Stealthy Malware Detection using RNN-based Automated Localized Feature Extraction and Classifier

Sanket Shukla[1], Gaurav Kolhe[2], Sai Manoj P D[2] , Setareh Rafatirad[1]
Email: {sshukla4, gkolhe, spudukot, srafatir}@gmu.edu
[1]Department of Information Sciences and Technology
[2]Department of Electrical and Computer Engineering
George Mason University, Fairfax VA, USA 22030

Abstract*—Malware analysis, detection and classification has allured a lot of researchers in the past few years. Numerous methods based on machine learning (ML), computer vision and deep learning have been applied to this task and have accomplished some pragmatic results. One of the basic assumption of these works is that malware is spawned as a separate thread and the distinguishing features can be extracted in a "clean" manner irrespective of the malware obfuscation deployed. However, this assumption does not hold true for the advanced malware obfuscation techniques such as code relocation, mutation and polymorphism. Stealthy malware is a malware created by embedding the malware in a benign application through advanced obfuscation strategies to thwart the detection. To perform efficient malware detection for traditional and stealthy malware alike, we propose a two-pronged approach. Firstly, we extract the microarchitectural traces obtained while executing the application, which are fed to the traditional ML classifiers to detect malware spawned as separate thread. In parallel, for an efficient stealthy malware detection, we introduce an automated localized feature extraction technique that will be further processed using the recurrent neural networks (RNNs) for classification. To perform this, we translate the application binaries into images and further convert it into sequences and extract local features for stealthy malware detection. With the proposed two-pronged approach, an accuracy of 94% and nearly 90% is achieved in detecting normal and stealthy malware created through code relocation obfuscation technique. Furthermore, the proposed approach achieves up to 11% higher detection accuracy compared to the CNN-based sequence classification and hidden Markov model (HMM) based approaches in detecting stealthy malware.

I. Introduction

The exponential growth in the embedded computing technology utilized in the Internet-of-Things (IoT) devices, has made the system's security an indispensable issue. Among multiple security threats, malware is a vital threat due to relatively less intricacy to design, craft and propagate into the device(s) [1]. Malicious software, generally known as 'malware' is a software program or an application developed by an attacker to gain unintended access to the computing device(s) in order to perform unauthorized accesses as well as malicious activities such as stealing data, accessing sensitive information such as credentials, and manipulating the stored information without user's consent. In addition to the level of threat posed by malware, a tremendous increase in the malware samples is observed in the recent years [2]. This calls for a comprehensive and robust malware detection technique to mitigate the malware threat.

Traditional and primitive software-based malware detection techniques such as signature-based and semantics-based anomaly detection techniques [3], [4] exist for more than two decades, though effective, induces remarkable computational and processing overheads and is inefficient to detect unseen threats [5]. To overcome the limitations of the software-based malware detection approaches, the work in [6] proposed using the microarchitectural event traces captured through on-chip hardware performance counter (HPC) registers. Despite the better performance compared to the software-based approaches, HPC-based approach fails to effectively detect stealthy malware[1] [7]. We perform a case study to determine the efficiency of HPC-based method using the data obtained through embedded 4 on-chip HPC registers in detecting malware (More details of HPC-based approach is presented in Section IV with experimental setup in Section V).

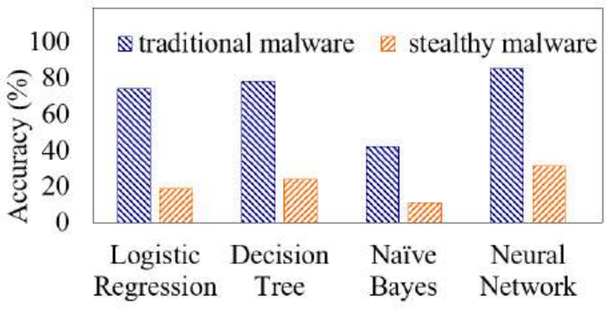


Fig. 1: Accuracy of HPC-based malware detection

Figure 1 illustrates the performance (accuracy) of HPC-based approach when using different machine learning (ML) models for detecting traditional malware and stealthy malware samples. As seen from Figure 1, HPC-based technique equipped with neural network classifier outperforms in detecting traditional malware samples with an accuracy of 85% followed by decision tree (78%) and logistic regression (74%). However, one can observe that this trend or performance does not hold true when the malware is obfuscated through

[1]Stealthy malware is a malware which is embedded inside a benign application through sophisticated malware obfuscation techniques, thereby making it complex to detect with traditional approaches as well as HPC-based approach and image processing approaches.

techniques such as code integration, code transposition, metamorphism, and polymorphism [8]. A significant drop in the performance (accuracy) (31% for neural network, 24% for decision tree and 19% for logistic regression) can be observed in detecting the stealthy malware when solely using the principle of HPC-based technique [9]. The efficiency of HPC-based method is questionable with advance code obfuscation techniques [10] which are used to embed malware into a benign application and create a stealthy malware. As the stealthy malware dissolves itself into the benign code and reunites dynamically at runtime to launch the malicious behavior, just leveraging globalized features for detecting stealthy malware is not sufficient [11]. This impels us to learn and extract the localized features for an efficient detection of stealthy malware.

Albeit the progress achieved by existing works in malware detection through HPC-based mechanism [4] or computer vision techniques [12], these works are confined mostly to detecting traditional malware. To overcome the shortcomings of existing works and address the aforementioned challenges, in this work we introduce a novel hybrid approach of detecting malware and stealthy malware, despite the advanced crafting techniques, with high efficiency. The major contributions of this work to achieve such high performance malware detection can be outlined in three-fold manner as follows:

- Our proposed traditional and stealthy malware detection technique uses a HPC-based approach as well as localized feature-based approach. In the HPC-based solution, the HPC traces of a given application are collected during runtime and is validated through a traditional ML classifier for malware detection and classification.
- In the localized feature-based method, the application binaries are translated into image binaries from which local features are extracted and processed through long short-term memory (LSTM) recurrent neural network (RNN) for malware detection and classification.
- Depending on the confidence of the classification for a given application, the class proposed by the two approaches is considered as the final output.

We have evaluated the proposed two-prong approach with novel localized feature extraction technique on over 2500 malware samples, 1000 stealthy malware samples and 500 benign samples. The accuracy which we achieved on malware samples was 94% and accuracy for stealthy malware was nearly 90% with a F1-score of 92% and recall score of 91%.

The rest of the paper is organized as follows: we present the motivation for our work and formulate the problem of stealthy malware in Section II. In Section III, we present existing works and outline the differences. We present the proposed malware detection technique with emphasis on localized feature-based extraction in Section IV, where we start with overview of the methodology followed by HPC-based approach, localized feature extraction based approach and an algorithmic description. The experimental evaluation of the proposed technique is illustrated in Section V with conclusions drawn in Section VI.

II. Motivation and Problem Formulation

A. Motivation

Here, we discuss the the key findings which motivated and impelled us to propose the hybrid two-pronged malware detection technique. Figure 2a, Figure 2b and Figure 2c visualizes the features for benign, stealthy malware and traditional malware applications respectively in the form of heatmaps. The y-axis (imageID) represents the sequence ID of the various patterns extracted from the a gray-scale image of an application's executable and the x-axis (uniqueID) represents the ID of the unique patterns that were obtained from the malware and benign application dataset. The unique patterns comprises of all the unique patterns that are found in the experiment wide range of malware and benign applications. The intensity of the each pixel in the heatmap is the cosine similarity percentage between the individual pattern in the file for which the heatmap is generated and the unique patterns belonging to the class. High intensity indicates ample presence of the sequence in the particular application class, which shows a close match between the various sequences.

We draw following observations from the plotted heatmaps: (1) In Figure 2a, which is a heatmap for benign applications, one can observe high intensity (dark) regions from unique ID 183 to 671. However, the same region appear with less intensity for the same unique IDs in the case of malware, as in Figure 2c. Moreover, across the malware heatmaps (Figure 2b and Figure 2c), no such intense regions can be observed. (2) In the heatmap for stealthy applications (Figure 2b), one can observe that for a given uniqueID, the heatmap across ImageIDs are not uniform i.e. one can observe equidistant horizontal light intensity regions and similar faded horizontal regions, indicating uneven pattern occurrence in stealthy malware; and (3) the intensity of patterns are spread across the uniqueID for stealthy malware, whereas localized in the case of traditional malware, as observed in Figure 2c and Figure 2b. Altogether, it makes stealthy malware harder to detect. This simple experiment states that the difference in heatmaps clearly illustrates that the localized features which we are extracting from gray-scale images play an important role to distinguish between stealthy malware, malware and benign applications.

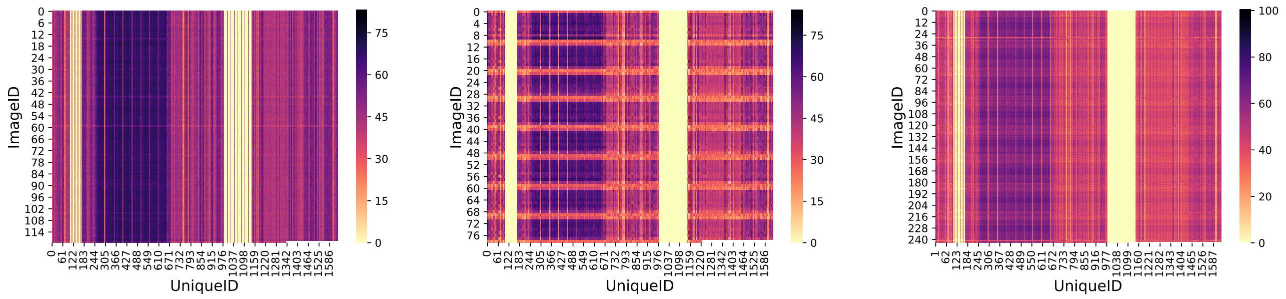
B. Problem Formulation

Before introducing the problem, we present the notations used in this work. We represent the corpus of dataset as D that encompasses of malware samples M, benign samples B and stealthy malware samples $\mathfrak{D}_\mathfrak{s}$ i.e.,

$$D = M \cup B \cup \mathfrak{D}_\mathfrak{s} \tag{1}$$

Malware dataset M is organized in the following manner,

$$\begin{aligned} M &= \{M^1, M^2, ..., M^j\} \\ M^j_n &= \{M^j_1, M^j_2, ..., M^j_n\} \end{aligned} \tag{2}$$



(a) (b) (c)

Fig. 2: Heatmap for (a) benign applications; (b) stealthy malware; and (c) malware applications

where M^j represents the j^{th} malware sample with M^j_n representing that there are n number of patterns in a given j^{th} malware sample.

In similar way, benign samples are organized as

$$\begin{aligned} B &= \{B^1, B^2, ..., B^k\} \\ B^k_q &= \{B^k_1, B^k_2, ..., B^k_q\} \end{aligned} \tag{3}$$

where B^k represents the k^{th} benign sample with B^k_q representing that there are q number of patterns in a given k^{th} benign sample.

The patterns for a given application are obtained and aligned from the spatial distribution of samples of their gray-scale images respectively.

Similarly, the stealthy malware created through code obfuscation techniques [10] are denoted by $\mathfrak{D}_{\mathfrak{s}}$ such that

$$\mathfrak{D}^{\mathfrak{i}}_{\mathfrak{s}} = M^j \cap B^k \neq \{\phi\} \tag{4}$$

where $\mathfrak{D}^{\mathfrak{i}}_{\mathfrak{s}}$ is a i^{th} stealthy malware sample which encompasses j^{th} malware sample and k^{th} benign sample. Every stealthy malware sample $\mathfrak{D}^{\mathfrak{i}}_{\mathfrak{s}}$ is structured in the following manner,

$$\mathfrak{D}^{\mathfrak{i}}_{\mathfrak{s}} = \{M^j_1, B^k_5, M^j_4, B^k_2, B^k_{10}, ..., M^j_{15}, ..., M^j_n, B^k_q\} \tag{5}$$

Considering the emerging techniques in hiding the malware (through polymorphism, metamorphism, dead-code insertion, code relocation and register reassignment) [8] from the detection techniques, it is non-trivial to devise a methodology that is capable of detecting traditional as well as stealthy malware will similar efficiency. Thus, we define the problem as follows:

Despite dispersing or hiding the malicious patterns to circumvent malware detection, one needs to devise a technique that is able to characterize and efficiently detect the malware through the sub-patterns that can distinguish stealthy malware and benign applications.

This is mathematically defined as

$$f : D \Rightarrow Z \tag{6}$$

where f is a mapping function that maps the incoming application (benign, traditional malware or stealthy malware) to its corresponding class efficiently, D is combination of malware, benign and stealthy malware dataset and Z is the class of the application. Our proposed approach extracts the localized features of the incoming application, thus making it feasible to detect the stealthy malware despite deploying malware obfuscating techniques.

III. Related Work

Here, we outline the existing works on traditional and stealthy malware detection.

Many researchers have leveraged architectural and application features for malware analysis and detection [13] in the past. In [14] Bilar et al. used the difference of opcodes between known malware and benign as a key to predict malware. Similarly, other methods utilize frequency of opcodes [15] and sequences of opcodes [16] to model the malicious behavior. A graphical technique was proposed by Runwal et al. in [17] to find the similarity of the opcode sequence. However, these proposed techniques requires considerable amount of work to model each program based on instructions. Since the code size increases day by day, modeling program based on opcodes becomes a time-consuming task and also increases the memory requirement. Considering that each instruction of the program has to be traced, there will be significant amount of performance overhead on the system. Demme et al. [6] proposed the use of hardware performance counter (HPC) to monitor the lower level micro-architectural parameters such as branch-misses, instruction per cycle, cache miss rate. In [18] NumChecker detects malicious modifications to a kernel function (system call) by inspecting the hardware events including total instructions, branches, returns and floating-point operations. However, all the above methods use low-level hardware features to model the malicious behavior and also have a low performance overhead but they are limited by a high false positive rate (nearly 10%) [6]. In comparison, our proposed technique not only uses HPCs, but also extracts localized features and has lower false positive rate, which is more significant in stealthy malware detection.

To overcome the shortcomings of these malware detection techniques, the features extracted from HPCs in HPC-based technique were given to specialized machine learning classifier in [4] at runtime. The main purpose of HPC is to analyze and tune architectural level performance of running applications [19]. In addition to HPC-based techniques, researchers explored computer vision-based application analysis for malware detection. Natraj et al. [11] classifies malware images by using k-nearest neighbors. This approach is not capable of handling code obfuscation techniques like code relocation and mutation, and may not be feasible in resource constrained environments because to extract the image texture as features for classification and the system needs pre-processing. Artificial neural network (ANN) for malware classification is used in [20] and [21] which is computationally costly as there are multiple fully connected layers in (ANN) for malware classification. In [22] authors convert the binary executable of malware samples to a gray-scale image and utilize a single-channel lightweight convolutional neural network (CNN) to efficiently detect IoT malware. David and Netanyahu apply a deep belief network (DBN) [23] to log files which are generated directly by a sandbox environment, which not only captures API calls but also other events from the running malware as a sequential representation. RNN is used widely in sequence classification [12], [24] and pattern-based feature selection and classification [25], [26]. The challenge of applying pattern-based feature selection on symbolic sequences is that the features must satisfy the following criteria: (1) frequent in at least one class (2) distinctive in at least one class (3) sequence should have a uniform length and (4) not redundant. Sequence classification with sequence of symbols is a strong classification task. We attain all the aforementioned criteria in our proposed methodology. First criteria is accomplished from equation (4). It clearly states that stealthy malware has atleast one pattern from a malware or benign class. To fulfill criteria 2 and 4, we are using cosine similarity to find distinct pattern based on spatial distribution and we discard repeating pattern from one class or more classes. To assure that every sequence has uniform length, we are using zero padding mechanism.

To the best of our knowledge, the existing works on malware detection lacks strategies to efficiently detect stealthy malware and traditional malware with same efficiency and requires plethora of resources, making them computationally expensive to adopt. In contrast, through the proposed two-prong approach in our work, the HPC-based method is effective in detecting malware, whereas through the localized feature based processing we can detect the obfuscated malware.

Our proposed technique for detecting stealthy malware is faster as we target to capture localized characteristics of applications and convert them to sequences that are fed to a ML sequence classifier which is trained to learn the sequences as features for several applications. This makes our proposed technique resource efficient as no additional hardware or memory is needed.

IV. Proposed Malware Detection Methodology

A. *Overview of the Proposed Methodology*

First, we present the overview of the proposed two-prong methodology depicted in in Figure 3 for an efficient malware detection, followed by in-depth details. The incoming application (malware or benign or stealthy malware) is fed to both, HPC-based method and localized feature extraction based computer vision approach simultaneously as shown in Figure 3. In the HPC-based approach, the prominent HPCs information is collected during runtime, which is then fed to ML classifier for malware detection. The prominent HPCs that are needed, are determined offline by obtaining all feasible HPC values and feeding to principal component analysis (PCA) for feature reduction. While, the HPC-based technique performs the dynamic analysis on incoming file, the localized feature based approach is a static approach that utilizes computer vision-based processing for malware detection. In this approach, the incoming binary file is converted to a gray-scale image from which the patterns are obtained. These patterns are labelled and compared with the stored patterns of stealthy and traditional malware by employing a RNN-LSTM. Depending on the classification confidence from both the techniques, the class predicted by the approach with higher confidence is considered as the output class for input application. We describe the details of individual approaches below.

B. *HPC-based Detection*

In the HPC-based detection technique, we require the microarchitectural event traces captured through HPCs for malware detection. One of the challenges is that there are a limited number of available on-chip HPCs that one can extract at a given time-instance. However, executing an application generates few tens of microarchitectural events. Thus, to perform real-time malware detection, one needs to determine the non-trivial microarchitectural events that could be captured through the limited number of HPCs and yield high detection performance. To achieve this, we use principal component analysis (PCA) for feature/event reduction on all the microarchitectural event traces captured offline by iteratively executing the application. Based on the PCA, we determine the most prominent events and monitor them during runtime. The ranking of the events is determined as follows:

$$\rho_i = \frac{cov(App_i, Z_i)}{\sqrt{var(App_i) \times var(Z_i)}} \tag{7}$$

where ρ_i is pearson correlation coefficient of any i^{th} application. App_i is any i^{th} incoming application. Z_i is an output data contains different classes, backdoor, rootkit, trojan, virus and worm in our case. $cov(App_i, Z_i)$ measures covariance between input and output. $var(App_i)$ and $var(Z_i)$ measure variance of both input and output data respectively. Based on the ranking, we can select most prominent HPCs and monitor them during runtime for efficient malware detection. These reduced features collected at runtime are provided as input to ML classifiers which determine the malware class label ($\widehat{Y}$ $\Rightarrow$ Backdoor, Rootkit, Trojan, Virus and Worm) with higher confidence (α). This HPC-based malware detection technique is fast, robust and accurate in detecting and classifying the malware but it has overhead in terms of area, power and latency. Despite the benefits, as seen in Figure 1, this approach does not yield higher performance on stealthy malware due to contamination of HPC when malware is embedded into the benign application. To address this critical issue, a computer vision-based approach is adopted in parallel.

C. Localized Feature Extraction based Detection

In the computer vision-based detection technique, as shown in Figure 3, the application binary is converted into a gray-scale image for localized feature extraction. A raster scanning is performed on the converted binary images to find the image patterns. Each pattern is of 32×32 block size. We utilize a cosine similarity to distinguish between multiple patterns i.e., if the cosine similarity of two patterns is higher than threshold (0.75 in this work based on conducted experiments), they are considered to be same. When more than one matched patterns are found, the one with the highest cosine similarity is considered. It needs to be noted that the pattern matching for an incoming binary is performed with the patterns in the database (created through similar process during training phase, but offline). Once the image patterns are recognized for a given binary file, the whole image binary is converted into a sequence of patterns (Each pattern is provided with a unique ID). This sequence is fed to a long short-term memory (LSTM) recurrent neural network (RNN). RNN can be fed with the sequences of same length. However, the size of sequence can vary according to the size of gray-scale image of the converted binary file. To address this challenge, we perform the padding of zeros to sequence in order to make its length uniform. As the pattern or sub-pattern of a malware cannot alter despite embedding the malware to launch malicious payload, through this technique the stealthy malware can be detected with higher performance (around 90% accuracy).

We generate a training dataset as shown in equation (2) and equation (3) which consists of several sequences for variety of classes of malware (backdoor, rootkit, trojan, virus and worm) and benign applications. We also create a testing dataset as shown in equation (5) which contains stealthy malware samples by embedding malware into benign applications through code obfuscation technique [10]. All the sequences for this application are generated using the discussed method.

Learning of RNN for the patterns happens as follows. Let u_t and h_t denote the input and state vectors, respectively, at time instance t. Let W_{in}, W_{rec}, b, W_{out}, b_{out} be the input to hidden layer weight matrix, recurrent weight matrix, bias, output weight matrix and output bias respectively. Let ω and ϵ be the activation function of the hidden layer and output layer respectively. In our proposed work, $\tanh$ is used for hidden layer and softmax is used for the output. The recurrent models are then described by the following equations:

$$h_t = \omega \times (W_{in} \times u_t + W_{rec} \times h_{t-1} + b) \tag{8}$$

$$\widehat{X_t} = \epsilon \times (W_{out} \times h_t + b_{out}) \tag{9}$$

This RNN-model is finally used to classify the incoming stealthy malware binary based on equation (8) and equation (9) and predicts the corresponding class label $\widehat{X_t}$. The rationale to utilize RNN is to exploit the temporal as well as spatial dependencies that attackers utilize to craft stealthy malware.

For the given input application, we have 2 labels (same or different) predicted through HPC-based and computer vision-based approaches. In our work, we consider the confidence of the prediction from both the approaches and the label from predictor with confidence higher than threshold (α = 75%, which is determined through trial-and-error method) is considered to be the associated class for the input application. For the dataset with 1000 stealthy malware samples and 2500 traditional malware samples, we did not encounter a scenario where both the detection techniques have high confidence and contrasting output classes.

D. Summary of Proposed Methodology

Algorithm 1 Pseudo-code describing proposed methodology for malware detection.

```
Require: Application $App_i \in \{M, B, \mathfrak{D}_\mathfrak{s}\}$
Ensure: $Z_{App_i}$ {$Z_{App_i}$ is class for $App_i$}
1: pthread_create $hpc\_function()$
2:   $hpc\_function(App_i)$:
3:     $f_e = f_1, f_2, ....., f_n$ { feature extracted using $perf$}
4:     $f_r = PCA(f_e)$ { feature reduction using PCA}
5:     $\alpha_1$, $\widehat{Y_t} = ML(f_r)$ {$ML(.)$ is a Machine learning model built offline}
6:   end
7: pthread_create $localized\_function()$
8:   $localized\_function(App_i)$:
9:     $App_i \Rightarrow$ gray-scale image
10:    $P_{App_i} \Leftarrow (P_0, P_1, ..., P_n)$ {Obtain patterns for gray-scale image}
11:    Compare $P_{App_i}$ with patterns in database(DB)
12:    $S_{App_i} \Leftarrow (a_1, b_1, ..., b_n, z_n)$ {Obtain sequence for gray-scale image}
13:    $\alpha_2$, $\widehat{X_t} = RNN(S_{App_i})$ {$RNN(.)$ is a RNN model built offline}
14:  end
15: if $\alpha_1 \geq \alpha$ and $\alpha_1 > \alpha_2$
16:     $Z_{App_i} \Leftarrow \widehat{Y_t}$
17: else
18:     $Z_{App_i} \Leftarrow \widehat{X_t}$
19: end
```

In Algorithm 1, we have provided the pseudocode for our proposed malware detection methodology. Let us consider the incoming application as App_i which may belong to either a malware, benign or stealthy malware. As soon as our architecture encounters the incoming application, it spawns two threads using pthread_create namely $hpc\_function()$ and $localized\_function()$.

Thread $hpc\_function(App_i)$ performs the dynamic analysis on the incoming application App_i based on HPC-based detection method. Initially we extract the features f_e of App_i using $perf$ tool. Due to limited number of HPCs, we utilize PCA to perform feature reduction to obtain reduced features f_r, which is performed offline. These reduced features f_r are monitored during runtime for the incoming application and fed

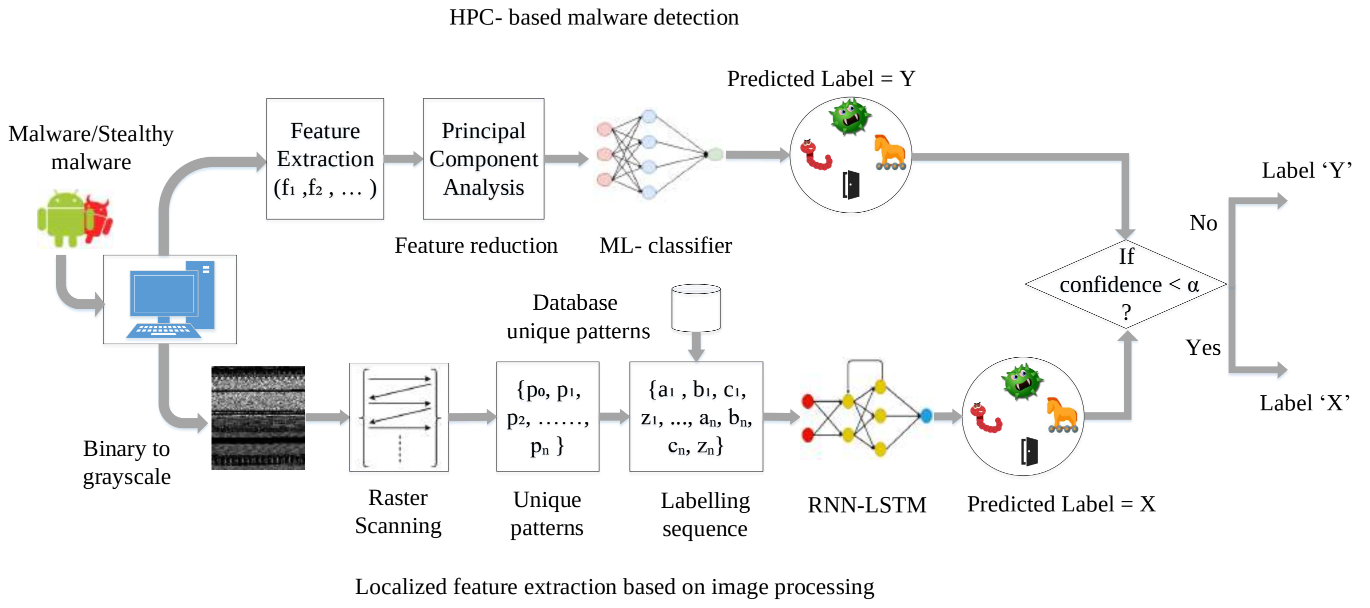


Fig. 3: Proposed hybrid approach for detecting stealthy malware

to ML classifier to determine the α_1 and $\widehat{Y_t}$ i.e., confidence and label of the application respectively, as shown in Line 1-5 in Algorithm 1.

On the other hand, thread $localized\_function(App_i)$ performs localized feature extraction scheme simultaneously. Here, the incoming application App_i is converted into a gray-scale image. Raster scanning is performed on this gray-scale image to obtain sequence of patterns P_{App_i} for the gray-scale image. These obtained sequence of patterns are compared with the patterns available in database (formed through similar procedure but carried out offline). We created this database (DB) with 2500 malware samples, 1000 stealthy malware samples and 500 benign samples. After comparison, we assign a sequence of labels S_{App_i} to these patterns. Lastly, this sequence is fetched as an input to a pre-trained RNN to classify incoming App_i and gives most relevant output α_2 and $\widehat{X_t}$, which are confidence and label of class respectively, as shown in Line 7-13 in Algorithm 1.

Once we obtain the outputs from both approaches, we compare confidence of both the detection techniques α_1 and α_2 with the threshold confidence α. Final Classification output label of the proposed malware detection methodology, Z_{App_i} is $\widehat{Y_t}$ if $\alpha_1 \geq \alpha$ and $\alpha_1 > \alpha_2$ otherwise Z_{App_i} is $\widehat{X_t}$.

V. Experimental Results

In this section we will discuss about (1) Experimental setup used for evaluating our work, (2) Visualization of stealthy malware, (3) Performance of malware detection and (4) Key findings.

A. *Experimental Setup*

The proposed methodology is implemented on an Intel core i7-8750H CPU with 16GB RAM. We have obtained malware applications from VirusTotal [27] with 2500 malware samples that encompasses of 5 malware classes: backdoor, rootkit, trojan, virus and worm. Further, we utilized benign applications such as documents (.pdf, .txt, .docx) inside which the binaries of above mentioned malware classes are integrated through code obfuscation (code relocation [8]) process to create 1000 stealthy malware samples. We selected a benign file and randomly embed a malware sample using code relocation technique. Thus, malware code is not only embedded using code relocation but to increase the stealthiness we randomly placed the code into a benign file. We created around 600 such complex stealthy malware. To test robustness of our technique we also created around 400 stealthy malware by embedding malware into benign file using code transposition technique [10] i.e. reordering the sequence of instructions of an original code without having any impact on its behavior. The HPC-based mechanism is a single layer neural network with 10 neurons in hidden layer. The RNN model has 1 dimensional dropout layer with threshold of 0.7, LSTM layer with 64 neurons and 0.7 recurrent dropout followed by a dense layer with softmax activation. It uses ADAM optimizer and loss is calculated based on the categorical cross entropy.

B. *Visualization of Stealthy Malware*

Figure 4 visualizes stealthy malware using the most prominent four features determined using the aforementioned feature reduction (PCA). Among all the captured HPC events, branch instructions, branch loads branch-instructions and LLC-loads seems to be prominent, as the malware impacts the branch and last-level cache (LLC) to inject the payload. We plot the impact of one feature on other and its correlation to the malware/benign class here. and resulted in significant As seen from Figure 4, trojan stealthy malware class is easily distinguishable from benign class for branch-instructions vs branch-loads graph, whereas, in branch-instructions vs LLC-loads graph, trojan stealthy malware is easily distinguishable from other classes of malware. Hence, from Figure 4 we can conclude that HPC-based detection technique could only detect one class of stealthy malware (trojan). So, utilizing a localized feature extraction based detection technique in parallel makes our proposed solution more robust.

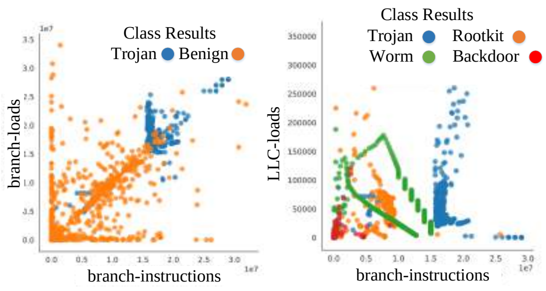


Fig. 4: Visualization of stealthy malware

C. *Performance of Malware Detection*

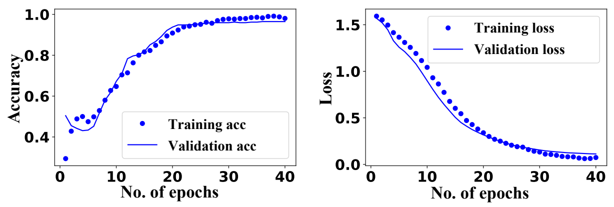


Fig. 5: Loss and accuracy of the proposed malware detection model

Figure 5 depicts the loss and accuracy for the overall malware dataset $D = M \cup B \cup \mathfrak{D}_{\mathfrak{s}}$. As seen from Figure 5, with the increase in epochs the accuracy increases and saturates at 40 epochs, hence we considered the model trained with 40 epochs for final evaluation. For the normal malware i.e., malware spawned as separate thread, an accuracy of nearly 90% is achieved with HPC-based malware detection. However, for stealthy malware, the HPC-based malware detection accuracy is shattered to 54% on an average (These individual results are not plotted for the purpose of brevity). However, with the proposed hybrid approach, an accuracy of 94% is achieved, with a loss of 0.14 as shown in Figure 5, respectively despite of code obfuscation.

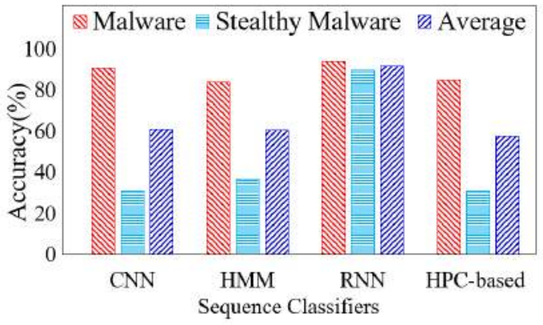


Fig. 6: Performance Comparison of Sequence Classifiers

Performance comparison of the sequence classifiers can be illustrated from Figure 6, where we have compared our technique with other traditional sequence classification mechanisms. We observe that for a normal malware application, RNN sequence classifier based on our localized feature extraction method, classifies malware with the highest accuracy of 94% accuracy as compared to some traditional sequence classifier which classifies the same malware application with 91% (CNN), 84% (HMM) and 85% (HPC-based) of accuracy. While the traditional sequence classification approaches failed badly with a significant drop in accuracy while testing stealthy malware 31% (CNN), 37% (HMM) and 31% (HPC-based); our localized feature extraction based method achieved an accuracy of nearly 90% while testing stealthy malware on RNN. The most important observation was the accuracy was definite even while testing stealthy malware with randomized code obfuscation, code relocation and polymorphism techniques used for creating stealthy malware [8], [10].

TABLE I: Comparison of Precision score

Classifier	Backdoor	Rootkit	Trojan	Virus	Worm	Stealthy
CNN	0.88					
HMM	0.81					
RNN (proposed)	**0.93**					
HPC-based	0.80					

TABLE II: Comparison of Recall score

Classifier	Backdoor	Rootkit	Trojan	Virus	Worm	Stealthy
CNN	0.9	0.93	0.81	0.85	0.82	0.4
HMM	0.8	0.85	0.76	0.82	0.7	0.45
RNN (proposed)	**1**	**0.98**	**0.9**	**1**	**1**	**0.91**
HPC-based	0.8	0.76	0.79	0.71	0.7	0.62

TABLE III: Comparison of F-1 score

Classifier	Backdoor	Rootkit	Trojan	Virus	Worm	Stealthy
CNN	0.87	0.9	0.8	0.83	0.78	0.37
HMM	0.72	0.79	0.8	0.83	0.67	0.4
RNN (proposed)	**0.94**	**0.97**	**0.94**	**0.97**	**0.93**	**0.92**
HPC-based	0.83	0.72	0.77	0.68	0.66	0.57

In addition to accuracy, we evaluate and compare other performance metrics for malware detection. Precision score of 0.93 is achieved with the proposed methodology with an average F-1 score and recall score of 0.94 and 0.96 respectively. As seen from Table I, the precision obtained with proposed methodology is 11% higher on an average compared to other approaches, including HPC-based approach. Similarly, from Table II and Table III, we conclude that the recall score and F-1 score attained with proposed methodology is approximately 24% higher and 23% higher respectively, on an average compared to other approaches.

We plot the ROC-curve for malware detection to evaluate the robustness, which is depicted in Figure 7. It is evident that with the proposed proposed malware detection, a higher area under curve (AUC) of 0.94 is obtained, which indicates a higher robustness. Considering the evaluation evidences, we can substantiate that our proposed hybrid malware detection technique successfully outperforms other malware detecting mechanisms in detecting traditional and stealthy malware.

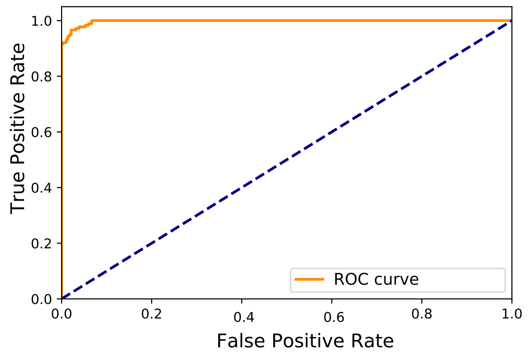


Fig. 7: ROC-curve

D. Key Findings

Some of the key findings of our proposed hybrid malware detection technique are as follows:

- *We perform raster scanning on a gray-scale image with a 32 $\times$ 32 scanning block, which help us to capture each and every pattern from that gray-scale image. This also ensures that there is minimal loss of information and localized features.*
- *To find spatial similarity in patterns, we use cosine similarity where all the patterns with cosine similarity greater than 0.8 are considered similar.*
- *Further, we generate a sequence of labels from these patterns for every gray-scale image, where every unique pattern will have a unique label and pattern with same cosine similarity or cosine similarity greater than 0.8 will be assigned the same label.*
- *Finally, a RNN model is trained with data comprising of sequences for every benign and malware application.*
- *In order to detect the obfuscated piece of malware, the extracted localized features play a vital role such that when a incoming application is a stealthy malware, our proposed detection technique successfully detects and classifies it despite of a malware application being randomly obfuscated into a benign application.*

VI. Conclusion

In this work, we propose a hybrid approach of utilizing architectural (trace) as well as code properties obtained through HPCs and localized features extraction, respectively for stealthy malware detection. In the HPC-based approach, we determine the most prominent HPCs for malware detection and feed them to ML classifier for malware detection. In parallel, we provide the incoming application to the devised image processing technique to convert application binary to a gray-scale image and extract distinct patterns over spatial distribution using. Sequence labelling was further performed to these distinct patterns. As discussed in the paper, our proposed methodology successfully addresses all the sequence classification criteria which makes sequence classification a difficult task. For sequence classification, we utilize a RNN to extract and process the localized features to attain the highest average accuracy of 90% over stealthy malware and 94% over traditional malware application. Thus, we conclude that our proposed methodology is robust and fast in detecting stealthy malware and traditional malware. We will study hardware implementation and analysis of this technique as our future research work.

References

[1] Z. Ling and et al., “Iot security: An end-to-end view and case study,” *CoRR*, 2018.

[2] M. Labs, “Inforgraphic: Mcafee labs threats report,” 2018.

[3] Jacob and et al., “Behavioral detection of malware: From a survey towards an established taxonomy,” *Journal in Computer Virology*, vol. 4, pp. 251–266, 08 2008.

[4] N. Patel, A. Sasan, and H. Homayoun, “Analyzing hardware based malware detectors,” in *Design Automation Conf.*, 2017.

[5] Q. Chen and R. A. Bridges, “Automated behavioral analysis of malware: A case study of wannacry ransomware,” in *Int. Conf. on Machine Learning and Applications (ICMLA)*, 2017.

[6] J. Demme and et al., “On the feasibility of online malware detection with performance counters,” in *ISCA’13*, 2013.

[7] S. J. Stolfo and et al., “Towards stealthy malware detection,” 2007.

[8] I. You and K. Yim, “Malware obfuscation techniques: A brief survey,” in *Int. Conf. on Broadband, Wireless Computing, Communication and Applications*, 2010.

[9] H. Sayadi and et al., “Ensemble learning for effective run-time hardware-based malware detection: A comprehensive analysis and classification,” *Design Automation Conf.*, 2018.

[10] Bashari and et al., “Camouflage in malware: From encryption to metamorphism,” *International Journal of Computer Science And Network Security (IJCSNS)*, pp. 74–83, 2012.

[11] L. Nataraj, S. Karthikeyan, G. Jacob, and B. S. Manjunath, “Malware images: Visualization and automatic classification,” in *Int. Symposium on Visualization for Cyber Security*, 2011.

[12] R. Pascanu, J. W. Stokes, H. Sanossian, M. Marinescu, and A. Thomas, “Malware classification with recurrent networks,” in *Int. Conf. on Acoustics, Speech and Signal Processing (ICASSP)*, 2015.

[13] Brasser and et al., “Advances and throwbacks in hardware-assisted security: Special session,” in *International Conference on Compilers, Architecture and Synthesis for Embedded Systems*, ser. CASES ’18, 2018.

[14] D. Bilar, “Opcodes as predictor for malware,” *IJESDF*, 2007.

[15] I. Santos and et al., “Idea: Opcode-sequence-based malware detection,” in *ESSoS*, 2010.

[16] ——, “Opcode sequences as representation of executables for data-mining-based unknown malware detection,” 2013.

[17] N. Runwal, R. M. Low, and M. Stamp, “Opcode graph similarity and metamorphic detection,” *Journal in Computer Virology*, 2012.

[18] X. Wang and R. Karri, “Numchecker: Detecting kernel control-flow modifying rootkits by using hardware performance counters,” in *Design Automation Conference (DAC)*, 2013.

[19] H. Sayadi and et al., “2smart: A two-stage machine learning-based approach for run-time specialized hardware-assisted malware detection,” in *Design, Automation Test in Europe Conf. Exhibition*, 2019.

[20] A. Makandar and A. Patrot, “Malware analysis and classification using artificial neural network,” in *Int. Conf. on Trends in Automation, Communications and Computing Technology (I-TACT-15)*, 2015.

[21] S. M. P. Dinakarrao and et al., “Adversarial attack on microarchitectural events based malware detectors,” in *Design Automation Conf.*, 2019.

[22] J. Su, V. Danilo Vasconcellos, S. Prasad, S. Daniele, Y. Feng, and K. Sakurai, “Lightweight classification of iot malware based on image recognition,” in *Computer Software and Applications Conf.*, 2018.

[23] O. E. David and N. S. Netanyahu, “Deepsign: Deep learning for automatic malware signature generation and classification,” in *Int. Joint Conference on Neural Networks (IJCNN)*, 2015.

[24] A. Graves, S. Fernández, F. Gomez, and J. Schmidhuber, “Connectionist temporal classification: Labelling unsegmented sequence data with recurrent neural networks,” in *Int. Conf. on Machine Learning*, 2006.

[25] J. D. Lafferty, A. McCallum, and F. C. N. Pereira, “Conditional random fields: Probabilistic models for segmenting and labeling sequence data,” in *Int. Conf. on Machine Learning*, 2001.

[26] N. Lesh and et al., “Mining features for sequence classification,” in *ACM SIGKDD Int. Conf. on Knowledge Discovery and Data Mining*, 1999.

[27] G. Sood, “virustotal: R client for the virustotal api,” 2017.