IEEE Access
Multidisciplinary | Rapid Review | Open Access Journal

Received May 2, 2020, accepted May 28, 2020, date of publication June 1, 2020, date of current version June 11, 2020.

Digital Object Identifier 10.1109/ACCESS.2020.2999320

MSIC: Malware Spectrogram Image Classification

AHMAD AZAB, (Member, IEEE), AND MAHMOUD KHASAWNEH, (Member, IEEE)
College of Engineering and Technology, American University of the Middle East, Egaila 15453, Kuwait

Corresponding author: Ahmad Azab (ahmad.azab@aum.edu.kw)

This work was supported by the College of Engineering, American University of the Middle East, Kuwait.

ABSTRACT The heavy reliance on digital technology, by individuals and organizations, has reshaped the traditional economy into a digital economy. In response, cybercriminals' attention has shifted dramatically from showing off skills and conducting individual attacks into high sophisticated attacks with financial gain as the goal. This, inevitably, poses a challenge to the cybersecurity community as they strive to find solutions to preserve the confidentiality, availability and integrity of the individual users' and corporates' private data and services. Cybercriminals mainly deploy malware to achieve their goals, which could be in the form of ransomware, botnets, etc. The use of encryption, packing and polymorphism techniques makes it harder to detect the malware files, especially when these are created in great numbers every day. In this paper, a novel framework, named Malware Spectrogram Image Classification (MSIC), is proposed. It employs spectrogram images in conjunction with the convolution neural network to classify a malware file to its corresponding family and to differentiate it from a benign file. Further, this research shares with the research community two privately collected labeled malicious and benign datasets. The evaluation of MSIC showed its effectiveness to be 91.6% F-measure and 92.8% accuracy in classifying malware files to their corresponding families, in comparison to, respectively, 90.6% and 92.3% results produced by the grayscale image classification approach. Likewise, in classifying files as malicious or benign, MSIC scored 96% F-measure and accuracy results compared to 95.5% with the grayscale solution. Also, MSIC required less computational time in converting and resizing the files than the grayscale framework.

INDEX TERMS CNN, cybersecurity, deep learning, malware, spectrogram.

I. INTRODUCTION

The rapid evolution of digital technology in various areas has led to its integration in such personal and corporate activities as personal banking and e-commerce. As a result, the traditional economy has transformed into an Internet economy. Google and Temasek research program reported that Southeast Asia's Internet economy reached US$100 billion [1]. Increasing reliance on digital technology and the recent transformation of the economy have caught the attention of cybercriminals and they have changed their intentions from individual attacks into well-organized attacks with financial gain as the objective. The total cost of cybercrime for each company increased from US$11.7 million in 2017 to a new high of US$13.0 million—a rise of 12% [2].

Malicious Software (Malware) is the primary tool that attackers use to gain their end. The form of the malware varies according to the purpose of the attack. It could be adware, worm, ransomware, spyware or botnets. A botnet is the main platform used nowadays [3]. McAfee's security report found that, in the last quarter of 2018, more than 60 million new malware samples have been identified [4]. The creation of such a huge number of malicious executables is mainly caused by the advancement in malware implementation that utilizes packing, polymorphic and metamorphic techniques [5]. In addition, a large quantity of newly released malware families are not coded from the scratch, whereas they were rewritten to be a variant of a previous released family [6]. Symantec found that the number of the new malware families has dropped as the cybercriminals are modifying existing ones [7]. For example, Fox IT found that Tilon malware is linked to Spyeye and Zeus malware family [8]. These findings are posing a clear evident of the importance in identifying malware files, classify them to their families and prevent them from being executed at the victim's device.

Motivated by this evident threat, information security society has proposed security measures to detect malware and prevent their attacks. Static based detection aims to find

The associate editor coordinating the review of this manuscript and approving it for publication was Mohamed Elhoseny.

a signature in the code by reverse engineering the executable and match it with an updatable database to find a match. Dynamic based detection aims to execute the malware and identify malicious behavior, without the need to reverse engineer the malware. Image based detection, which is a recent approach in the literature, aims to detect malware by converting it to an image and classifying the image as malicious or benign. The latter approach, unlike static and dynamic analyses, does not require domain knowledge for malware detection.

This research was motivated by the need to identify the huge number of malware files that cybercriminals create and classify them to their corresponding families. Our paper provides the following four contributions:

- We propose a novel framework named Malware Spectrogram Image Classification (MSIC) to distinguish malicious files from benign files and classifying the malware files to their corresponding families by utilizing voice spectrogram analysis in conjunction with deep learning algorithms.
- Evaluation analysis of MSIC is conducted for different deep learning parameters to classify malware files to their corresponding families and distinguish them from benign files. The results are compared to the grayscale classification framework that has been used by researchers in the literature.
- The research provides the information security community with a labeled malware dataset containing 11 malware families.
- The research provides the information security community with a labeled malicious-benign dataset that has been collected from the Internet at our private lab.

The rest of the paper is organized as follows. Section II reviews the various classification models that have been proposed by researchers in the literature. In section III, the proposed framework, MSIC, is introduced, along with a discussion of audio signals and Convolution Neural Network (CNN). Section IV introduces the experimental environment and the evaluation metrics used. Section V explains the malware dataset collection process and presents the results of the evaluation of MSIC and the grayscale frameworks for classifying the malware files to their corresponding families. Section VI describes the collected malicious-benign dataset, along with the experimental evaluation of both the MSIC and the grayscale frameworks for distinguishing the malicious and benign files. Section VII concludes the research and points to the directions of possible future work.

II. CLASSIFICATION MODELS

Researchers have been striving to frustrate the malicious intentions of cybercriminals. The following subsections shed light on the different techniques that have been proposed to distinguish malicious-benign files and to classify malware to their corresponding families.

A. STATIC ANALYSIS TECHNIQUE

Static analysis detection works without the need to execute the malware. The solution is to first reverse engineer the executable. Thereafter, a signature is extracted from the malware source code and compared with a database that is updated periodically. This solution usually integrates machine learning algorithms in the classifier.

The authors in [9] were among the first researchers to use machine learning in building static analysis classifier to detect malware based on features such as program headers, strings and byte sequence. Their work was improved upon by the use of n-grams of byte codes as features of the classifier [10]. The researcher in [11] utilized opcode sequence with support vector machine algorithm to identify malicious executables. The Application Programming Interface (API) sequence in the code was used in detecting malware and proved to be effective and faster as compared to assembly analysis [12]. MalConv classifier detects malicious executables using their raw bytes feature as input for a fully connected neural network [13].

Static analysis suffers from many shortcomings. The solution requires domain knowledge of the executable architecture to build the classifier. Further, the information like size of data structures or variables gets lost thereby complicating the malware code analysis [14]. In addition, static analysis performs poorly in detecting unknown malware. Although some static analysis techniques, such as byte n-gram, do not require domain-level knowledge, they suffer from low performance and high computational requirements [15].

B. DYNAMIC ANALYSIS TECHNIQUE

Unlike the static analysis, dynamic analysis addresses the behavior of an executable by executing it and identifies the features accordingly. Such features include registry changes, memory writes, API and system calls. Usually, the dynamic analysis takes place inside a virtual machine.

The work in [16] differentiated malicious binaries from benign binaries by monitoring API calls and applying n-gram technique. The authors in [17] extracted API calls using a 5-minute time window. Then, they utilized Recurrent Neural Network (RNN) to build the classifier and CNN to evaluate the classifier. This solution attained 96% for area under curve accuracy measure. A solution to detect malware by monitoring the network traffic was proposed by the authors in [18]. They were able to reduce the detection time by 67%, compared to conventional methods. In [19] the authors combined RNN and CNN to perform hierarchical feature extraction, and used n-gram technique to select appropriate opcodes for malware detection. In [20], the authors conducted a comprehensive comparison between static and dynamic analyses and a hybrid solution using a large number of malware families. They found that dynamic analysis achieved the most accurate results.

Dynamic analysis is free from such drawbacks of static analysis as in detecting unknown malware. However,

it requires higher computational resources and more frequent occurrence of false positive results.

C. IMAGE PROCESSING TECHNIQUE

The recent technique in classifying malicious files is the utilization of image processing algorithms. Malware raw bytes are converted into grayscale images and neural networks are used to classify the image to its corresponding malware family or to classify it as malicious or benign. This technique is motivated by the fact that attackers generate many malware variants from existed malware families, without the need to write the code from the scratch, with the help of packing and metamorphic techniques. The image processing technique identifies and uses the layout and texture of malware to classify it to its corresponding family and distinguish it from benign executables.

The research conducted in [21], [22] addressed the effectiveness in classifying malware to their corresponding families by converting the malware into gray images. The proposed frameworks, Signal Processing Approach to Malware Analysis (SigMal) and Search and Retrieval of Malware (SARVAM), gave very accurate results. In [23], the researchers compared binary texture classification and dynamic analysis classification. The two techniques showed very similar results regarding accuracy. However, binary texture solution was faster with classification. The authors in [24] provided preliminary experimental results of the image processing technique in classifying malware, achieving 98% accurate classification. The malware dataset contained 9,458 samples under 25 different malware families. The researchers in [25]–[27] used image processing techniques to classify malware files by unpacking malware files and then representing their assembly code and opcode as images. The authors evaluated CNN, ResNet and Googlenet algorithms to classify the images to their corresponding classes. The drawback of this solution lies in the necessity of the unpacking process to generate the images. To overcome this limitation, the authors in [28] proposed a hybrid method of malware visualization in a big data environment and evaluated it against public and private malware datasets and achieved high accuracy. The solution converts raw bits of the files into grayscale images, which permits the files to be classified without the need for unpacking them. This image conversion overcomes the packing and metamorphic evasion techniques. Ensemble CNN architecture for malware grayscale image classification has been addressed by the authors in [29]. They evaluated the results of VGG16 and ResNet-50 algorithms, using the Malimg dataset. Later, the authors provided another solution called Image based Malware Classification using Fine-tuned Convolutional Neural Network Architecture (IMCFN), which converts raw malware binaries into images that are used by the fine-tuned CNN architecture to detect and identify malware families [30].

Unlike static and dynamic analyses, implementing the image processing solution does not require strong domain knowledge. Further, it overcomes the packing and metamorphic evasion techniques by not requiring reverse engineering of the file. It converts the raw malware binaries into images. Also, it is fast and works on various malware irrespective of the operating system.

MSIC framework is evaluated and compared against the CNN grayscale image classification solution that has been proposed by the researchers in the literature [28]. In the latter technique, the grayscale images are prepared by converting the raw bits of the files into bit strings and grouping each 8-bit as an unsigned number to represent a pixel color and these are fed to CNN. Differing from the grayscale classification, the proposed MSIC framework converts the raw bits of the files into spectrogram images using audio signals fundamentals and feeds them to CNN. This avoids the need to extract static features, which enables it to overcome evasion techniques, such as packing, polymorphic and metamorphic.

III. MSIC FRAMEWORK

The proposed solution in this work aims to classify malware files to their corresponding families and distinguish them from benign files by utilizing an image processing technique. Such a solution does not require domain knowledge and is fast as compared to other solutions. MSIC is shown in Figure 1. It is divided into two main stages, malware image preparation and malware image classification. These two stages are conducted with the use of audio signals fundamentals and neural network algorithms.

An audio signal can be visualized as a time domain, frequency domain or spectrogram. A time-domain visualizes an audio signal in an x-y plane, showing the amplitude variations of the signal against time. The frequency-domain visualizes an audio signal in an x-y plane, showing the frequencies in the signal with their magnitude. Fourier Transform (FT) is a mathematical concept used to convert a signal from time-domain to frequency-domain. Spectrogram visualization represents an audio signal's frequency with time in an x-y plane. The x-axis represents time and the y-axis represents frequencies. The third dimension in the spectrogram is the magnitude of a frequency at a specific time, where it is represented as a colored heatmap. Spectrogram can be attained by applying the short-time FT (STFT) on the time domain signal. STFT breaks the signal into small windows and calculates the FT for each window. The literature has shown the effectiveness of using spectrogram images for speech recognition [31]–[34]. FT and STFT functions are computed using the following equations for continuous signals:

$$F(\omega) = \int_{-\infty}^{\infty} f(x)\, e^{-i\omega x} dx \tag{1}$$

$$STFT = X(\tau, \omega) = \int_{-\infty}^{\infty} x(t)\, w(t-\tau) e^{-i\omega t} dt \tag{2}$$

where $x(t)$ is the time-domain signal to be transformed, τ is the time, ω is the frequency, $w(t)$ is the window function and $X(\tau, \omega)$ is the complex function representing the phase and magnitude of the signal over time and frequency.

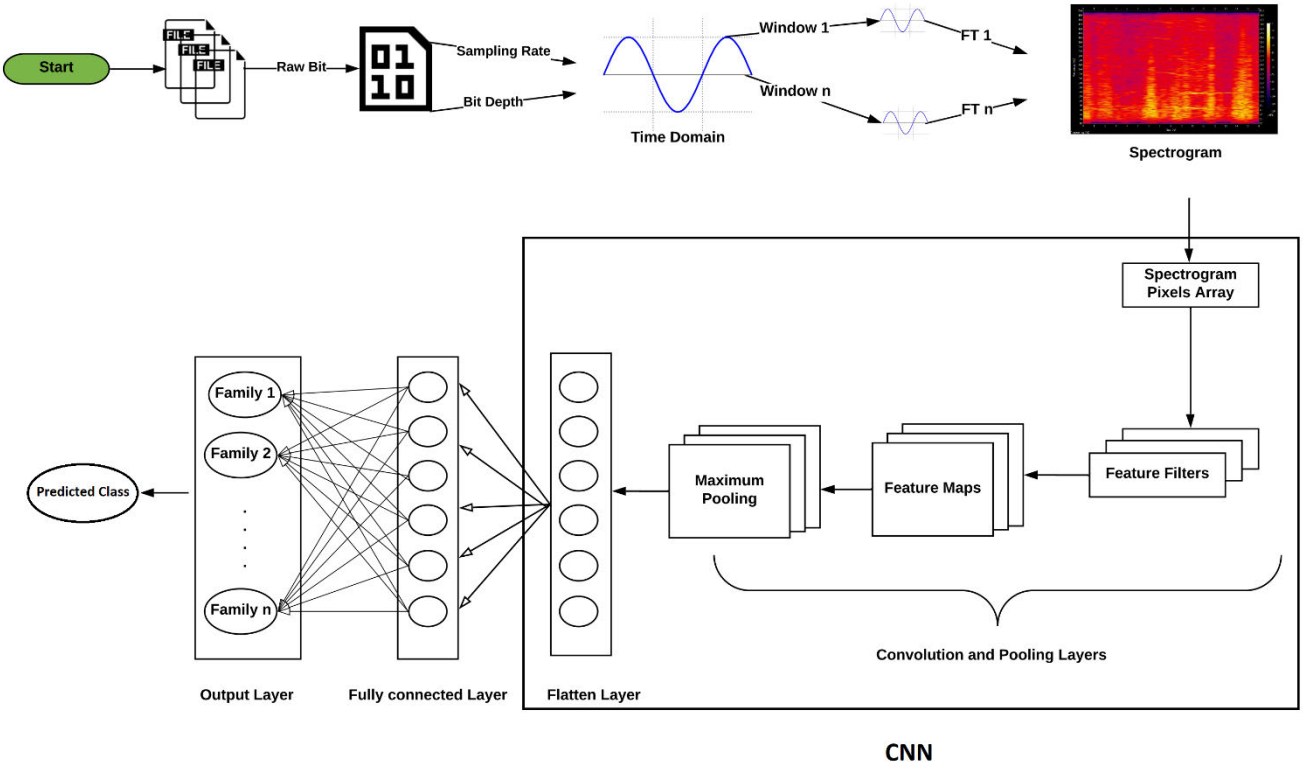

FIGURE 1. MSIC framework.

Analog audio signals are transformed into digital audio signals to be processed by digital devices. To fulfill this, two criteria must be met—sampling rate and bit depth. The sampling rate is defined as the number of samples taken from the analog signal to represent a digital signal, usually measured as the number of samples per second or Hz. For example, 8 KHz = 8000 samples/second. A higher sample rate provides higher quality. Bit depth defines the number of bits used to represent each sample's amplitude. For example, wav audio files usually use 44100 Hz sampling rate and 16 bits as bit depth.

Since signals are stored as numbers on computers, discrete functions must be used. Discrete FT (DFT) is used to convert a discrete signal from time-domain to frequency-domain. Fast FT (FFT) algorithm is used to compute DFT. FFT reduces the DFT computation complexity from $O(N^2)$ to $O(N \log N)$. Spectrogram can be attained using STFT, where the FFT algorithm is computed over each window.

DFT and STFT equation for discrete signals are given below:

$$F_k = \sum_{n=0}^{N-1} x_n . e^{-\frac{j2\pi kn}{N}} \tag{3}$$

$$STFT = X(m, \omega) = \sum_{n=-\infty}^{\infty} x_n w_{n-m} e^{-i\omega t_n} \tag{4}$$

where $X(m, \omega)$ is the STFT of the time-domain sequence ($STFT\{x_n\}(m, \omega)$), x_n is the sequence of discretized time-domain signal to be transformed, m is the time index, ω is the frequency and w_n is the sequence of the discretized window function.

The first stage in MSIC framework aims to visualize the file as a spectrogram image to be used as an input to the neural network classifier. To achieve this, first, the file is converted into raw data of 0s and 1s bit string. Then, the sampling rate and bit depth parameters are defined. Thereafter, the time domain signal is generated by using the sampling rate and bit depth values identified earlier. Subsequently, the generated signal is segmented using symmetrical fixed window sizes with overlaps between them. For each segmented signal, FFT is calculated. The computed FFT over the segmented signals is the process of STFT. The calculated FFT of the different segments are constructed to provide the spectrogram image of the file, which is the representation of how the frequency content of a signal changes with time.

In the second stage, a classifier is built to classify the spectrogram images. To fulfill this, CNN will be used because its effectiveness is proved in the literature [35], [36]. CNN is considered a deep learning algorithm that takes an image, assigns importance to several aspects of the image and differentiates the image from the rest. The CNN aims to reduce the images to a form that is simpler to process and avoids the loss of features that are important for achieving a good prediction. To achieve this, the image is represented as a matrix of pixel values and a filter matrix is identified. The filter slides from the top left of the image to the right with a certain stride value, extracting features at each stride move till it covers the entire width. Then, it moves down and starts from the left of the

TABLE 1. Pseudo code of MSIC framework.

Input: Malware Files
Output: CNN Training and Testing Classifiers
Begin
1. N: The total number of Malware samples
2. M: A Malware sample, where $\boldsymbol{M}_{list} = \{M_1, M_2, \ldots, M_N\}$
3. **For** J=1 to N // Produce raw bit strings and time domain signals
   a. B_J = raw bit string of M_J
   b. Set sampling rate
   c. Set bit depth
   d. TD_J = Time domain signal of B_J
4. **End for**
5. SP: A spectrogram image
6. W: Number of windows
7. **For** J=1 to N //STFT computation and spectrogram preparation
   a. For K=1 to W
      i. S_K = Signal Segmentation of TD_j
      ii. Calculate FFT of S_K
      iii. Shift Window
   b. Construct SP_J
8. **End for**
9. T= Number of training samples
10. E= Number of evaluation samples
11. FN= Number of feature filters
12. FS: A feature filter, where $\boldsymbol{FS}_{list} = \{FS_1, FS_2, \ldots, FS_{FN}\}$
13. FM: The feature map resulted from applying FS
14. MP: The pooling layer with maximum parameter
15. **For** J=1 to T //CNN Training
   a. For K=1 to FN
      i. Convolve FS_K over SP_J
      ii. Apply MP_K with FM_K as an input
   b. **End for**
   c. Let F_J be the flatten data from MP_J
   d. F_J fed to fully connected Neural Network
16. **End for**
17. **For** J=1 to E //CNN Testing
   a. For K=1 to FN
      i. Convolve FS_K over S_J
      ii. Extract MP_K from FM_K
   b. **End for**
   c. Let F_J be the flatten data from MP_J
   d. F_J fed to fully connected Neural Network
18. **End for**
End

image with the same stride value. It repeats this process to cover the entire image. The extracted features are set using the defined filter to obtain a feature map. The optimal number of filters and their size are usually identified by the parameter tuning approach. The filter uses Rectified Linear Units (ReLU), a non-linear activation function, on each element.

After the convolution layer is processed, each feature map is fed into the pooling layer. This layer aims to reduce the spatial size of the convolved feature, thus reducing the computational requirement to process the data. The pooling layer utilizes maximum, average or minimum pooling techniques. Maximum pooling was used since it selects the maximum output from the image and suppresses noise. Finally, after the output from the pooling layer is flattened, it is fed to a fully connected neural network for the classification process. The pseudo-code for MSIC is illustrated in Table 1, which summarizes the steps of spectrogram preparation and CNN building processes.

IV. EXPERIMENT ENVIRONMENT AND EVALUATION METRICS

The data collection and experimental evaluation procedures in this research have been conducted in an isolated environment to avoid any unintentional malware infection or spread. To ensure the prevention or spread of malware, the following measures were applied. First, the workstation used for data collection and experimental evaluation were not connected to any local network, which has a dedicated Internet connection. Second, a Linux virtual machine was used to control the connections. Third, all the outbound connections were blocked, except to those pre-defined addresses that were necessary for our experiment. The experiment's workstation runs Ubuntu 19.10 (64-bit) with intel core i7 7th generation, 512 GB SSD, 16 GBRAM and 1 Gbps network card.

For evaluation purposes, four metrics have been identified: accuracy, recall, precision and F-measure. Recall measures the number of positive class predictions made from among all the positive examples in the entire dataset. Precision measures how precise are the positive class predictions made by the built model. F-measure is defined as the weighted average of both recall and precision and its effectiveness in an imbalanced dataset [37]. To measure these metrics, True Positive (TP), True Negative (TN), False Positive (FP) and False Negative (FN) must be identified. TP is the outcome of correctly identified number samples by the classifier as a positive class. TN is the outcome of correctly identified number samples by the classifier as a negative class. FP is the outcome of incorrectly identified number samples by the classifier as a positive class. FN is the outcome of incorrectly identified number samples by the classifier as a negative class. Accuracy, recall, precision and F-measure metrics are given by the following equations:

$$Accuracy = \frac{(\#TP + \#TN)}{(\#TP + \#TN + \#FP + \#FN)} \quad (5)$$

$$Recall = \frac{\#TP}{\#TP + \#FN} \tag{6}$$

$$Precision = \frac{\#TP}{\#TP + \#FP} \tag{7}$$

$$F - measure = 2 \times \left(\frac{Precision.Recall}{Precision + Recall} \right) \tag{8}$$

V. MALWARE FAMILIES CLASSIFICATION EXPERIMENT

This experiment aims to build a multiclass classifier to classify malicious files to their corresponding families and evaluate them against privately collected data. To achieve this, first, a labeled dataset was required to be prepared. This was done manually in our labs. Then, MSIC framework was built and evaluated. Later, the raw byte grayscale image classification framework was built and evaluated. The grayscale image classification associates each raw byte of a file to a pixel color. Black has a value of 0, white has a value of 255 and the rest of the values represent intermediate shades of gray [38].

A. DATASET DESCRIPTION

Preparing a labeled malware dataset is not a trivial task. Further, no single site provides a malware dataset with enough labeled files. Although some sites provide public datasets, such as Malimg, they are only grayscale images and cannot be used for other types of analyses besides grayscale classification. Therefore, labeled malware files must be acquired, so they can be converted into spectrogram images for the MSIC framework. For collecting and labeling the data, an isolated environment was created, in which the process illustrated in Figure 2 was followed. First, malware files were downloaded from three main sources: Malshare,[1] Virusign[2] and Dasmalware.[3] Second, the hash values of the collected files were uploaded to Virustotal website. Third, the JSON files of the uploaded hashes were downloaded from VirusTotal. Fourth, AVClass tool [39] was used to label the malware samples using the collected JSON files. Fifth, the obtained labels were mapped to the malware files. The collected dataset is summarized in Table 2 and has been shared publicly for the research community [40]. In total, 9187 malware files representing 11 malware families were collected. The data were divided into datasets, namely, 60% train, 20% validate and 20% test. The training dataset was used to build the classifier. Validate dataset was used for selecting the best parameters to be used during the building process. Test dataset was used to evaluate the accuracy of the built classifier in classifying unseen malware files to their corresponding families.

B. MSIC RESULTS AND ANALYSIS

It was necessary to represent the collected files as spectrogram images to be fed to the CNN classifier as illustrated in Figure 1. For that purpose, the files were first converted into raw bit strings. Then the sampling rate and bit depth were set to 44100 and 16-bit signed integer respectively. These parameters were the same as the wav audio files. Thereafter, spectrogram visualization of the files has been achieved using discrete STFT with a Hanning window [41]. Figure 3 shows a spectrogram sample of emotet malware. One expected observation of the resulted spectrogram is the low periodicity behavior. This is predictable as the nature of the audio files have a higher periodicity compared to non-audio files. The resultant spectrogram images had different dimensions in terms of the pixels' height and width. The reason for this was the sizes of the files in the collected dataset ranged from bytes to megabytes. Since the input of CNN must be symmetric images, python resize functionality was used to have input images of a standard size of 200∗200 pixels.[4] The spectrograms of all malware files were obtained and used to build and evaluate the CNN classifiers. Train and validate datasets were used to build and select the most suitable parameters for the built CNN classifiers. Data augmentation was applied to the training dataset, such as zoom, crop and flip to reduce the overfitting impact. The following parameters were tested to select for building the best classifier:

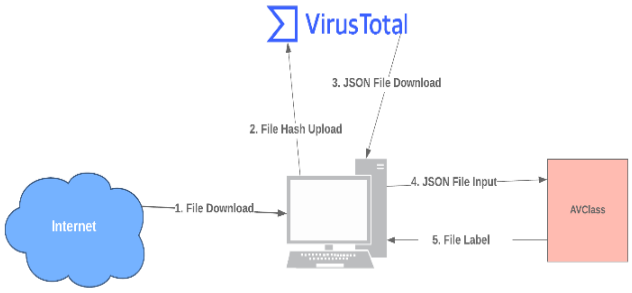

FIGURE 2. Malware families dataset collection and labelling process.

TABLE 2. Malware families dataset.

Family	Train	Validate	Test	Total
coinhive	243	81	81	405
emotet	545	181	181	907
fareit	950	317	317	1584
gafgyt	891	297	297	1485
gandcrab	234	78	78	390
icedid	275	92	92	459
lamer	416	138	138	692
mepaow	492	164	164	820
mirai	864	288	288	1440
ramnit	202	68	67	337
razy	400	134	134	668
Total	**5512**	**1838**	**1837**	**9187**

[1] https://www.malshare.com/
[2] https://www.virusign.com/
[3] https://dasmalwerk.eu/
[4] https://github.com/AhmadAzab/MSIC-Paper.git

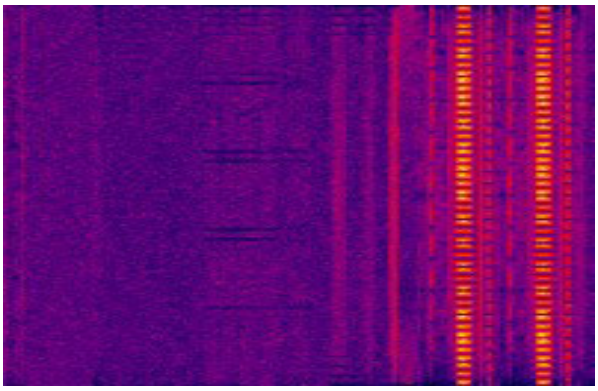

FIGURE 3. Spectrogram image of emotet malware.

- $200 \times 200 \times 3$ input shape
- 16, 32 and 64 filters with filter length 3
- Maxpooling with length 2
- ReLU
- Adam optimizer learning rates 0.1, 0.01 and 0.001
- Sotmax activation function
- Categorical-cross entropy loss function
- 100 epochs
- Batch size of 10
- Adding and removing dropout layer

The various experiments showed learning rate value 0.001 as the effectiveness, compared to 0.1 and 0.01, regardless of the number of the filters and the use of the dropout layer, throughout the 100 epochs. Therefore, 0.001 was selected for the rest of the experiment. 16 filters showed lower accuracy results with a high fluctuation loss pattern as compared to 32 and 64 filters. With 32 and 64 filters, the results showed similar accuracy and loss behavior. The number of filters used in building our classifier for resources purposes was 32. The validation results showed that the performance with the dropout layer was better than without it, regardless of the number of the filters and learning rates. Therefore, the dropout layer was integrated into our classifier. As a result, the best classifier's parameters were 0.001 learning rate, 32 filters and the presence of the dropout layer. This classifier was used for the rest of the experiment.

The validate dataset was used to compare three network structures, CNN_1, CNN_2 and CNN_3, using 1, 2 and 3 layers respectively. Table 3 summarizes the number of parameters of the three structures. Figure 4 and Figure 5 show the accuracy and loss behavior over the 100 epochs. Overall, the three structures showed good accuracy results, where CNN_1 scored the lowest accuracy during the 100 epochs. The accuracy of CNN_2 was comparable to CNN_3, with the latter having a slightly better accuracy. The loss behavior of the three structures showed a high loss pattern before reaching epoch 20. Thereafter they stabilized. Two spikes were noticed afterwards, specifically in epochs 81 and 90, where the latter had the highest loss spike throughout the 100 epochs. By the 100^{th} epoch, the three network structures showed low loss results.

TABLE 3. CNN parameters of the malware families experiment.

Layer (type)	Output Shape	Param #
CNN_1		
conv2d_1 (Conv2D)	(None, 198, 198, 32)	896
max_pooling2d_1	(MaxPooling2 (None, 99, 99, 32)	0
flatten_1 (Flatten)	(None, 313632)	0
dense_1 (Dense)	(None, 128)	40145024
dropout_1 (Dropout)	(None, 128)	0
dense_2 (Dense)	(None, 2)	1419
Trainable params: 40,147,339		
CNN_2		
conv2d_1 (Conv2D)	(None, 198, 198, 32)	896
max_pooling2d_1	(MaxPooling2 (None, 99, 99, 32)	0
conv2d_2 (Conv2D)	(None, 97, 97, 32)	9248
max_pooling2d_2	(MaxPooling2 (None, 48, 48, 32)	0
flatten_1 (Flatten)	(None, 73728)	0
dense_1 (Dense)	(None, 128)	9437312
dropout_1 (Dropout)	(None, 128)	0
dense_2 (Dense)	(None, 2)	1419
Trainable params: 9,448,875		
CNN_3		
conv2d_1 (Conv2D)	(None, 198, 198, 32)	896
max_pooling2d_1	(MaxPooling2 (None, 99, 99, 32)	0
conv2d_2 (Conv2D)	(None, 97, 97, 32)	9248
max_pooling2d_2	(MaxPooling2 (None, 48, 48, 32)	0
conv2d_3 (Conv2D)	(None, 46, 46, 32)	9248
max_pooling2d_3	(MaxPooling2 (None, 23, 23, 32)	0
flatten_1 (Flatten)	(None, 16928)	0
dense_1 (Dense)	(None, 128)	2166912
dropout_1 (Dropout)	(None, 128)	0
dense_2 (Dense)	(None, 2)	1419
Trainable params: 2,187,723		

Test dataset has been used to evaluate the three built structures in classifying unseen malware samples. Table 4 lists the levels of accuracy, precision, recall and F-measure resulting from the three structures. CNN_1 attained 90.5% recall and precision, 90.4% F-measure and 91.7% accuracy. The lowest F-measure class result was in case of ramnit malware, whereas the highest F-measure class result was with icedid malware. CNN_2 at 90.2% scored less than CNN_1 recall result but scored higher precision, F-measure and accuracy with 91.1%, 90.6% and 92% results respectively. Like CNN_1, the lowest F-measure class result in CNN_2 was ramnit malware and the highest was icedid malware. CNN_3 outperformed the other two classifiers,

TABLE 4. MSIC evaluation results of the malware families experiment.

Model	Metric (%)	coinhive	emotet	fareit	gafgyt	gandcrab	icedid	lamer	mepaow	mirai	ramnit	razy	Average
CNN_1	Recall	100	95	90.2	99	88.5	98.9	89.9	91.5	91.3	71.6	79.1	90.5
	Precision	96.4	88.7	96.9	91.9	93.2	98.9	89.9	91.5	94.3	73.8	80.3	90.5
	F-measure	98.2	91.7	93.5	95.3	90.8	98.9	89.9	91.5	92.8	72.7	79.7	90.4
	Accuracy	91.7											
CNN_2	Recall	96.3	95	90.9	99	88.5	98.9	89.1	93.3	93.1	65.7	82.1	90.2
	Precision	100	89.1	96.3	93	95.8	98.9	91.8	91.1	93.4	73.3	79.7	91.1
	F-measure	98.1	92	93.5	95.9	92	98.9	90.4	92.2	93.2	69.3	80.9	90.6
	Accuracy	92											
CNN_3	Recall	100	96.1	92.4	99	88.5	98.9	89.9	93.3	93.1	76.1	79.1	**91.5**
	Precision	100	92.6	93.6	93.3	93.2	98.9	93.2	91.6	96.1	73.9	84.1	**91.9**
	F-measure	100	94.3	93	96.1	90.8	98.9	91.5	92.4	94.5	75	81.5	**91.6**
	Accuracy	**92.8**											

TABLE 5. CNN_3 MSIC confusion matrix of the malware families experiment.

		Predicted Classes										
		coinhive	emotet	fareit	gafgyt	gandcrab	icedid	lamer	mepaow	mirai	ramnit	razy
True Classes	coinhive	**81**	0	0	0	0	0	0	0	0	0	0
	emotet	0	**174**	3	0	1	0	0	0	2	0	1
	fareit	0	4	**293**	0	1	1	1	0	4	4	9
	gafgyt	0	0	0	**294**	1	0	0	0	2	0	0
	gandcrab	0	1	3	0	**69**	0	0	0	0	3	2
	icedid	0	1	0	0	0	**91**	0	0	0	0	0
	lamer	0	0	0	0	0	0	**124**	12	0	0	0
	mepaow	0	0	1	0	0	0	8	**153**	0	1	1
	mirai	0	0	1	19	0	0	0	0	**268**	0	0
	ramnit	0	3	5	0	0	0	0	0	1	**51**	7
	razy	0	5	7	2	2	0	0	0	2	10	**106**

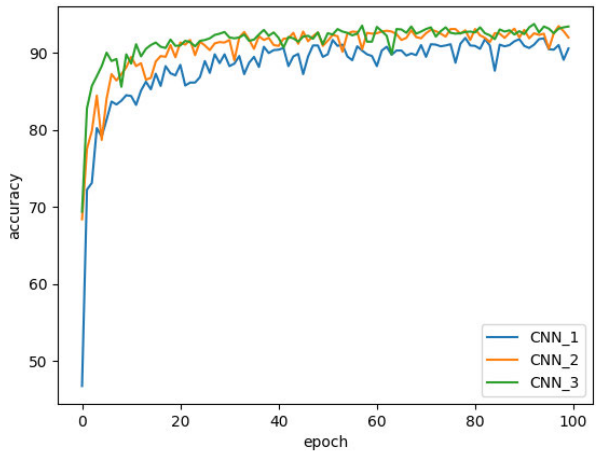

FIGURE 4. MSIC accuracy of the malware families experiment.

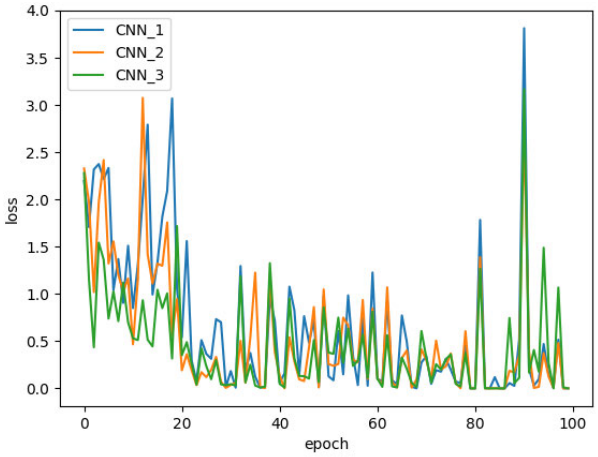

FIGURE 5. MSIC Loss of the malware families experiment.

achieving 91.5%, 91.9%, 91.6% and 92.8% for recall, precision, F-measure, and accuracy, respectively. Ramnit class scored the least evaluation metrics, where coinhive scored the highest.

The confusion matrix of the CNN_3 structure is listed in Table 5. For the least class accuracy result, that was with ramnit, the classifier successfully classified 51 out of 67 files. However, it misclassified 7 files as razy, 5 as fareit, 3 as emotet and 1 as mirai. For the second least class, razy, the classifier correctly classified 106 out of 134 files. However, it misclassified 10 files as ramnit, 7 as fareit, 5 as emotet,

TABLE 6. Grayscale evaluation results of the malware families experiment.

Model	Metric (%)	coinhive	emotet	fareit	gafgyt	gandcrab	icedid	lamer	mepaow	mirai	ramnit	razy	Average
CNN_1	Recall	100	96.1	92.1	97.3	83.3	98.9	85.5	90.9	92.4	26.9	76.1	85.4
	Precision	90	82.5	92.7	92.3	95.6	98.9	90.8	88.7	93.3	81.8	71.3	88.9
	F-measure	94.7	88.8	92.4	94.8	89	98.9	88.1	89.8	92.8	40.4	73.6	85.8
	Accuracy	89.5											
CNN_2	Recall	93.8	95.6	90.9	98.3	88.5	98.9	85.5	93.3	93.1	43.3	79.9	87.4
	Precision	95	88.7	95	92.7	93.2	92.9	94.4	87.9	94.4	82.9	69.5	89.7
	F-measure	94.4	92	92.9	95.4	90.8	95.8	89.7	90.5	93.7	56.9	74.3	87.9
	Accuracy	90.6											
CNN_3	Recall	100	97.2	89.9	98.7	91	98.9	89.9	90.0	94.1	71.6	79.1	**91**
	Precision	95.3	86.3	98.6	94.8	87.7	95.8	91.9	91.4	97.8	73.8	79.1	**90.2**
	F-measure	97.6	91.4	94.1	96.7	89.3	97.3	90.8	91.1	95.9	72.7	79.1	**90.6**
	Accuracy	**92.3**											

2 as gafgyt, 2 as gandcrab and 2 as mirai. The results suggest a spectrogram image similarity between the ramnit and razy malware. Of the two best performing classes, CNN_3 correctly classified all coinhive files and 294 out of 297 files of gafgyt malware.

C. GRAYSCALE RESULTS AND ANALYSIS

This experiment aims to evaluate the accuracy of the classification of the malware files to their corresponding families using the grayscale framework that has been used by researchers in the literature, e.g., in [28]. For this, first, the grayscale images were prepared by converting the files into bit strings and grouped each 8-bit as an unsigned number to represent a pixel color. As in the MSIC experiment, train and validate datasets were used to build and select the best parameters. Thereafter, the test dataset was deployed to evaluate the selected classifiers. The same CNN parameters mentioned earlier were used, except for the input shape. In this experiment, the shape used was $200 \times 200 \times 1$ for the input. Python resize functionality was used for the purpose.

After going through many experiments, the following findings were observed. Learning rates 0.1 and 0.01 provided good accuracy and loss results for different numbers of filters. However, the learning rate at 0.001 was the best result among the three. With the use of the dropout layer, fluctuation loss behavior was reduced as compared to when not using it, regardless of the number of the filters used. The utilization of 64 filters showed better accuracy and loss results than the 32 and 16 filters across the 100 epochs. As a result, learning rate 0.001, the addition of the dropout layer and 64 filters were the parameters used for the rest of the experiment.

For selecting the network structure, three models were built and evaluated using the train and validate datasets along with the identified best parameters. The models were named CNN_1, CNN_2 and CNN_3 using 1, 2 and 3 layers respectively. Figure 6 and Figure 7 show the accuracy and loss behavior of each structure, across 100 epochs. CNN_3 showed the best accuracy, CNN_2 demonstrated

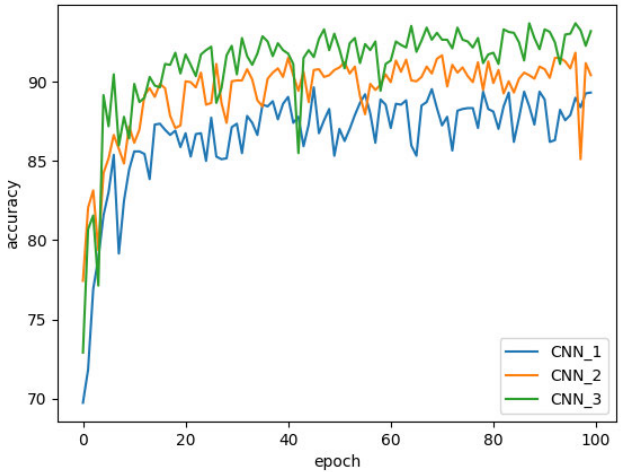

FIGURE 6. Grayscale accuracy of the malware families experiment.

better accuracy behavior than CNN_1, although it showed a drop in accuracy in epoch 97. In terms of the loss results, CNN_3 gave the best outcome with the least fluctuation behavior. On the other hand, CNN_1 showed the highest fluctuation with the worst results. The loss pattern in CNN_2 was comparable to that of CNN_3, although it had a spike in epoch 90.

The three network structures have been evaluated against unseen malware files, using the test dataset. Table 6 describes the attained results. CNN_1 showed the least performance at 85.4%, 88.9%, 85.8% and 89.5% for recall, precision, F-measure and accuracy metrics, respectively. It is noticed that ramnit class demonstrated the lowest results compared to the other classes. CNN_2 showed better results with 87.4% for recall, 89.7% for precision, 87.9% for F-measure and 90.6% for accuracy. As in the case of CNN_1, ramnit class reflected the poorest performance. CNN_3 outperformed CNN_1 and CNN_2 classifiers with 91% in recall, 90.2% in precision, 90.6% in F-measure and 92.3 in accuracy. Ramnit class's performance was better as compared to CNN_1 and CNN_2 but remained the lowest amongst the all the classes.

TABLE 7. CNN_3 Grayscale confusion matrix of the malware families experiment.

True Classes \ Predicted Classes	coinhive	emotet	fareit	gafgyt	gandcrab	icedid	lamer	mepaow	mirai	ramnit	razy
coinhive	**81**	0	0	0	0	0	0	0	0	0	0
emotet	0	**176**	0	0	1	2	0	0	0	0	2
fareit	1	6	**285**	0	4	0	0	0	3	6	12
gafgyt	0	1	0	**293**	0	0	0	0	2	0	1
gandcrab	0	3	0	0	**71**	1	0	0	0	0	3
icedid	0	0	0	0	0	**91**	0	0	0	0	1
lamer	0	0	0	0	0	0	**124**	14	0	0	0
mepaow	0	0	0	0	1	0	11	**149**	0	2	1
mirai	0	1	0	16	0	0	0	0	**271**	0	0
ramnit	0	8	1	0	1	1	0	0	0	**48**	8
razy	3	9	3	0	3	0	0	0	1	9	**106**

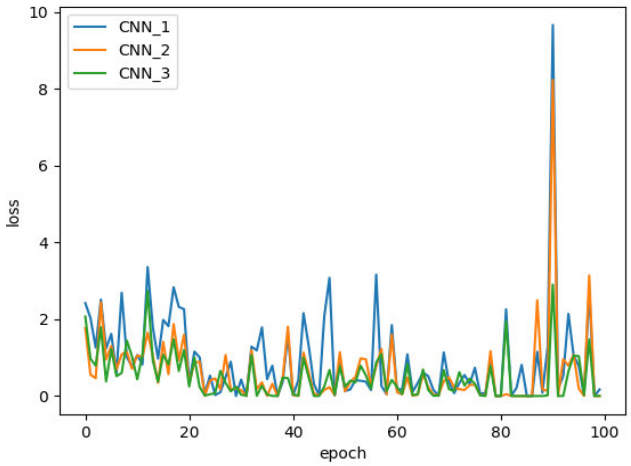

FIGURE 7. Grayscale loss of the malware families experiment.

The confusion matrix of CNN_3 is illustrated in Table 7 for the 11 classes. CNN_3 correctly classified most of the files to their corresponding families. The lowest class results, 48 out of 67 files correctly classified, was for ramnit. Of the 19 incorrectly classified files, 8 were classified as razy, 8 as emotet, 1 as fareit, 1 as gandcrab and 1 as icedid. The next in inaccuracy was razy with 106 files out of 134 identified correctly. The incorrect predictions of it were 9 files as ramnit, 9 as emotet, 3 as coinhive, 3 as fareit, 3 gandcrab and 1 as mirai. Those results indicate the similarity between razy, ramnit and emotet malware, where they target windows OS. On the other hand, the classifier successfully predicted all coinhive files. Also, it was able to detect 91 out of 92 files of iceid malware.

Malware families experiment proved the effectiveness of MSIC in classifying malware files to their corresponding families irrespective of the OS they work on or their type. Further, MSIC outperformed the grayscale framework for the used evaluation metrics.

TABLE 8. Malicious-Benign dataset.

	Benign	Malicious
Train	1566	1628
Validate	522	498
Test	523	500
Total	2611	2626

VI. MALICIOUS-BENIGN CLASSIFICATION EXPERIMENT

This experiment aims to build a binary classifier that classifies benign and malicious files. First, MSIC was built and evaluated. Afterwards, the grayscale classification framework has been built and evaluated.

A. DATASET DESCRIPTION

A dataset must be collected to build and evaluate MSIC and the grayscale frameworks. The dataset should contain both benign and malicious files. The collection process used the environment and workstation's specification explained earlier. The sources of the collected files were Malshare, Virusign and Dasmalware websites. The privately collected dataset is described in Table 8 and has been publicly shared to the research community [42]. The total number of the benign files is 2611, where the total number of the malicious files is 2626. The dataset was divided into 60% train, 20% validate and 20% test datasets. Train dataset is used to build the classifier, validate dataset is utilized to select the best parameters during the building process. Test dataset is used to evaluate the built classifier in classifying unseen malicious and benign files.

B. MSIC RESULTS AND ANALYSIS

The collected files were converted into spectrogram images to be fed to the CNN classifier. The same process in the previous experiment was followed for the conversion into

spectrogram images. Wav sampling rate and bit depth values and Hanning window were used for the conversion purpose. The resulting spectrograms of all the benign and malicious files were used to build and evaluate the CNN classifiers. Train and validate datasets were used to build and select the most suitable parameters for the built CNN classifiers. Data augmentation was applied to reduce the overfitting impact. The same parameters listed in the previous experiment were used to select the best classifier.

The following findings were arrived at after intensive experiments. For learning rates 0.1 and 0.01, the accuracy results were low and the loss results were high, for any number of the filters or the use of the dropout layer. Learning rate 0.001 provided the best accuracy and loss results for the various conducted experiments. 16 filters with the dropout layer achieved slightly more accurate results with lower fluctuation loss behavior as compared to using 16 filters without the dropout layer. Similarly, 32 filters with the dropout layer showed better accuracy and less loss, compared to 32 filters without the dropout layer. Finally, 64 filters showed a considerable fluctuation in the high losses, whether the dropout layer was used or not. However, the accuracy results were acceptable. Compared to 16 and 64 filters, the classifier with 32 filters and with the dropout layer provided the best accuracy and loss results along with the least fluctuation throughout the 100 epochs. Therefore, these were selected as the best parameters for the rest of the experiment.

To determine the network structure, three CNN models have been evaluated through 100 epochs against the validate dataset, using 1, 2 and 3 layers. The summaries of the parameters for each model are shown in Table 9. The accuracy and loss behavior for CNN_1, CNN_2 and CNN_3 are shown in Figure 8 and Figure 9. CNN_3 model showed the most stable and highest accuracy results throughout the 100 epochs. CNN_1 and CNN_2 showed less accuracy and stability results than CNN_3. On reaching the 100^{th} epoch, CNN_1 and CNN_2 gave similar accuracy results. The loss figure reflects that CNN_1 had the most fluctuating behavior, with high spikes during the 100 epochs. CNN_2 was the most stable with the best loss pattern, starting from epoch 20. Although CNN_3 reflected higher fluctuation pattern than CNN_2, it became stable with the same result as those of CNN_2 by epoch 100.

The three classifiers have been evaluated against unseen samples, using the test dataset. Table 10 shows the accuracy, recall, precision and F-measure results for benign and malicious classes and their average. While all the classifiers achieved good accuracy results, CNN_3 provided the best performance. CNN_1 correctly identified 90.4% and 96.6% of the benign and malicious samples, respectively. The predicted precision for benign and malicious samples was 96.5% and 90.6%, respectively. CNN_2 identified 93.5% of the benign samples, with a precision of 95.3% and detected 95.2% of the malicious files with a precision of 93.3%. CNN_3 detected 95.2% of the benign samples with a precision of 96.9% and identified 96.8% of the malicious samples with a precision of 95.1%. CNN_3 outperformed the other two classifiers, with an average recall precision, F-measure and accuracy of 96%.

TABLE 9. CNN parameters of the malicious-benign experiment.

Layer (type)	Output Shape	Param #
CNN_1		
conv2d_1 (Conv2D)	(None, 198, 198, 32)	896
max_pooling2d_1	(MaxPooling2 (None, 99, 99, 32)	0
flatten_1 (Flatten)	(None, 313632)	0
dense_1 (Dense)	(None, 128)	40145024
dropout_1 (Dropout)	(None, 128)	0
dense_2 (Dense)	(None, 2)	258
Trainable params: 40,146,178		
CNN_2		
conv2d_1 (Conv2D)	(None, 198, 198, 32)	896
max_pooling2d_1	(MaxPooling2 (None, 99, 99, 32)	0
conv2d_2 (Conv2D)	(None, 97, 97, 32)	9248
max_pooling2d_2	(MaxPooling2 (None, 48, 48, 32)	0
flatten_1 (Flatten)	(None, 73728)	0
dense_1 (Dense)	(None, 128)	9437312
dropout_1 (Dropout)	(None, 128)	0
dense_2 (Dense)	(None, 2)	258
Trainable params: 9,447,714		
CNN_3		
conv2d_1 (Conv2D)	(None, 198, 198, 32)	896
max_pooling2d_1	(MaxPooling2 (None, 99, 99, 32)	0
conv2d_2 (Conv2D)	(None, 97, 97, 32)	9248
max_pooling2d_2	(MaxPooling2 (None, 48, 48, 32)	0
conv2d_3 (Conv2D)	(None, 46, 46, 32)	9248
max_pooling2d_3	(MaxPooling2 (None, 23, 23, 32)	0
flatten_1 (Flatten)	(None, 16928)	0
dense_1 (Dense)	(None, 128)	2166912
dropout_1 (Dropout)	(None, 128)	0
dense_2 (Dense)	(None, 2)	258
Trainable params: 2,186,562		

C. GRAYSCALE FRAMEWORK RESULTS AND ANALYSIS

The grayscale images of all the benign and malicious files were used to build and evaluate the CNN classifiers. Train and validate datasets were used in building and selecting the most suitable parameters for the built CNN classifiers. Data augmentation was used on the training dataset. The same parameters as used in the previous experiment were used, except that the input shape selected was $200 \times 200 \times 1$.

The applied experiments showed the following findings. The learning rates values of 0.1 and 0.01 achieved low accuracy and high loss results, whatever was the number of filters

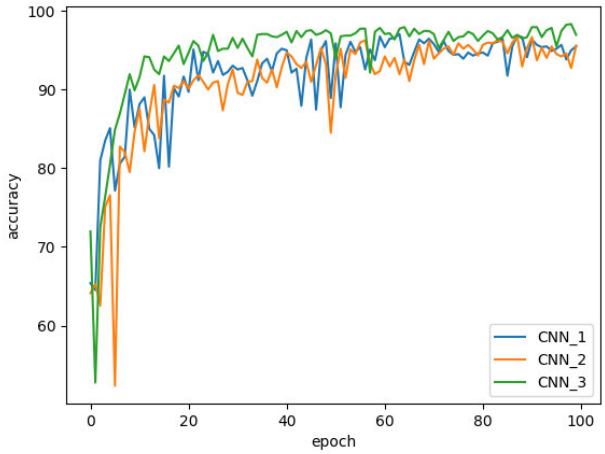

FIGURE 8. MSIC accuracy of the malicious-benign experiment.

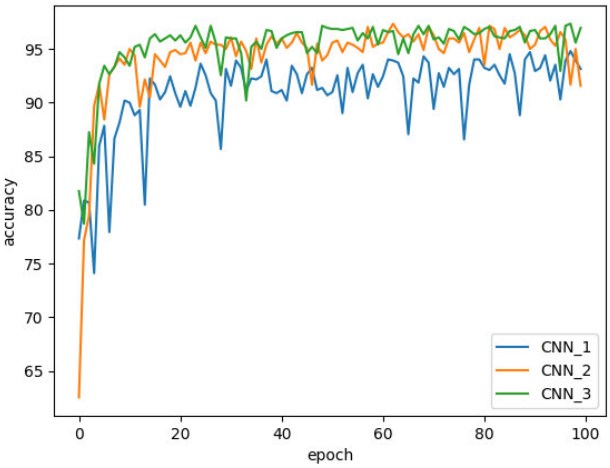

FIGURE 10. Grayscale accuracy of the malicious-benign experiment.

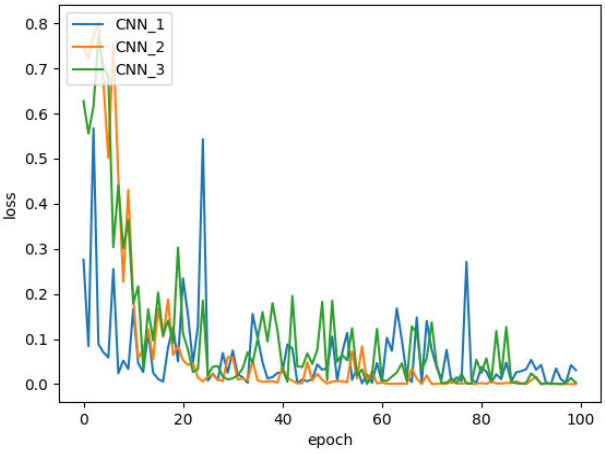

FIGURE 9. MSIC loss of the malicious-benign experiment.

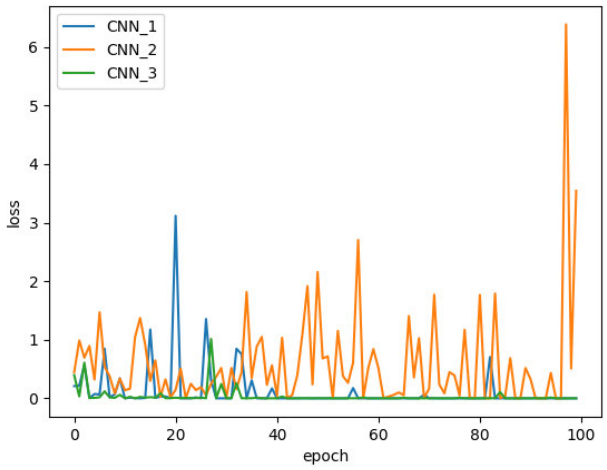

FIGURE 11. Grayscale loss of the malicious-benign experiment.

TABLE 10. MSIC evaluation results of the malicious-benign experiment.

Model	Class	Recall (%)	Precision (%)	F-measure (%)	Accuracy (%)
CNN_1	Benign	90.4	96.5	93.4	93.5
	Malicious	96.6	90.6	93.5	
	Average	93.5	93.6	93.5	
CNN_2	Benign	93.5	95.3	94.4	94.3
	Malicious	95.2	93.3	94.3	
	Average	94.3	94.3	94.3	
CNN_3	Benign	95.2	96.9	96	**96**
	Malicious	96.8	95.1	95.9	
	Average	**96**	**96**	**96**	

used. On the other hand, the learning rate 0.001 attained high accuracy with low loss results. The experiments showed the effectiveness of adding the dropout layer for 16, 32 and 64 filters. The addition of the layer raised the level of accuracy and lowered the loss fluctuation. The deployment of 32 filters provided the best accuracy and loss results compared to 16 and 64 filters. Amongst all the filters, a classifier with 64 filters showed the highest fluctuation loss pattern. As a result, the best parameters for the classifier were learning rate 0.001, 32 filters with the addition of the dropout layer. These were used for the remaining of the experiment.

Three network structures were built and evaluated using 1, 2 and 3 layers. The accuracy and loss results achieved are shown in Figure 10 and Figure 11. The accuracy results of CNN_1 were the lowest but improved by the end of the 100 epochs. On the other hand, CNN_1 achieved good loss results, despite the spike in epoch 20. CNN_2 showed better accuracy than CNN_1 throughout the 100 epochs but its accuracy dropped just before the 100^{th} epoch. CNN_2 showed the worst loss results with a high fluctuation pattern during the 100 epochs, especially right before the 100^{th} epoch. Among the three structures, CNN_3 achieved the best accuracy and loss results all through the 100 epochs. The number of parameters for the three classifiers was the same as in the MSIC experiment.

TABLE 11. Grayscale evaluation results of the malicious-benign experiment.

Model	Class	Recall (%)	Precision (%)	F-measure (%)	Accuracy (%)
CNN_1	Benign	88.1	94.7	91.3	91.4
	Malicious	94.8	88.4	91.5	
	Average	91.5	91.5	91.4	
CNN_2	Benign	95.6	88.3	91.8	91.3
	Malicious	86.8	95	90.7	
	Average	91.2	91.7	91.3	
CNN_3	Benign	94.1	97	95.5	**95.5**
	Malicious	97	94	95.5	
	Average	**95.5**	**95.5**	**95.5**	

TABLE 12. MSIC and Grayscale frameworks best results comparison.

Experiment	Framework	Avg. Recall %	Avg. Precision %	Avg. F-measure %	Accuracy %
Malware Families	MSIC	91.5	91.9	91.6	92.8
	Grayscale	91	90.2	90.6	92.3
Malicious-Benign	MSIC	96	96	96	96
	Grayscale	95.5	95.5	95.5	95.5

The three built structures were tested with the test dataset to evaluate their effectiveness in classifying unseen files. The test results are presented in Table 11. CNN_1 successfully identified 88.1% of the benign samples and 94.8% of the malicious samples. The precision's values of the benign and malicious classes were 94.7% and 88.4% respectively. CNN_2 detected 95.6% of the benign samples and 86.8% of the malicious samples, that is, precision values of 88.3% and 95%. CNN_3 outperformed the other two classifiers with average recall, precision, F-measure and accuracy results of 95.5%. CNN_3 could detect 94.1% of the benign samples and 97% of the malicious samples. The precision was 97% for benign samples and 94% for the malicious samples.

Table 12 illustrates the best results achieved with both MSIC and grayscale frameworks in terms of classifying malware families and distinguishing malicious-benign files. MSIC is seen to have performed better on recall, precision, F-measure and accuracy metrics on both tests.

The computational time of the different phases of MSIC and grayscale frameworks are depicted in Table 13 for both malware families and malicious-benign experiments. For the malware families' experiment, MSIC required 723.96 seconds to convert the 9187 malware files into spectrogram images, with an average of 0.079 seconds per file. On the other hand, the grayscale framework spent 3837.58 seconds to convert the malware files, with an average of 0.42 seconds per file. The image resizing process of MSIC framework required less time than the grayscale framework, with 148.74 seconds compared to 250.08 for all the images. The classification time of the test dataset were almost identical

TABLE 13. Computational time comparison.

Phase	Required Time in Seconds	
	MSIC Framework	Grayscale Framework
Malware Families File Conversion	723.96	3837.58
Malware Families Image Resizing	148.74	250.08
Malware Families Image Classification	50.39	52.04
Malicious-Benign File Conversion	720.74	2424.28
Malicious-Benign Image Resizing	90.64	200.56
Malicious-Benign Image Classification	12.17	11.54

for both frameworks. On average, a single image required 0.027 seconds to be classified.

For the malicious-benign experiment, MSIC required 720.74 seconds to convert the 5237 files into spectrogram images, with an average of 0.14 seconds per file. On the contrary, grayscale converted the files in 2424.28 seconds, with an average of 0.46 seconds per file. MSIC's resizing process time was less than the grayscale framework by 109.92 seconds. The classification time of the test dataset were nearly equal. On average, a single image needed.012 seconds to be classified. It can be concluded that the image preparation in MSIC framework, in terms of conversion and resizing, needed less time than the grayscale framework. The classification times were almost equal since both have the same CNN parameters and input shape.

VII. CONCLUSION

The arms race between cybersecurity analysts and cybercriminals is intensifying. The cybercriminals' attacks increasingly better organized as the financial rewards are great and cause severe direct and indirect losses for individuals and corporations. The prevention of cybercriminals' attacks is hampered by polymorphism and packing, the techniques used by the criminals to evade detection. Static analysis detects known malware with great accuracy but fails to overcome the evasive techniques of polymorphism and packing. Dynamic analysis is effective against the evasion techniques but shows a high frequency of false positive results. Image processing detection overcomes the limitations of the evasion techniques. Its additional advantage is that it does not require domain knowledge for implementation. Grayscale malware image classification has been extensively studied and its effectiveness in classifying malware to their corresponding families and differentiate them from benign files has been proved.

This paper has introduced a novel framework, MSIC, that classifies the malicious files to their corresponding families and distinguishes them from benign files. The evaluation experiments have proved the effectiveness of the proposed solution and showed its performance to be better than that of the grayscale framework used extensively in the past research. MSIC's highest F-measure and accuracy results in classifying malware files to their corresponding families were 91.6% and 92.8% respectively as compared to 90.6% and 92.3% for grayscale image classification. Further, MSIC scored 96% on F-measure and accuracy in distinguishing malicious from benign files as compared to 95.5% achieved with the grayscale solution. MSIC also proved to be faster than grayscale framework in processing for file conversion and resizing. This paper also offered the cybersecurity community a publicly labeled malware dataset that has been privately compiled in our labs.

There are four possible directions in which this research may proceed. First, the evaluation of new parameters during the building process of the classifier. Second, deployment of different sampling rates and bit depth for mutual comparison of accuracy and loss results. Third, the evaluation of techniques other than resizing, such as zero-padding for small file sizes and converting a specific number of bytes for large file sizes. The latter would help in reducing the requirement of resources since not all the bits of the file are converted. Fourth, the study of the effectiveness of the proposed solution in other malware areas, especially in malware authorship analysis [43].

REFERENCES

[1] S. S. S. Davis, R. Sipahimalani, F. Hoppe, W. Lee, I. M. Girona, C. Choi, and W. Smittinet. (Feb. 2, 2019). *e-Conomy SEA 2019: Swipe up and to the Right: Southeast Asia's 100 Billion Internet Economy*. [Online]. Available: https://www.thinkwithgoogle.com/intl/en-apac/tools-resources/research-studies/e-conomy-sea-2019-swipe-up-and-to-the-right-southeast-asias-100-billion-internet-economy/

[2] Accenture. (2019). *The Cost Of Cybercrime*. Accessed: Feb. 2, 2020. [Online]. Available: https://www.accenture.com/_acnmedia/pdf-96/accenture-2019-cost-of-cybercrime-study-final.pdf

[3] A. Azab, M. Alazab, and M. Aiash, "Machine learning based botnet identification traffic," in *Proc. IEEE Trustcom/BigDataSE/ISPA*, Aug. 2016, pp. 1788–1794.

[4] McAfee. (2018). *McAfee Lab Threat Report*. Accessed: Feb. 2, 2020. [Online]. Available: https://www.mcafee.com/enterprise/en-us/assets/reports/rp-quarterly-threats-dec-2018.pdf

[5] B. Kolosnjaji, A. Zarras, G. Webster, and C. Eckert, *Deep Learning for Classification of Malware System Call Sequences*. Cham, Switzerland: Springer, 2016, pp. 137–149.

[6] A. Azab, R. Layton, M. Alazab, and J. Oliver, "Mining malware to detect variants," in *Proc. 5th Cybercrime Trustworthy Comput. Conf.*, Nov. 2014, pp. 44–53.

[7] Symantec. (2018). *Internet Threat Report*. Accessed: Feb. 2, 2020. [Online]. Available: https://www.symantec.com/content/dam/symantec/docs/reports/istr-23-2018-en.pdf

[8] F. IT. (2014). *Tilon/SpyEye2 Intelligence Report*. Accessed: Feb. 2, 2020. [Online]. Available: https://foxitsecurity.files.wordpress.com/2014/02/spyeye2_tilon_20140225.pdf

[9] M. G. Schultz, E. Eskin, F. Zadok, and S. J. Stolfo, "Data mining methods for detection of new malicious executables," in *Proc. IEEE Symp. Secur. Privacy. S&P*, May 2001, pp. 38–49.

[10] J. Z. Kolter and M. A. Maloof, "Learning to detect malicious executables in the wild," in *Proc. ACM SIGKDD Int. Conf. Knowl. Discovery Data Mining KDD*, 2004, pp. 470–478, doi: 10.1145/1014052.1014105.

[11] T. Singh, *Support Vector Machines and Metamorphic Malware Detection*. San Jose, CA, USa: San Jose State Univ., 2015.

[12] M. K. Shankarapani, S. Ramamoorthy, R. S. Movva, and S. Mukkamala, "Malware detection using assembly and API call sequences," *J. Comput. Virology*, vol. 7, no. 2, pp. 107–119, May 2011.

[13] E. Raff, J. Barker, J. Sylvester, R. Brandon, B. Catanzaro, and C. K. Nicholas, "Malware detection by eating a whole exe," in *Proc. Workshops 32nd AAAI Conf. Artif. Intell.*, 2018, pp. 268–276.

[14] M. Ijaz, M. H. Durad, and M. Ismail, "Static and dynamic malware analysis using machine learning," in *Proc. 16th Int. Bhurban Conf. Appl. Sci. Technol. (IBCAST)*, Jan. 2019, pp. 793–806.

[15] E. Raff, R. Zak, R. Cox, J. Sylvester, P. Yacci, R. Ward, A. Tracy, M. McLean, and C. Nicholas, "An investigation of byte n-gram features for malware classification," *J. Comput. Virol. Hacking Techn.*, vol. 14, no. 1, pp. 1–20, Feb. 2018.

[16] M. Eskandari, Z. Khorshidpur, and S. Hashemi, "To incorporate sequential dynamic features in malware detection engines," in *Proc. Eur. Intell. Secur. Informat. Conf.*, Aug. 2012, pp. 46–52.

[17] S. Tobiyama, Y. Yamaguchi, H. Shimada, T. Ikuse, and T. Yagi, "Malware detection with deep neural network using process behavior," in *Proc. IEEE 40th Annu. Comput. Softw. Appl. Conf. (COMPSAC)*, Jun. 2016, pp. 577–582.

[18] T. Shibahara, T. Yagi, M. Akiyama, D. Chiba, and T. Yada, "Efficient dynamic malware analysis based on network behavior using deep learning," in *Proc. IEEE Global Commun. Conf. (GLOBECOM)*, Dec. 2016, pp. 1–7.

[19] B. Kolosnjaji, A. Zarras, G. Webster, and C. Eckert, "Deep learning for classification of malware system call sequences," in *Advances in Artificial Intelligence*. Cham, Switzerland: Springer, 2016, pp. 137–149.

[20] A. Damodaran, F. D. Troia, C. A. Visaggio, T. H. Austin, and M. Stamp, "A comparison of static, dynamic, and hybrid analysis for malware detection," *J. Comput. Virol. Hacking Techn.*, vol. 13, no. 1, pp. 1–12, Feb. 2017.

[21] L. Nataraj, "A Signal Processing Approach To Malware Analysis," Ph.D. dissertation, Elect. Comput. Eng., Univ. California, Oakland, CA, USA, 2015.

[22] L. Nataraj, D. Kirat, B. Manjunath, and G. Vigna, "Sarvam: Search and retrieval of malware," in *Proc. Annu. Comput. Secur. Conf. (ACSAC) Worshop Next Gener. Malware Attacks Defense (NGMAD)*, 2013, pp. 1–9.

[23] L. Nataraj, V. Yegneswaran, P. Porras, and J. Zhang, "A comparative assessment of malware classification using binary texture analysis and dynamic analysis," in *Proc. 4th ACM Workshop Secur. Artif. Intell. AISec*, 2011, pp. 21–30, doi: 10.1145/2046684.2046689.

[24] L. Nataraj, S. Karthikeyan, G. Jacob, and B. S. Manjunath, "Malware images: Visualization and automatic classification," in *Proc. 8th Int. Symp. Visualizat. Cyber Secur. VizSec*, 2011, pp. 1–7, doi: 10.1145/2016904.2016908.

[25] R. U. Khan, X. Zhang, and R. Kumar, "Analysis of resNet and GoogleNet models for malware detection," *J. Comput. Virol. Hacking Techn.*, vol. 15, no. 1, pp. 29–37, Mar. 2019.

[26] R. U. Khan, X. Zhang, R. Kumar, and E. O. Aboagye, "Evaluating the performance of ResNet model based on image recognition," presented at the Proc. Int. Conf. Comput. Artif. Intell., Chengdu, China, Mar. 2018, doi: 10.1145/3194452.3194461.

[27] R. Kumar, Z. Xiaosong, R. U. Khan, I. Ahad, and J. Kumar, "Malicious code detection based on image processing using deep learning," in *Proc. Int. Conf. Comput. Artif. Intell. ICCAI*, 2018, pp. 81–85, doi: 10.1145/3194452.3194459.

[28] R. Vinayakumar, M. Alazab, K. P. Soman, P. Poornachandran, and S. Venkatraman, "Robust intelligent malware detection using deep learning," *IEEE Access*, vol. 7, pp. 46717–46738, 2019.

[29] D. Vasan, M. Alazab, S. Wassan, B. Safaei, and Q. Zheng, "Image-based malware classification using ensemble of CNN architectures (IMCEC)," *Comput. Secur.*, vol. 92, May 2020, Art. no. 101748.

[30] D. Vasan, M. Alazab, S. Wassan, H. Naeem, B. Safaei, and Q. Zheng, "IMCFN: Image-based malware classification using fine-tuned convolutional neural network architecture," *Comput. Netw.*, vol. 171, Apr. 2020, Art. no. 107138.

[31] A. M. Badshah, J. Ahmad, N. Rahim, and S. W. Baik, "Speech emotion recognition from spectrograms with deep convolutional neural network," in *Proc. Int. Conf. Platform Technol. Service (PlatCon)*, Feb. 2017, pp. 1–5.

[32] J. Dennis, H. D. Tran, and H. Li, "Spectrogram image feature for sound event classification in mismatched conditions," *IEEE Signal Process. Lett.*, vol. 18, no. 2, pp. 130–133, Feb. 2011.

[33] Q. Mao, M. Dong, Z. Huang, and Y. Zhan, "Learning salient features for speech emotion recognition using convolutional neural networks," *IEEE Trans. Multimedia*, vol. 16, no. 8, pp. 2203–2213, Dec. 2014.
[34] D. Yu, M. L. Seltzer, J. Li, J.-T. Huang, and F. Seide, "Feature learning in deep neural networks–studies on speech recognition tasks," 2013, *arXiv:1301.3605*. [Online]. Available: http://arxiv.org/abs/1301.3605
[35] M. Kalash, M. Rochan, N. Mohammed, N. D. B. Bruce, Y. Wang, and F. Iqbal, "Malware classification with deep convolutional neural networks," in *Proc. 9th IFIP Int. Conf. New Technol., Mobility Secur. (NTMS)*, Feb. 2018, pp. 1–5.
[36] O. Suciu, S. E. Coull, and J. Johns, "Exploring adversarial examples in malware detection," in *Proc. IEEE Secur. Privacy Workshops (SPW)*, May 2019, pp. 8–14.
[37] A. Azab, O. Maruatona, and P. Watters, "AVOCAD: Adaptive terrorist comms surveillance and interception using machine learning," in *Proc. 18th IEEE Int. Conf. Trust, Secur. Privacy Comput. Commun./13th IEEE Int. Conf. Big Data Sci. Eng. (TrustCom/BigDataSE)*, Aug. 2019, pp. 85–94.
[38] J. Fu, J. Xue, Y. Wang, Z. Liu, and C. Shan, "Malware visualization for fine-grained classification," *IEEE Access*, vol. 6, pp. 14510–14523, 2018.
[39] M. Sebastián, R. Rivera, P. Kotzias, and J. Caballero, "AVclass: A tool for massive Malware labeling," in *Research in Attacks, Intrusions, and Defenses*. Cham, Switzerland: Springer, 2016, pp. 230–253.
[40] A. Azab and M. Khasawneh. *Malware Dataset*. Accessed: May 1, 2020. [Online]. Available: http://doi.org/10.6084/m9.figshare.12034569
[41] P. Podder, T. Zaman Khan, M. Haque Khan, and M. Muktadir Rahman, "Comparative performance analysis of Hamming, hanning and blackman window," *Int. J. Comput. Appl.*, vol. 96, no. 18, pp. 1–7, Jun. 2014.
[42] A. Azab and M. Khasawneh. *Malicious-Benign Dataset*. Accessed: May 1, 2020. [Online]. Available: http://doi.org/10.6084/m9.figshare.12084915
[43] R. Layton and A. Azab, "Authorship analysis of the zeus botnet source code," in *Proc. 5th Cybercrime Trustworthy Comput. Conf.*, Nov. 2014, pp. 38–43.

AHMAD AZAB (Member, IEEE) received the Ph.D. degree in information technology from the School of Science, Information Technology and Engineering, Federation University of Australia. He is currently an Assistant Professor with the College of Engineering, Computer Engineering Department, American University of the Middle East (AUM), Kuwait. He works closely with academia and industry on many projects. He has more than ten years of academic and industrial experience. He has worked on planning, implementing, and auditing cybersecurity solutions for various projects in Australia and the Middle East. His research interests include cybersecurity, network analysis, artificial intelligence, and digital forensics. He has published journal articles and conference papers in the area of his research interest.

MAHMOUD KHASAWNEH (Member, IEEE) received the bachelor's degree in computer engineering from the Jordan University of Science and Technology (JUST), in 2010, and the M.Sc. and Ph.D. degrees in electrical and computer engineering from Concordia University, Canada, in 2012 and 2017, respectively. He is currently an Assistant Professor with the Department of Computer Engineering, American University of the Middle East, Kuwait. He has published many journal articles, conference papers, and a book chapter. His current research interests include wireless networks including security, authentication, and route management.