the ransomware within the first 10 seconds of an application execution. In other words, the proposed system can be deployed on the fog layer to serve as a fully automated ransomware detection mechanism. Findings from our evaluations demonstrated that *LSTM* with 8 units results in a more powerful binary classifier in comparison with *CNN* for hunting ransomware. This is because *LSTM* explicitly extracts features from sequential activities to distinguish ransomware. *DRTHIS* uses a pre-trained *LSTM* model with multiple class labels in the *DFE* for vectorizing a sequence of activities into a vector to feed into *One Class Classifiers*. Our evaluation also showed that the multi-class classifier used in the *DFE* component of our threat intelligence agent works much better when it is trained with an *LSTM* network with 18 units of *LSTM* in comparison with the *CNN* technique. Specifically, we trained and evaluated our models using 220 *Locky* ransomware samples, 220 *Cerber* ransomware samples and 220 samples of *TeslaCrypt* ransomware, as well as evaluating our model with new samples of ransomware belonging to unforeseen families of *CryptoWall*, *TorrentLocker* and *Sage*. We achieved an F-Measure of 0.996 in detecting ransomware samples with a true positive rate of 0.972 and a false positive rate of 0.027 in identifying the ransomware family.

Findings from our studies suggested the potential of applying deep learning techniques in achieving better results in comparison with traditional neural networks to detect ransomware. Future work will include exploring the utility of other deep learning techniques, such as the Sequence-discriminative training of *Deep Neural Networks(DNNs)*, and *Ensemble Deep Neural Network(DNN)/CNN/RNN*, in ransomware detection.

As *DRTHIS* has the capability of fast classification of new instances, it can be considered as a basic method in cyber security industry for implementing new threat hunting and intelligence tools. Moreover, *DRTHIS* can detect samples from new emerged families of ransomware, and it makes *DRTHIS* a general and suitable solution for ransomware detection in practice.

Acknowledgment

he authors would like to thank *ransomwaretracker.abuse.ch* and *virustotal.com* for their support of this research. This work was partially supported by the European Council International Incoming Fellowship (FP7-PEOPLE-2013-IIF) grant number 625402, and the Cloud Technology Endowed Professorship. The information and views set out in this paper are those of the authors and do not necessarily reflect the official opinion of institutes they are working at.

References

[1] EUROPOL, The internet organised crime threat assessment (iocta) 2016 (2016).
URL https://www.europol.europa.eu/activities-services/main-reports/internet-organised-crime-threat-assessment-iocta-2016

[2] ENISA, ENISA Threat Landscape Report 2016: 15 Top Cyber-Threats And Trends, Tech. rep. (January 2017). doi:10.2824/92184.
URL https://www.enisa.europa.eu/news/enisa-news/enisa-threat-landscape-2016-report-cyber-threats-becoming-top-priority{%}0Ahttps://www.enisa.europa.eu/publications/enisa-threat-landscape-report-2016{%}0A

[3] TrendMicro, The reign of ransomware, Tech. rep. (2016).
URL http://www.trendmicro.com/cloud-content/us/pdfs/security-intelligence/reports/rpt-the-reign-of-ransomware.pdf

[4] H. L. Kevin Savage, Peter Coogan, The evolution of ransomware, Symantec, 2015.

[5] Look into locky ransomware - malwarebytes labs — malwarebytes labs, https://blog.malwarebytes.com/threat-analysis/2016/03/look-into-locky/, (Accessed on 09/08/2017) (Mar 2016).

[6] Cerber ransomware - new, but mature - malwarebytes labs — malwarebytes labs, https://blog.malwarebytes.com/threat-analysis/2016/03/cerber-ransomware-new-but-mature/, (Accessed on 09/08/2017) (Mar 2016).

[7] J. Wyke, A. Ajjan, The Current State of Ransomware, Tech. Rep. December, Sophos (2015).
URL https://www.sophos.com/en-us/medialibrary/PDFs/technicalpapers/sophos-current-state-of-ransomware.pdf?la=en

[8] K.-K. R. Choo, R. Lu, L. Chen, X. Yi, A foggy research future: Advances and future opportunities in fog computing research, Future Generation Computer Systems 78 (2018) 677–679. doi:10.1016/j.future.2017.09.014.
URL https://doi.org/10.1016/j.future.2017.09.014

[9] Gartner says 8.4 billion connected "things" will be in use in 2017, up 31 percent from 2016, http://www.gartner.com/newsroom/id/3598917, (Accessed on 09/25/2017) (Feb 2017).

[10] C. Huang, R. Lu, K.-K. R. Choo, Vehicular fog computing: Architecture, use case, and security and forensic challenges, IEEE Communications Magazine 55 (2017) 105–111. doi:10.1016/j.future.2017.09.014.
URL https://doi.org/10.1016/j.future.2017.09.014

[11] A. Azmoodeh, A. Dehghantanha, M. Conti, K.-K. R. Choo, Detecting crypto-ransomware in IoT networks based on energy consumption footprint, Journal of Ambient Intelligence and Humanized Computingdoi:10.1007/s12652-017-0558-5.
URL https://doi.org/10.1007/s12652-017-0558-5

[12] M. Conti, A. Dehghantanha, K. Franke, S. Watson, Internet of things security and forensics: Challenges and opportunities, Future Generation Computer Systems 78 (2018) 544–546. doi:10.1016/j.future.2017.07.060.
URL https://doi.org/10.1016/j.future.2017.07.060

[13] X. Luo, Q. Liao, Awareness education as the key to ransomware prevention, Information Systems Security 16 (4) (2007) 195–202. doi:10.1080/10658980701576412.
URL https://doi.org/10.1080%2F10658980701576412

[14] G. L. White, Education and prevention relationships on security incidents for home computers, Journal of Computer Information Systems 55 (3) (2015) 29–37. doi:10.1080/08874417.2015.11645769.
URL https://doi.org/10.1080/08874417.2015.11645769

[15] G. White, T. Ekin, L. Visinescu, Analysis of protective behavior and security incidents for home computers, Journal of Computer Information Systems 57 (4) (2016) 353–363. doi:10.1080/08874417.2016.1232991.
URL https://doi.org/10.1080/08874417.2016.1232991

[16] W. Hardy, L. Chen, S. Hou, Y. Ye, X. Li, Dl4md: A deep learning framework for intelligent malware detection, in: Proceedings of the International Conference on Data Mining (DMIN), The Steering Committee of The World Congress in Computer Science, Computer Engineering and Applied Computing (WorldComp), 2016, p. 61.

[17] B. Kolosnjaji, A. Zarras, G. Webster, C. Eckert, Deep learning for classification of malware system call sequences, in: AI 2016: Advances in Artificial Intelligence, Springer International Publishing, 2016, pp. 137–149. doi:10.1007/978-3-319-50127-7_11.
URL https://doi.org/10.1007/978-3-319-50127-7_11

[18] H. Location, L. Carnaudo, M. Mellia, E. Baralis, S. Saha, S. Miskovic, G. Modelo-Howard, S.-J. Lee, Magma network behavior classifier for malware traffic, Computer Networks 109 (Part 2) (2016) 142 – 156, traffic and Performance in the Big Data Era. doi:https://doi.org/10.1016/j.comnet.2016.03.021.
URL http://www.sciencedirect.com/science/article/pii/S1389128616300949

[19] A. Kharraz, W. Robertson, D. Balzarotti, L. Bilge, E. Kirda, Cutting the gordian knot: A look under the hood of ransomware attacks, in: Detection of Intrusions and Malware, and Vulnerability Assessment, Springer Nature, 2015, pp. 3–24. doi:10.1007/978-3-319-20550-2_1.
URL https://doi.org/10.1007%2F978-3-319-20550-2_1

[20] A. Kharaz, S. Arshad, C. Mulliner, W. Robertson, E. Kirda, Unveil: A large-scale, automated approach to detecting ransomware, in: 25th USENIX Security Symposium (USENIX Security 16), USENIX Association, Austin, TX, 2016, pp. 757–772.
URL https://www.usenix.org/conference/usenixsecurity16/technical-sessions/presentation/kharaz

[21] C. Moore, Detecting ransomware with honeypot techniques, in: 2016 Cybersecurity and Cyberforensics Conference (CCC), Institute of Electrical and Electronics Engineers (IEEE), 2016. doi:10.1109/ccc.2016.14.
URL https://doi.org/10.1109%2Fccc.2016.14

[22] M. M. Ahmadian, H. R. Shahriari, 2entfox: A framework for high survivable ransomwares detection, in: 2016 13th International Iranian Society of Cryptology Conference on Information Security and Cryptology (ISCISC), Institute of Electrical and Electronics Engineers (IEEE), 2016. doi:10.1109/iscisc.2016.7736455.
URL https://doi.org/10.1109%2Fiscisc.2016.7736455

[23] J. Saxe, K. Berlin, Deep neural network based malware detection using two dimensional binary program features, in: 2015 10th International Conference on Malicious and Unwanted Software (MALWARE), 2015, pp. 11–20. doi:10.1109/MALWARE.2015.7413680.

[24] Z. Yuan, Y. Lu, Y. Xue, Droiddetector: android malware characterization and detection using deep learning, Tsinghua Science and Technology 21 (1) (2016) 114–123. doi:10.1109/TST.2016.7399288.

[25] Keras documentation, https://keras.io/, (Accessed on 09/25/2017).

[26] M. T. Hagan, H. B. Demuth, M. H. Beale, Neural Network Design, Martin Hagan, 2002.

[27] S. Homayoun, A. Dehghantanha, M. Ahmadzadeh, S. Hashemi, R. Khayami, Know abnormal, find evil: Frequent pattern mining for ransomware threat hunting and intelligence, IEEE Transactions on Emerging Topics in Computing (2017 - In Press) 1–1doi:10.1109/tetc.2017.2756908.
URL https://doi.org/10.1109/tetc.2017.2756908

[28] N. Srivastava, G. Hinton, A. Krizhevsky, I. Sutskever, R. Salakhutdinov, Dropout: A simple way to prevent neural networks from overfitting, Journal of Machine Learning Research 15 (2014) 1929–1958.
URL http://jmlr.org/papers/v15/srivastava14a.html

[29] G. E. Hinton, N. Srivastava, A. Krizhevsky, I. Sutskever,

R. Salakhutdinov, Improving neural networks by preventing co-adaptation of feature detectors, CoRR abs/1207.0580.
URL http://arxiv.org/abs/1207.0580

[30] M. Sun, X. Li, J. C. S. Lui, R. T. B. Ma, Z. Liang, Monet: A user-oriented behavior-based malware variants detection system for android, IEEE Transactions on Information Forensics and Security 12 (5) (2017) 1103–1112. doi:10.1109/tifs.2016.2646641.
URL https://doi.org/10.1109%2Ftifs.2016.2646641

[31] M. R. Watson, N. ul-hassan Shirazi, A. K. Marnerides, A. Mauthe, D. Hutchison, Malware detection in cloud computing infrastructures, IEEE Transactions on Dependable and Secure Computing 13 (2) (2016) 192–205. doi:10.1109/tdsc.2015.2457918.
URL https://doi.org/10.1109%2Ftdsc.2015.2457918

[32] S. Boughorbel, F. Jarray, M. El-Anbari, Optimal classifier for imbalanced data using matthews correlation coefficient metric, PLOS ONE 12 (6) (2017) e0177678. doi:10.1371/journal.pone.0177678.
URL https://doi.org/10.1371/journal.pone.0177678

[33] B. Matthews, Comparison of the predicted and observed secondary structure of t4 phage lysozyme, Biochimica et Biophysica Acta (BBA) - Protein Structure 405 (2) (1975) 442–451. doi:10.1016/0005-2795(75)90109-9.
URL https://doi.org/10.1016%2F0005-2795%2875%2990109-9

[34] D. M. Powers, Evaluation: from precision, recall and f-measure to roc, informedness, markedness and correlation, Tech. rep., School of Informatics and Engineering-Flinders University (2011).

[35] Ransomware becomes most popular form of attack as payouts approach $1bn a year, Network Security 2017 (1) (2017) 1–2. doi:10.1016/s1353-4858(17)30001-6.
URL https://doi.org/10.1016/s1353-4858(17)30001-6

[36] UK major target for ransomware, Computer Fraud & Security 2016 (1) (2016) 3. doi:10.1016/s1361-3723(16)30003-3.
URL https://doi.org/10.1016/s1361-3723(16)30003-3

[37] J. K. Lee, S. Y. Moon, J. H. Park, CloudRPS: a cloud analysis based enhanced ransomware prevention system, The Journal of Supercomputingdoi:10.1007/s11227-016-1825-5.
URL https://doi.org/10.1007/s11227-016-1825-5

[38] D. Sgandurra, L. Muñoz-González, R. Mohsen, E. C. Lupu, Automated dynamic analysis of ransomware: Benefits, limitations and use for detection, CoRR abs/1609.03020.
URL http://arxiv.org/abs/1609.03020

[39] R. Zhao, K. Mao, Fuzzy bag-of-words model for document representation, IEEE Transactions on Fuzzy Systems (2017) 1–1doi:10.1109/tfuzz.2017.2690222.
URL https://doi.org/10.1109/tfuzz.2017.2690222

[40] T. Chen, R. Xuab, Y. Hec, X. Wang, Improving sentiment analysis via sentence type classification using bilstm-crf and cnn, Expert Systems with Applications 72 (2017) 221–230. doi:10.1016/j.eswa.2016.10.065.
URL https://doi.org/10.1016/j.eswa.2016.10.065

[41] H. Zeng, M. D. Edwards, G. Liu, D. K. Gifford, Convolutional neural network architectures for predicting dnaprotein binding, Bioinformatics 32 (12) (2016) 121–127. doi:10.1093/bioinformatics/btw255.
URL http://dx.doi.org/10.1093/bioinformatics/btw255

[42] N. G. Nguyen, V. A. Tran, D. L. Ngo, D. Phan, F. R. Lumbanraja, M. R. Faisal, B. Abapihi, M. Kubo, K. Satou, Dna sequence classification by convolutional neural network, Journal of Biomedical Science and Engineering 9 (2016) 280–286. doi:10.4236/jbise.2016.95021.
URL http://dx.doi.org/10.4236/jbise.2016.95021

[43] Z. Zhao, W. Chen, X. Wu, P. C. Y. Chen, J. Liu, LSTM network: a deep learning approach for short-term traffic forecast, IET Intelligent Transport Systems 11 (2) (2017) 68–75. doi:10.1049/iet-its.2016.0208.
URL https://doi.org/10.1049/iet-its.2016.0208

[44] Y. Yan, Y. Wang, W.-C. Gao, B.-W. Zhang, C. Yang, X.-C. Yin, Lstm 2: Multi-label ranking for document classification.

[45] T. Chen, R. Xu, Y. He, X. Wang, Improving sentiment analysis via sentence type classification using BiLSTM-CRF and CNN, Expert Systems with Applications 72 (2017) 221–230. doi:10.1016/j.eswa.2016.10.065.
URL https://doi.org/10.1016/j.eswa.2016.10.065

[46] M. Wollmer, C. Blaschke, T. Schindl, B. Schuller, B. Farber, S. Mayer, B. Trefflich, Online driver distraction detection using long short-term memory, IEEE Transactions on Intelligent Transportation Systems 12 (2) (2011) 574–582. doi:10.1109/tits.2011.2119483.
URL https://doi.org/10.1109/tits.2011.2119483

[47] T. Mikolov, I. Sutskever, K. Chen, G. Corrado, J. Dean, Distributed representations of words and phrases and their compositionality, CoRR abs/1310.4546.
URL http://arxiv.org/abs/1310.4546

[48] Vector representations of words — tensorflow, https://www.tensorflow.org/tutorials/word2vec, (Accessed on 08/05/2017) (Jun 2017).

[49] T. Mikolov, K. Chen, G. Corrado, J. Dean, Efficient estimation of word representations in vector space, ArXiv e-printsarXiv:1301.3781.

[50] B. Christopher, Pattern Recognition and Machine Learning, 1st Edition, Springer-Verlag New York, 2006.

[51] J. Shawe-Taylor, B. Žličar, Novelty Detection with One-Class Support Vector Machines, Springer International Publishing, Cham, 2015, pp. 231–257. doi:10.1007/978-3-319-17377-1_24.
URL https://doi.org/10.1007/978-3-319-17377-1_24

[52] C. Cortes, V. Vapnik, Support-vector networks, Machine Learning 20 (3) (1995) 273–297. doi:10.1007/bf00994018.
URL https://doi.org/10.1007/bf00994018

[53] Y. Xiao, H. Wang, W. Xu, Parameter selection of gaussian kernel for one-class SVM, IEEE Transactions on Cybernetics 45 (5) (2015) 941–953. doi:10.1109/tcyb.2014.2340433.
URL https://doi.org/10.1109/tcyb.2014.2340433