IEEE Access
Multidisciplinary | Rapid Review | Open Access Journal

Received December 2, 2019, accepted December 26, 2019, date of publication January 20, 2020, date of current version January 30, 2020.

Digital Object Identifier 10.1109/ACCESS.2020.2967746

The Image Game: Exploit Kit Detection Based on Recursive Convolutional Neural Networks

SUYEON YOO[1,2], SUNGJIN KIM[3], AND BRENT BYUNGHOON KANG[1,2], (Member, IEEE)
[1]Graduate School of Information Security, Korea Advanced Institute of Science and Technology, Daejeon 34141, South Korea
[2]School of Computing, Korea Advanced Institute of Science and Technology, Daejeon 34141, South Korea
[3]Department of Intelligent System Engineering, Cheju Halla University, Jeju-si 63092, South Korea

Corresponding author: Brent Byunghoon Kang (brentkang@kaist.ac.kr)

This work was supported in part by the National Research Foundation of Korea under Grant NRF-2017R1A2B3006360, in part by the Institute for Information and Communications Technology Promotion under Grant IITP-2017-0-01889, and in part by the Office of Naval Research under Grant N00014-18-1-2661.

ABSTRACT Malware has been installed through drive-by downloads via exploit kit attacks. However, the prior signature- or dynamic-based detection approach to the continuously increasing number of suspicious samples is time-consuming. In such circumstances, convolutional neural networks (ConvNets) can help in rapid detection owing to their direct image-feature generation using exploit codes. However, the general ConvNet model entails the vanishing gradient problem, where the features used for a deep learning-based detection method will become less effective as the network is deepened to improve detection accuracy. In this paper, we propose a multiclass ConvNet model to classify exploit kits, where we adopt various image processing techniques and adjust the size and other parameters of images. The proposed ConvNet model recursively updates images and is designed for fully preserving image properties. This model updates the output of feature maps and pooling using an original image. This model was tested using 36,863 real-world datasets, achieving a 98.2% accuracy in exploit kit detection and family classification. Most importantly, the proposed model is 38 times faster than previous machine learning models, and training time is reduced by 77.8% when compared with prior well-known ConvNet models.

INDEX TERMS Exploit kit, image processing, ConvNet, image classification.

I. INTRODUCTION

Most malware delivered via the Web is contaminated by attack toolkits referred to as exploit kits (EKs). An EK is defined as ''an off-the-shelf software package containing easy-to-use attacks against known and unknown vulnerabilities'' by McAfee [1], [2]. GrandCrab ransomware (built for the monetary profit of the attackers), which is being extensively transmitted in recent years, is spread via EK attacks.

EKs have been widely used in malware propagation via email attachments. Malicious emails are combined with malicious documents that contain EKs or/and links that are redirected to EKs. In particular, the recent distribution of coin-miner, ransomware, trojans, and bankers has increased considerably via EK attacks. Moreover, antivirus products provide low detection rates for most EKs. Therefore, there are limited solutions to prevent malware dissemination. Hence, users who depend on antivirus products are exposed to contamination risks owing to detection difficulties.

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

Regarding security response, previous studies have focused on machine learning-based detection [3] or call-graph-based detection [4] to protect users from malware. These methods have low performance. The prior EK detection based on machine learning showed a high trade-off owing to the extensive computation required for feature extraction (i.e., Prophiler [3] consumed 3.3 s/file.) Recent image-based machine learning models [5], [6] experience limitations in classification because of adversarial attacks caused by adding garbage values. Furthermore, attackers neutralize malware to benign via packers. At present, this disablement is generalized.

Under this circumstance, convolutional neural networks (ConvNets) provide high accuracy and performance in image classification based on text strings. Based on this, we implement an image-based EK classification model using a new deep learning approach. The benefits of this model are fast performance and high detection rate.

EK is generally described using obfuscated script code. This property is more suitable for an image-based detection because the obfuscation can be distinguished from benign

files without attempting to interpret the code. The property is evident in the image. In particular, in techniques that avoid detection using noise, current EK exhibits limited performance compared to malware because EK is programmed in a limited format such as HTML and JavaScript. The image is a preferable medium that can be used to block malware contamination propagated via the Internet. In this study, we built the image-based detection model for the following additional reasons:

ConvNets are known to provide 1) high accuracy in image classification (as image similarity matching) and 2) high performance via the imaging of direct code scripts. However, the detection rate of ConvNets is sensitive to changes in the depth, size, and hyperparameters of models, as previously known. Additionally, the detection rate is influenced by color (i.e., texture, color, and brightness) and image complexity. Thus, in this study, we build a grayscale-based hybrid ConvNet model to overcome these sensitivities. The main contributions of this study are as follows:

- We propose a new ConvNet model, i.e., a recursive convolutional neural network (RCNN) model, which provides an enhanced image classification.
- We demonstrate that our model is 38.31 times faster than previous models. Additionally, its detection rate is extremely high (more than 98.2%) compared to previous models.
- The RCNN can be used for the detection of various malicious attributes and for other types of image classification.

The remainder of this paper is organized as follows: Section II presents related work and describes our research background. Section III provides an overview of various features and classifiers for our model setup. In Section IV, we explain the core system framework and the technical details of our design. Then, we describe the dataset used, the experimental setup, and our experimental results in Section V. The limitations of our model are discussed in Section VI, and the conclusions are outlined in Section VII.

II. BACKGROUND AND RELATED WORK

A. EXPLOIT KITS

The CVE-2018-4878 IE 0-day vulnerability emerged in 2018, and it was deployed in RIG, Magnitude, GF Sundown, KaiXin, Underminer, and Pseudo EK. Malware has been globally distributed via these EKs. Currently, these EKs are connected to links hidden in attached files and email URLs, and numerous ransomware attacks are distributed via these EK attachments. Additionally, a new EK, i.e., fallout, was found at the end of August 2018. This EK is mainly hidden in the obfuscated JavaScript of webpages [7], [8]. If internet users access malicious websites, hidden redirect URLs deliver user access points to a webserver that is already deployed to these EKs and user PCs are often compromised. That is, attacks via EKs consist of various malicious URLs and brokers, such as emails and websites as intermediaries for facilitating contamination.

To detect Malicious URL that leads to an exploit kit site, Eshete *et al.* [9] and Kim *et al.* [10] leveraged the abnormal behavior of EKs. WebWinnow [11] analyzed workflow of exploit kits and extracted features from their attack-centric and self-defense behavior. Kim *et al.* [10] focused on attackers' habitual URL manipulation behavior and then employed similarity matching to classify suspicious URLs that have a similar character array.

The other efforts are to discriminate malicious Webpages and JavaScript code that host drive-by download exploits [3], [12]–[17]. Prophiler [3] is a lightweight machine-learning filter for malicious Web pages. They extracted features from HTML, JavaScript, URL, and DNS records. The purpose of Prophiler is to quickly filter non-malicious pages so it allows higher false positive rates than other detection model, Zozzle [12], which automatically extracts hierarchical features from the JavaScript abstract syntax tree. Recently published FriSM [15] enhanced string-similarity features to detect variants derived from existing EKs. Xu *et al.* [13] presented a combination of two detection models: one is based on application layer traffic information and the other on network-layer traffic information. To complete cross-layer detection, they drew the final decision via OR-, AND- and XOR-aggregation. Yoo *et al.* [14] presented a two-phase malicious Web page detection model using the misuse and anomaly detection approaches hierarchically to overcome drawbacks of both approaches. Wang *et al.* [16] applied deep learning based malicious JaveScript code detection. Their detection system consists of random projection, Stacked denoising auto-encoders, which learn text-type features with unsupervised pre-training, and logistic regression layer. KIZZLE [17] can generate features of exploit kits by analyzing unpacked binary malware code and using fewer variants in metamorphic malwares caused by a code reuse.

Although previous EKs detection approaches show a high detection accuracy, the time-consuming feature analysis and relatively the high training time are inevitable shortcomings. Furthermore, code deobfuscation or unpacking malware is a prerequisite to extract features.

B. VISUALIZATION

There has been a continuous effort towards utilizing visualization approaches for malware analysis and detection because visualization allows the detection model to find features from different perspectives such as texture, color, and graph structure. Yoo [18] used self-organizing maps (SOMs) to visualize malicious code in an executable file. Using SOMs, the author showed the difference between the formats of a virus-infected file and a normal executable file caused by inserting virus code into the original code by force. Trinius *et al.* [19] showed the distributions of API calls obtained after malware behavior analysis using treemaps and the sequence of API calls according to an operation section using thread graphs. Han *et al.* [20] proposed a visualization method using opcode sequences.

Whereas prior studies focused on visualizing malware features, studies [21]–[26] that consider malware code as a digital image has begun to emerge. Nataraj *et al.* [21], [22] adopted an image processing technique: they converted malware binaries into gray-scale images then extracted pattern features from the images using GIST [27]. After that, they used a k-nearest neighbor algorithm to classify malware. Xiaofang *et al.* [23] visualized malware as gray images and extracted image features with a speeded-up robust features algorithm. The research of Liu and Wang [24] also focused on the gray image, and the local mean method was used to reduce the image size to speed up the ensemble learning process. Han *et al.* [25] proposed a malware visualization method and an entropy graph generated based on the gray image was used to realize automatic analysis. However, this method cannot be applied to packed malware, because the entropy of packed malware are usually very high and cannot indicate any specific pattern. Recently, Fu *et al.* [26] focus on converting malware binaries into RGB-colored images to extract low dimensional features that lead to reduce the model complexity.

C. CONVNET

Visualization approaches have been applied to support malware analysis and detection for several years. However, there are limitations in using a visualization approach to detect malware. Most of the previous detection models in this approach require the structure of malware but the structure of non-PE files is difficult to get. The other limitation is the poor performance when malware is packed or encrypted.

On the other hand, even though EK script data are more appropriate for building ConvNet models using EK scripts converted to images, to the best of our knowledge, there are no previous studies that have addressed the detection of EKs through visualization.

A ConvNet is a neural network model that uses convolution layers mainly for image recognition or image classification. An input image passes through convolution layers with filters and pooling several times. Then, it passes through fully connected layers and the softmax function to predict an image object with values ranging from [0,1]. This CNN-based malware detection model [28]–[30] discriminates malware via deep layers for classifying pixels after changing the binary sections of malware into grayscale images. This model provides a high detection rate; however, it has limitations in experiments on malware detection with packers. Other similar studies [31], [32] have shown 96.2% classification accuracy with Microsoft malware dataset [33] and 96% accuracy with KAIST Cyber Security Research Center [34] dataset.

The drawback of image-based detection is that it cannot classify malware when it is packed by packers. That is, malware images show the same/similar grayscale images. On the contrary, an image-based approach is suitable for script-based EK detection. Previous studies have typically suggested three models that analyze the EKs used as vectors for malware delivery: a feature-based ML model, static analysis, and dynamic analysis. In this work, excluding the prior trials, we performed newly EK detection with another image processing approach referred to as a CNN.

FIGURE 1. Example of `Gondad` exploit kit changed to grayscale.

III. MODEL SETUP

We built an optimized image-based model via the following four approaches to increase the detection rate of EKs: color, size, classifier optimization, and a hybrid model. This section presents the details of the model.

A. COLOR

In general, the type of color affects the detection rate of a CNN model. Hence, when loading images, we normally change the image type to grayscale. However, there is a broad range of color types such as grayscale, RGB, and gamma. Figure 1 shows an example when a Gondad EK is changed to grayscale. The obfuscated code is black, and the initial script code is changed to dark gray and black. The image exposes the entire contour of an EK.

EKs contain various exploit codes that attack various application vulnerabilities; for instance, Java, Flash Player, plugins, and web browser vulnerabilities. These EKs attack web access users, obtain system privileges and subsequently distribute malware. There are several dozens of EK types, such as RIG, Angler, Blackhole, and the recent Spelevo. Particularly, all types of EKs have different code schemes. For instance, an image of Angler is different from that of RIG. There are several types of Anglers. Each type has a different image, but a similar variant image. Figure 2 shows two types of Angler and their variants. The images among the same variant are quite similar, but those of different variants are different.

We require an optimized image reproduced by image transformation to accurately detect EKs. In this approach, we generated five different image types from the same image (see Figs. 3 and 4). In the experiment using transformed images, we examined the change in detection rate. The model for this experiment was tested with 2 convolution layers, 2 pooling layers, the ReLU activation function, the Adam optimizer, and 24 epochs.

As shown in Figure 5, to improve the color-based classification accuracy, we measured classification accuracy for an image that was transformed to 8-bit grayscale, RGB, gamma 1.5, gamma 0.5 and limited grayscale. In this experiment, the detection rates were 99.86%, 99.77%, 99.68%, 99.83%, and 99.91%. In this test, the limited grayscale image showed

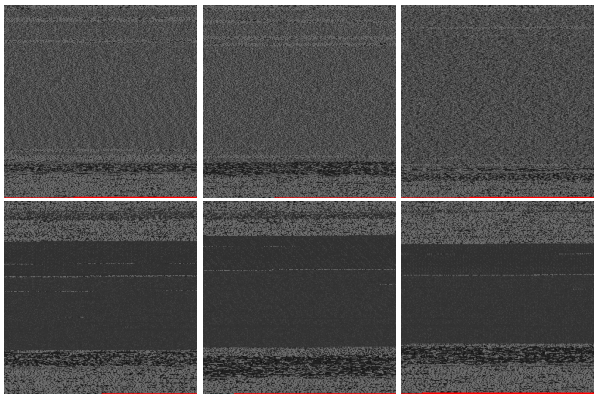

FIGURE 2. Two types of Angler variants projected to grayscale; these variants were detected in 2014, 2015, and 2016. In our datasets, there were nine different types of Angler exploit kits. Here, we present two types among them.

FIGURE 3. Examples transformed to different images. From left to right: RGB, gamma 0.5, gamma 1.5, grayscale, and scale limited.

FIGURE 4. Image comparison in RGB, grayscale, and limited grayscale.

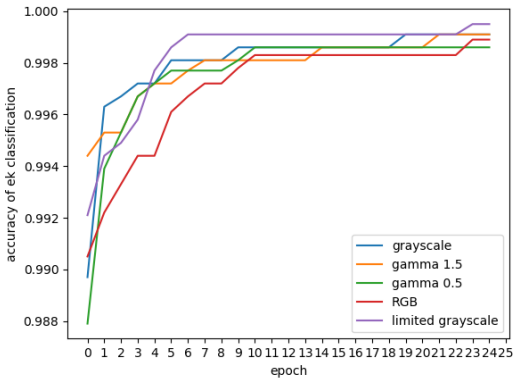

FIGURE 5. This output was the result obtained using each different image, as seen in Figure 3.

the highest accuracy of 99.91%. We adopted the limited grayscale image for the proposed model.

Limited grayscale reduces the original gray scope by applying limited scope. Thus, we varied the value from 0 to 10 to further highlight the image. In image transformation, we allocated one value from the abovementioned range corresponding to each image pixel. For instance, when the range of pixels was 0–255, a pixel value of 0–25 was allocated to 0 and that of 26–50 was assigned to 2. This approach provided higher classification accuracy compared to typical grayscale. Precision, recall, and F1-score was 99.91%.

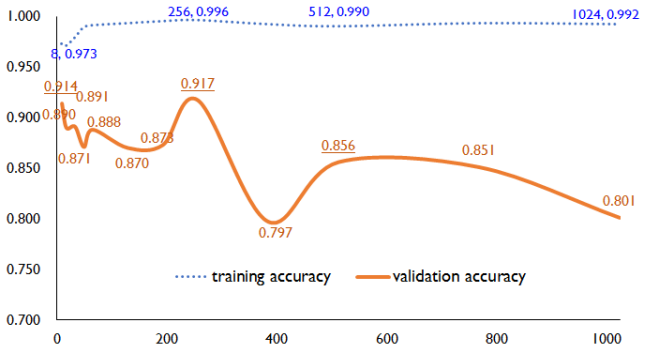

FIGURE 6. Changes in classification rate with size. We used 17593 train images, 3350 validation images, and 2326 test images. These images were resized from 8×8 to 1024×1024 pixels. The results were average values received from three iterations of a normal CNN model.

The overall performance including image transformation and prediction was about 0.08 s per file.

B. SIZE

In general, the original image sizes in MNIST are fixed. However, in the real world, the sizes of EKs are not fixed. There are a variety of sizes, and benign sizes have a wide range. Hence, we considered various EK image sizes in real-world circumstances.

Most image sizes of EKs range from 4×4 to 1453×1453 pixels. However, we must resize original EKs to utilize images in CNN models. The resizing of images is one of the critical factors that affect the detection rate. Hence, to determine the best image size in an EK, we compared classification accuracy for different image sizes. We resized images to a fixed size, such as 8×8. In this experiment, we used pixels between 8 and 1024. Figure 6 illustrates that 256×256 pixels is the best optimized size. We used a standard CNN model with 2 convolution layers, the ReLU activation function, max pooling, same padding[1], 10 epochs, a batch size of 32, 10 channels, 64 hidden units (which are used in the dense layer), a filter size of 3×3, a dropout rate of 0.25, and the softmax regression.

Figure 6 shows the classification accuracies for different image sizes. This figure shows the best image size according to accuracy. In light of these results, the variation in classification accuracy with size is based on the characteristics of color, texture, and brightness. These features are different for every image. Therefore, the features of an image can be expressed by its size. We identified that classification accuracy peaked at image sizes of 8×8, 256×256, and 512×512 pixels and decreased significantly after these peaks. Based on this trend, we clustered images into three size ranges, i.e., 8–255, 256–511, and 512 and above, and used these sizes to adapt our CNN models. That is, we require different classification models based on the image sizes, such as the hybrid model (evaluation results in this approach are detailed in Section IV and Section V).

[1]To produce an output of the same size as the input, we use zero padding outside the edges to create images with the same shape.

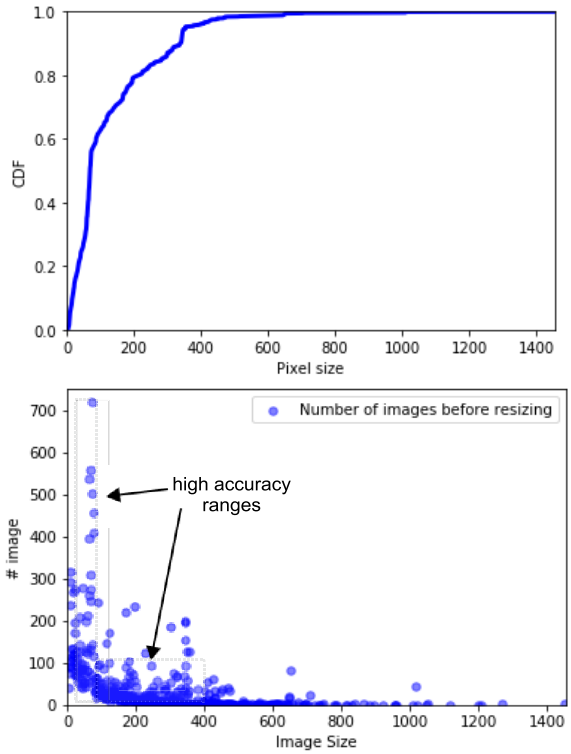

FIGURE 7. CDF according to pixel sizes (upper), and the number of images according to original image size (lower). The *x* axis label indicates size by size. For instance, 200 represents an image size of 200 × 200 pixels.

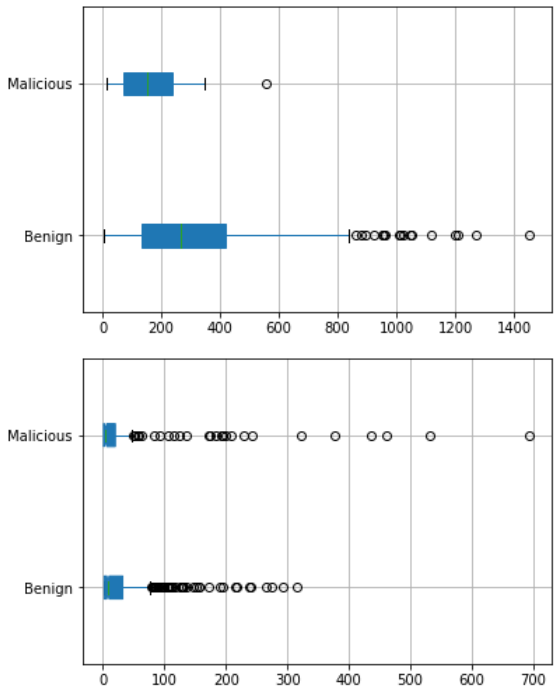

FIGURE 8. Sizes of images range from 4 to 1453 pixels in benign samples and from 13 to 558 in malicious samples (upper). Based on each image size, the number of original images range from 1 to 316 in benign samples and 1 to 694 in malicious samples (lower). The benign images are smaller and the number of malicious samples in each image size is broadly distributed in our dataset.

As shown in Figure 7, detection rates were obtained after resizing the original images to specific sizes. Particularly, the CDF in Figure 7 shows that the cumulative number of files in the size ranges of 8–512, 256–511, and more than 512 are 26.4%, 68.2%, and 80.3% of the total files, respectively. The number of images (lower) in Figure 7 influences the detection rate. Central areas in this number of images may affect high detection rates. The images were broadly distributed according to the *y* axis at these points. The central points along the *x* axis between 1 and 500 image sizes are at the bottom. From this figure, we can assume that a high detection rate is related to specific image sizes and the number of images trained.

Figure 8 illustrates the outliers for the mean and standard deviation. The sizes of benign images in the outliers can be easily detected compared to the sizes of malicious images (upper). The lower figure shows that the numbers of malicious images between 300 and 700 can be distinguished from those of benign images. The malicious images in this range can be more precisely trained in our model.

C. RCNN

In this section, we introduce an RCNN model to increase classification accuracy. This model is based on a normal CNN model. However, the distinguishing characteristic from a general CNN is that the RCNN model recursively updates the original input image. The basic concept is to multiply the **feature maps** created through a convolution layer with **the original image** and to update the feature maps. All feature maps are updated; the size of a map is equal to that of the input image. The details of the RCNN model are as follows:

1) PSEUDO CODE

To accurately classify a given image, we used approaches related to image update in feature maps and pooling. These image update algorithms are utilized for removing the ''blur'' phenomenon, in which image features are diminished.

Algorithm 1 is an implementation that uses the same size matrix between an image and a feature map via *SAME* padding. Hence, the code can be different if padding is not applied. In Algorithm 1, an image update after the Conv layer is based on the element-wise multiplication of two matrices; one is an $M \times M$ input matrix and the other is an $M \times M$ conv1 matrix, which is the output matrix of the first convolution layer operation after adding the *SAME* padding option. In the case of Algorithm 1, we set $M = 50$, b is the number of input images, and c is the number of nodes (i.e., channels) of the first convolution layer. When $i = \{0, 1,..., b–1\}$ and $j = \{0, 1,..., c–1\}$, we extract the $M \times M$ conv1 matrix of the j-th channel in the i-th image. Next, we reshape the conv1 matrix to a $1 \times M^2$ matrix (flattening step) and the i-th original input image to a $1 \times M^2$ matrix in the same manner. We conduct element-wise multiplication of these two flattened matrices. Then, the output of multiplication is reshaped into an $M \times M$ matrix to be fed to the next convolution or pooling layer. The process described above is performed for all values of i and j.

Algorithm 1 Image Update Algorithm on Feature Maps

Input: output matrix via a Conv layer (Conv1)
Output: updated *Conv1*

$F \leftarrow$ feature maps
$\_X \leftarrow$ original images
$B \leftarrow$ a batch size
$C \leftarrow$ # channel

Conv1 = ConvImageUpdate(F, $\_X$, B, C)

function ConvImageUpdate(f, $\_x$, b, c)
  UpdateFeatureMap $\leftarrow$ *tf.zeros*([0, 50, 50, c])
  for $i = 0;\ i \leq b - 1;\ i$++ **do**
    conv_row $\leftarrow$ *tf.zeros*([1, 50, 50, 0])
    for $j = 0;\ j \leq c - 1;\ j$++ **do**
      conv_1 $\leftarrow f[i, :, :, j]$
      conv_1 $\leftarrow$ *reshape conv_1 to vector*[2500])
      *img_* $\leftarrow$ *_x*$[i, ;, ;, :, 0]$
      img_1 $\leftarrow$ *reshape img_ to vector*[2500]
      conv_1 $\leftarrow$ *multiply conv_1 and img_1*
      conv_1 $\leftarrow$ *rearray conv_1 to* [1, 50, 50, 1]
      conv_row $\leftarrow$ *concat conv_1*
    end for
    UpdateFeatureMap $\leftarrow$ *concat conv_row*
  end for
  return *UpdateFeatureMap*
end function

Algorithm 2 Image Update Algorithm on Max Pooling

Input: output matrix via a max pooling
Output: updated *pooling*

$M \leftarrow$ max pooling values
$\_X \leftarrow$ original images
$B \leftarrow$ a batch size
$C \leftarrow$ # channel

MaxPooling = MaxImageUpdate(M, $\_X$, B, C)

function MaxImageUpdate(m, $\_x$, b, c)
  UpdateMaxPool $\leftarrow$ *tf.zeros*([0, 25, 25, c])
  for $i = 0;\ i \leq b - 1;\ i$++ **do**
    max_row $\leftarrow$ *tf.zeros*([1, 25, 25, 0])
    for $j = 0;\ j \leq c - 1;\ j$++ **do**
      max_1 $\leftarrow f[i, :, :, j]$
      max_1 $\leftarrow$ *reshape max_1 to vector*[625])
      *img_* $\leftarrow$ *_x*$[i, ;, ;, :, 0]$
      img_1 $\leftarrow$ *sum of all* $(a_{i,j} + a_{i+1,j} + a_{i,j+1} + a_{i+1,j+1})$
      img_1 $\leftarrow$ *reshape img_ to vector*[625]
      max_1 $\leftarrow$ *multiply max_1 and img_1*
      max_1 $\leftarrow$ *rearray conv_1 to* [1, 25, 25, 1]
      max_row $\leftarrow$ *concat max_1*
    end for
    UpdateMaxPool $\leftarrow$ *concat max_row*
  end for
  return *UpdateMaxPool*
end function

In a nutshell, the feature map update method, used to prevent feature loss in our recursive ConvNet model, is as follows.

To create a feature map, the input image data of the kernel size is sequentially composited with the kernel. Then, input data including kernel size is extracted from the image data in the original image, and the result is multiplied and combined with the corresponding feature map. Then, the final result matrix with the required feature updates is created. This can be expressed using math notation. The general feature map values are calculated[2] and simply stated by the equation $G[m, n] = (i \times h)[m, n]$, where the input image is denoted by i and our kernel is denoted by h. An output matrix comprises the indexes of rows and columns, whose elements are marked as m and n, respectively.

$$G[m, n] = \sum^{j}\sum^{k} h[j, k]i[m + j, n + k] \tag{1}$$

We updated the subsequent feature map with the feature values of the original image, and the formula is stated by $F_U = G \times I$ and detailed as.

$$F_U[m, n] = (\sum^{j}\sum^{k} h[j, k]i[m + j, n + k]) \times i[m, n] \tag{2}$$

[2]Referred `https://towardsdatascience.com/gentle-dive-into-math-behind-{\penalty-\@M}convolutional-neural-networks-79a07dd44cf9`

where F_U means feature update and I denotes the matrix of an input image. F_U (2) works internally in TensorFlow when executing Algorithm 1.

In our model, a max pooling is also updated in a similar manner. The only change is to element-wise multiply the matrix sum of an original image with filter size by the matrix of max-pooling. In Algorithm 2, we sum all i,j elements $(a_{i,j} + a_{i+1,j} + a_{i,j+1} + a_{i+1,j+1})$ and place the results in $a_{i,j}$ of the new image matrix (*img_1* in Algorithm 2). In TensorFlow, we can simply replace the corresponding line of code, *img_1* = (*img_*$[0 :: 2, 0 :: 2]$ + *img_*$[1 :: 2, 0 :: 2]$ + *img_*$[0 :: 2, 1 :: 2]$ + *img_*$[1 :: 2, 1 :: 2]$). Then, the original image matrix size can be halved. For instance, a 50×50 image matrix is reduced to a 25×25 matrix. Hence, we multiply the new image matrix and max pooling output matrix. (Besides, when applying average pooling, we can obtain mean values by dividing by 4 as we use a 2×2 pooling size in this algorithm). Then, the algorithm sums all i, j elements. This algorithm depends on user definitions in the hyperparameter. Thus, the result can be slightly different according to model architecture.

If the order of the code itself is an important factor in detecting exploit code, natural language processing (NLP) may provide a good method of detection. However, in exploit

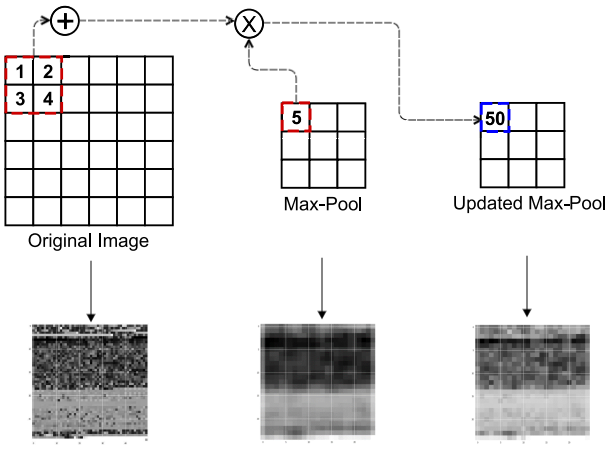

FIGURE 9. Values of max-pool are updated with the resized image of 50×50 pixels. In summary, the values in a 2×2 pixel area in the image are summed, and the output is multiplied element-wise by a value that is mapped with a max pool. This operation is processed until all image pixels are calculated and updated with two strides.

TABLE 1. EK dataset list based on types, and the number of images used in trainset and test.

EK name	#	image size in trainset			image size in test		
		1-63	64-511	512+	1-63	64-511	512+
angler	189	0	152	0	0	37	0
blackhole	402	0	322	0	0	80	0
ck vip	13,445	10,741	16	0	2,684	4	0
gondad	4,520	1,414	2,204	0	352	550	0
magnitude	23	12	8		2	1	0
neutrino	128	104	0	0	24	0	0
nuclear	62	0	50	0	0	12	0
Redkit	598	0	480	0	0	118	0
RIG	369	159	138	0	38	34	0
sundown	12	0	10	0	0	2	0
sweet	1,499	0	1,200	0	0	299	0
benign	15,616	5,588	6,618	288	1,396	1,654	72
Total	36,863						

kit detection, finding obfuscation features is more important than ordering code. Therefore, we applied a CNN model whose features are well represented in the image. In particular, the types of exploit kits are limited to approximately thousands. While the variants of those types are myriad, when converted, the difference from the original image is not noticeable. Thus, there is no difficulty in comparing with transformed images of other benign codes.

2) IMAGE COMPLEXITY

The location of the image update in RCNN is dependent on image complexity. We can use the image update after the Conv output matrix, after the max-pooling output matrix, or after both. Someone can construct the image update randomly between many deep layers. All this is dependent on image complexity and model situations.

Table 1 shows the EK dataset we used in this experiment. Twelve types of EK were used as described in the table. In an experiment to evaluate image complexity, we also adopted Caltech101 datasets. Among the datasets, we selected 12 image types, i.e., airplanes, Google backgrounds, car side, chandelier, faces, easy face, hawksbill, helicopter, ketch, leopards, motorbikes, and watch. The number of images is 3,906. Additionally, we used the dataset of 42,000 MNIST. In this experiment, we divided the number of each train, valid, and test into 7:1:2. According to the image complexity, we assumed that the accuracy and depth of the CNN model are affected. Thus, we need to know the position of image complexity in EK to build the model (i.e., to establish a deep or shallow model).

First, the image entropies of MNIST, EK, and Caltech101 are 1.1885-4.55047, 2.89596-6.60579 (except only four benign samples, where their entropies are 0.263, 0.477, 0.539, and 1.931, respectively), and more than 6.7, respectively. Image complexity is based on Shannon's information theory [35]. In our datasets, the image complexity of EK is located in the middle.

3) CORRELATION BETWEEN IMAGE COMPLEXITY AND CONVNET MODEL DEPTH

Table 2 shows the results of accuracy according to the image complexity and model depth between the basic CNN model and our RCNN.

In a shallow model with a short layer, a basic CNN model was composed of conv → max-pooling → dropout → dense → dropout → dense. In contrast, the RCNN model was built using conv → conv-update → max-pooling → max-update → dropout → dense → dropout → dense. These models had 12 classes, 3×3 filters, 10 channels, 2×2 max pooling size, and used 64 nodes in dense layer. We used a learning rate of 0.001 and a batch size of 16 in this experiment. The same values of the hyperparameters were used in deep models.

In a deep model with a deep layer, a basic model used 3 conv, dropout, 3 conv, dropout, 4 conv, max-pooling, dropout, dense, dropout, and dense in a sequential manner. On the other hand, a deep RCNN model had conv → conv-update 3 times, dropout, and repeated conv → conv-update 3 times again, had dropout, and repeated conv → conv-update 4 times again, had max-pooling, max-update, dropout, dense, dropout, and dense in a sequential manner.

Figure 10 shows the accuracies of both models for various image types. In most cases, the RCNN showed higher accuracy than the general CNN. In particular, the RCNN exhibited high performance in datasets with high image complexities.

This property is more clearly expressed in Figure 11. The degree of image complexity in this figure is the same as MNIST < EK < caltech101. The mean entropy of caltech101 is 7.42. This figure illustrates that image complexity and the depth of the layer are correlated.

As shown in Figure 11, for images with low entropy, such as MNIST and EK, high accuracies are indicated in the initial stage, whereas the accuracies for caltech101 images were low in the initial stage and gradually increased as layers were added. Consequently, low-entropy images use less number of layers, but images with high entropies need deeper layers. RCNN accommodates this rule, and the image complexity in RCNN is proportional to the depth of the layer.

IV. DESIGN

In this section, we introduce a hybrid model based on our RCNN. The proposed model is a multisize

TABLE 2. Classification accuracy of basic CNN and our RCNN based on each different image type. In this experiment, we used 20 epochs. In this experiment, RCNN showed a better performance than a basic CNN model in general. In particular, RCNN exhibited a high performance in a deep model.

Model	Image type	input size	# epoch	shallow model (# conv: 1)		deep model(# conv: 10)	
				Train Accuracy	Valid Accuracy	Train Accuracy	Valid Accuracy
basic CNN	EK data	8	10	0.9480	0.8794	0.4768	0.4986
			20	0.9632	0.8966	0.4278	0.4256
RCNN			10	0.9536	0.8728	0.7642	0.6676
			20	0.9686	0.8818	0.8066	0.7302
basic CNN	EK data	256	10	0.8302	0.7942	0.4462	0.4384
			20	0.889	0.833	0.446	0.4384
RCNN			10	0.9868	0.9016	0.6916	0.6404
			20	0.9948	0.9164	0.6684	0.5646
basic CNN	MNIST	28	10	0.9656	0.9358	0.1110	0.1110
			20	0.9952	0.9534	0.1110	0.1110
RCNN			10	0.9356	0.9026	0.5330	0.5430
			20	0.9856	0.9248	0.7762	0.7522
basic CNN	Caltech101	150	10	0.3350	0.2316	0.1656	0.1580
			20	0.4040	0.2626	0.1630	0.1542
RCNN			10	0.3820	0.2520	0.6472	0.3522
			20	0.4516	0.3064	0.7482	0.3844

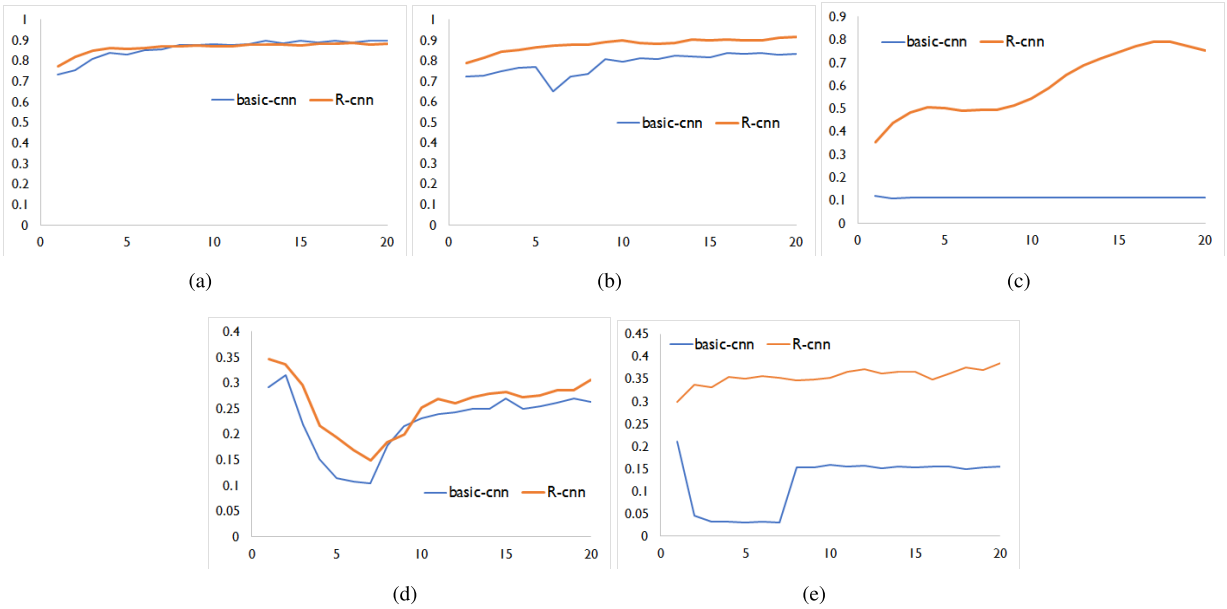

FIGURE 10. Accuracy of ConvNet and RCNN. Experiments were repeated five times and mean values are reported. Results for the shallow model with 20 epochs on the EK datasets with an input size of (a) 8 and (b) 256. (c) Results of the deep model on the MNIST dataset. Results on the Caltech101 dataset for the (d) shallow and (e) deep models.

classification model. The images are classified by size, forwarded to the appropriate RCNN model, and detected by the same model.

This model involves the image-based classification. Without feature extraction, this model computes the weight $(Wx + b)$ of an image of the same size mapped with filters and keeps the weight of image features in (feature) maps with matrices. Then, it reduces the size of the features via pooling. These matrix values are trained using a labeling method called softmax. Specifically, our RCNN appends the features of the original image in the features of all (or some) output matrices in this processing, and supplies more distinct image features. This model generates the final output matrix via the multiplication of two matrices (i.e., the resized image and output matrix).

Figure 12 describes an RCNN classification model in which a hybrid approach is added according to the image size. In a file-size-based multi-RCNN model, each RCNN has different nodes, epochs, batch sizes, and hidden layers, resulting in high accuracy. This model predicts the class of EK through each different model based on size.

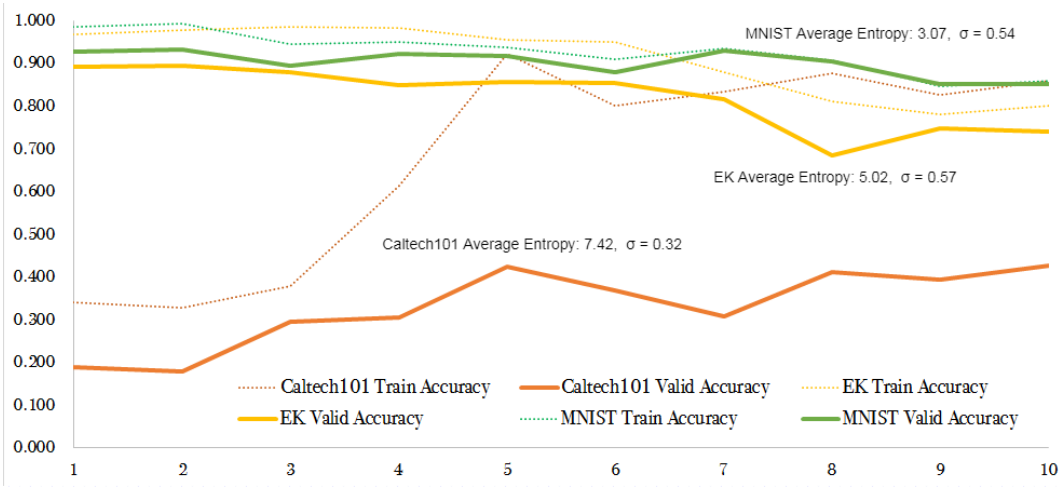

FIGURE 11. Deep model results according to image complexity. The *X*-axis denotes a deep layer. Here, we applied 10 conv layers.

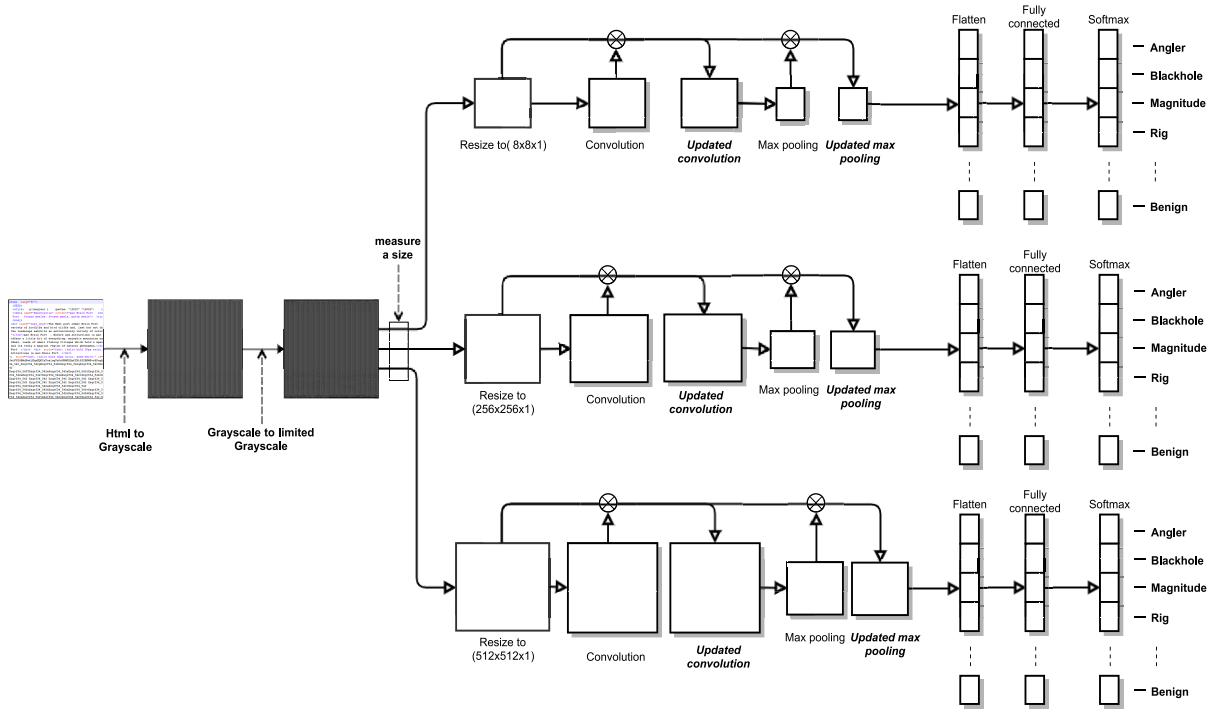

FIGURE 12. Overview of our proposed model.

As shown in Figure 12, an original script code (EK source codes like HTML, JavaScript, PHP, etc.) is first transformed to grayscale. Then, the grayscale image is retransformed to a limited grayscale (with one of 11 values, from 0 to 10). This image is more distinguished. Subsequently, the size of the image is calculated and forwarded to one of the RCNN models according to the size. The input image is resized to a different size after being entered into an RCNN model (i.e., 8×8 pixels). Then, a feature map produced from the 1^{st} Conv layer is updated via matrix multiplication between the feature map and image map. The results are forwarded to max pooling, and then the pooling matrix is updated with the resized image in the same manner. The output matrix is flattened and labeled via a fully connected dense net. In this approach, each RCNN model has different hyper-parameters. Thus, the filter sizes and channels are different.

Each RCNN is optimized based on size. The model provides one-hot encoding between 12 values when classifier distinguishes an EK i in class j. The normalized image size in our multi-classifier can be defined by the following equation:

$$\text{resized image size } i,\ 8 \leq i \leq 512 \qquad \text{for all } i \in \mathbb{R}$$

where i is an image size and R denotes all EKs and benign files with size i. The other sizes might give different results. The EKs are determined as labeled outputs according to the above multisized and multiclass approach. This model covers

TABLE 3. Accuracy comparison between various ConvNet models. In this evaluation, we used 23,327 image datasets including 16,337 train images, 4,663 test images, and 2,327 images for prediction (including 1561 benign images and 766 EK samples), and equally applied 256 input size, 40 epoches, 1 batch size, and Adam optimzer in all image models. These models used 10 classes as exploit kits and benign types. In this experiment, a RCNN showed the highest accuracy at this size.

Model	TP	FP	Accuracy
Vgg19	1561	766	67.1
Vgg16	47	2280	2.0
MobileNetV2	1070	1257	46.0
InceptionV3	1571	756	67.5
Xception	1611	716	69.2
ResNet50	1655	672	71.1
DenseNet121	1206	1121	51.8
Basic RCNN	2301	26	98.9

various EK image forms widely used in malware proliferation and benign files with various image types.

V. EVALUATION

In this section, we evaluated our model, and compared it to prior works.

A. DATASETS AND EXPERIMENTAL SETUP

To demonstrate the performance of the proposed model, we used 15,617 benign files and 17,709 EK files. This dataset was labeled via the VirusTotal and manual inspection. The datasets used for evaluation are detailed in Table 1. The datasets are from *VirusChaser*, an antivirus company. *VirusChaser* provided 35,616 samples in total, including 15,616 benign samples and 20,000 EK samples. The remainder of the dataset comprised 3.4% of the combined dataset from `http://www.malware-traffic-analysis.net/`. The datasets used will be made publicly available.[3]

We ran the experiments on Linux with an Intel® XeonCore® Silver 4116 CPU 2.10 GHz, 48 cores, with 128 GB memory and NVIDIA TITAN V GPU.

This RCNN (or basic CNN) was programmed by using Python 3.5.5, TensorFlow 1.12.0, and Keras 2.2.2 library as the learning framework, and had 330 lines of code, and 228 code lines for data preprocessing were used.

B. ACCURACY AND PERFORMANCE

This classification in various types is a more difficult task. Thus, the classification model should be reliable and accurate for classifying EK types, and the model should satisfy the overall classification. In this section, we verify the accurate classification rate of EK types via classifiers of various CNN models. Then, we determine the optimized model according to the best results.

Table 3 gives the results of EK classification from various CNN models. In this evaluation, we used 2,327 test images, and its results denote the accuracy. Among them, RCNN increased classification accuracy to 98.9%, providing high performance. Other image models also exhibited low accuracy in indicating benign and EK types, where unique image features affected the classification rate, while the deep-layer models did not demonstrate the improved effectiveness of classification.

[3]`https://bit.ly/2R3Bnmu`

TABLE 4. Accuracy result in RCNN. In this evaluation, we used 25,820 training images, 3,680 validation images, and 7,356 prediction images. There are 12 classes. This dataset is even more complicated than that of Table 3.

RCNN	Multi-class hybrid model		
image size	1-63	64-511	512-
input size	8	256	512
epochs	40	40	40
classes	12	12	12
filter size	5	5	5
channels	20	30	10
max k	2	2	2
dropout	1	1	0.8
learning rate	0.001	0.001	0.001
batch size	4	4	4
# test image	4496	2788	72
cost value	0.003	0	0
training accuracy	0.999	1	0.984
validation accuracy	0.944	0.917	1.0
test accuracy	0.982	0.982	1.0

In this experiment, ResNet50 and DenseNet121 showed low accuracy because their models exhibited a relatively high detection rate in a benign dataset but showed a low TP rate in a malicious dataset; thus, we identified that most EKs were classified as benign. These models showed overfitting for this benign training data. In particular, MobileNetV2 exhibited the lowest accuracy. The common point between the models is that they do not properly maintain the image characteristics. Accuracy is denoted by the ratio of the number of correct predictions to the total number of predictions made. In particular, we use macro average precision for multiclass classification. In this experiment, we applied the same hyperparameter configuration in all image nets and RCNN in terms of input size, batch size, # of epoch and applied Adam as an optimizer. In addition, each image model used its default configuration values as the model hyperparameters. In this respect, RCNN exhibited the highest accuracy in this evaluation, as seen in Table 4. RCNN was very stable and exhibited a high performance particularly in terms of training and validation accuracy, as shown in Figure 13 and 14, as compared to other image models.

Table 4 illustrates the accuracy result of RCNN[4] in the classification of EK and benign datasets. In this experiment, we increased the number of classes to 12 and the number of image datasets in the tests, as shown in Table 4. RCNN adopted a filter size of 5 × 5, 40 epochs, and 12 classes. The input size differs according to the original image size. Accordingly, from the accuracy of this experiment, it can be inferred that RCNN depends on image update techniques,

[4]The related datasets and codes that were used in the paper are going to be opened later.

TABLE 5. Performance comparison with state of the art.

	Classifier	Features	Feature Collection Time	Image Change Time	Classification Time	Total Time
Prophier [3]	SVM	HTML, JavaScript, URL, Host-based	3.06 s/page	-	0.237 s/page	3.297 s/page
[36]	Logistic Regression	URL, Hostbased	3.56 s/URL	-	0.020 s/URL	3.580 s/URL
JAST [37]	Random Forest	HTML, JavaScript	0.222 s/sample	-	0.0005 s/sample	0.2225 s/sample
[38]	Decision Tree	HTML, JavaScript	0.15 s/page	-	0.034 s/page	0.184 s/page
MalFilter [39]	AdaBoost	URL, Host-based, Server-based	0.057 s/URL	-	0.018 s/URL	0.075 s/URL
RCNN	CNN	HTML/JavaScript-based image	-	0.081 s/page	0.00506 s/page	0.08606 s/page

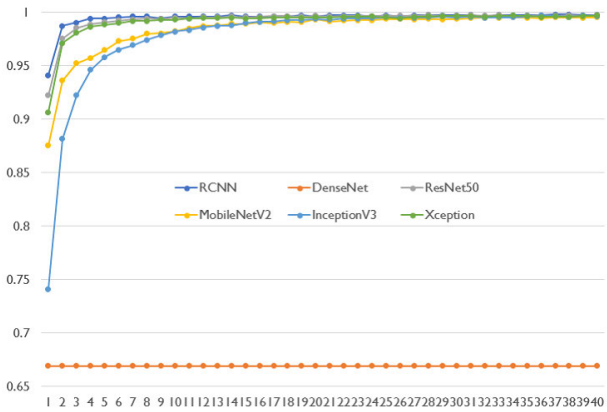

FIGURE 13. Changes in training accuracy of image models during 40 epochs.

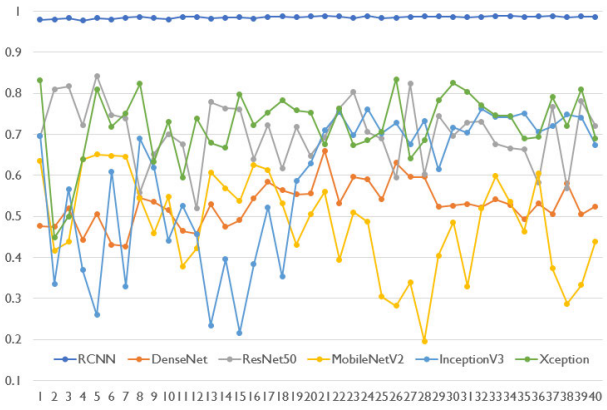

FIGURE 14. Changes in validation accuracy of image models.

and can categorize image types well as either benign or one of the EK types.

To measure the validation metric for multiclass classification, we calculate the micro-average and macro-average accuracy. Both accuracies can be calculated as follows:

$$ACC_{micro} = \frac{(ACC_1 * N_1) + \cdots + (ACC_k * N_k)}{N_1 + \cdots + N_k} \quad (3)$$

where ACC is the accuracy and N_k is the number of samples, respectively. Micro accuracy is calculated from individual accuracy of k in the multiclass model, whereas macro accuracy averages the performance of each individual class as formula (4).

$$ACC_{macro} = \frac{ACC_1 + \cdots + ACC_k}{k} \quad (4)$$

From Table 4, the ACC_{micro} of RCNN is 0.982. Its ACC_{macro} is 0.988.

TABLE 6. Train time comparison with CNN models. In this experiment, we used 25,820 samples for training, and applied 256 input size, 4 batch size, 40 epochs and 12 classes.

Model	Train Time
RCNN	9282.14 s
DenseNet121	41759.47 s
MobileNetV2	47640.38 s
InceptionV3	56754.79 s
ResNet50	66123.24 s
Xception	87797.60 s
VGG16	89238.82 s
VGG19	98585.27 s

In the performance comparison, as detailed in Table 5, RCNN is 38.3 times faster than typical machine-learning models [3]. The result is caused by the properties of RCNN that require unnecessary time consumption used for feature extraction. This benefit is well exposed to other CNN models. However, image-based CNN models require image transformation time, which accounts for most of the time consumed. Although image preprocessing is required, the overall performance is much higher than those of typical models.

In training time evaluation, most well-known CNN models require a large amount of time for training. In this case, RCNN is approximately 4.5 – 10 times faster than other CNN models, as presented in Table 6.

VI. DISCUSSION AND FUTURE WORK

RCNN exhibits high performance in a deep model with high image complexity. Relatively, the performance with low complexity is similar to that of the typical CNN model. Hence, it might be necessary to build an optimized model that exhibits high performance for images with low complexity. In future studies, we are going to add different map updates to distinguish the properties further. For instance, this is to use the highest value in the submatrix of the image pixel mapped to max-pooling when updating the max-pooling.

In general, CNN is a model in which the output matrix is reduced whenever the layers are deep. Sometimes, to make a more effective model, CNN increases the number of channels. Instead, we can extend the output matrix size by increasing the sizes of feature maps within our algorithm. Further, we can update max-pooling differently by using contrast padding. This can be one of the solutions to preserve image features more clearly.

Furthermore, we plan to build new CNN models that have suitable classification accuracies for high image complexities. Many CNN models often diminish the image features when they have deeper layers

TABLE 7. Confusion matrix between 1 and 63 image size. In this evaluation, we used 4496 test samples.

	p_angler	p_benign	p_blackhole	p_ckvip	p_gondad	p_magnitude	p_neutrino	p_nuclear	p_redkit	p_rig	p_sundown	p_sweet
y_angler	**0**	0	0	0	0	0	0	0	0	0	0	0
y_benign	0	**1369**	0	18	2	2	3	0	0	2	0	0
y_blackhole	0	0	**0**	0	0	0	0	0	0	0	0	0
y_ckvip	0	4	0	**2679**	0	0	0	0	0	1	0	0
y_gondad	0	17	0	0	**335**	0	0	0	0	0	0	0
y_magnitude	0	2	0	0	0	**0**	0	0	0	0	0	0
y_neutrino	0	17	0	0	1	0	**6**	0	0	0	0	0
y_nuclear	0	0	0	0	0	0	0	**0**	0	0	0	0
y_redkit	0	0	0	0	0	0	0	0	**0**	0	0	0
y_rig	0	17	0	0	1	0	0	0	0	**20**	0	0
y_sundown	0	0	0	0	0	0	0	0	0	0	**0**	0
y_sweet	0	0	0	0	0	0	0	0	0	0	0	**0**

TABLE 8. Confusion matrix between 64 and 511 image size. In this evaluation, we used 2788 test samples.

	p_angler	p_benign	p_blackhole	p_ckvip	p_gondad	p_magnitude	p_neutrino	p_nuclear	p_redkit	p_rig	p_sundown	p_sweet
y_angler	**34**	1	0	0	1	0	0	0	0	0	1	0
y_benign	0	**1647**	0	0	0	0	0	0	1	4	0	2
y_blackhole	0	6	**72**	0	0	0	0	0	0	2	0	0
y_ckvip	0	1	0	**2**	0	0	0	0	0	1	0	0
y_gondad	1	10	0	0	**538**	0	0	0	0	1	0	0
y_magnitude	0	1	0	0	0	**0**	0	0	0	0	0	0
y_neutrino	0	0	0	0	0	0	**0**	0	0	0	0	0
y_nuclear	0	1	0	0	1	0	0	**10**	0	0	0	0
y_redkit	0	0	0	0	0	0	0	0	**118**	0	0	0
y_rig	0	7	0	0	0	0	0	0	0	**27**	0	0
y_sundown	0	1	0	0	0	0	0	0	1	0	**0**	0
y_sweet	0	1	0	0	0	0	0	0	0	0	0	**295**

(i.e., vanishing tendencies). EKs have low image complexity, and general CNN models can be easily adopted in EK detection. However, malware is even more difficult to detect. To resolve the low detection problem, commercial methods select a large-scale dataset such as more than Peta datasets in the number of data elements (i.e., one of the commercial products, CylancePROTECT, can use this approach.) In this circumstance, we can update CNN models with the n-th repeats in image update according to image complexities. High image complexities can have a greater n. This model will help support complex image detection. Future work will consider the extension of this proposed model.

We also need additional experiments for proving the performance with a large-scale EK. Thus, we need a technique to discern remarkably image features accompanied by a large fraction of benign instances. This feature investigation can provide better detection models. We intend to evaluate this approach in future work.

We can face nontrivial issues in classifying benign images with apparently malicious images. Thus, problem-solving studies should be considered in this case. Nonetheless, the proposed model exhibited the potential for image-based detection by showing high accuracy in EK detection and type classification. This approach is rapid due to no feature extraction. Instead, the proposed model changes malicious codes to one-time image transformation. This simple approach provides quick and accurate detection performance. This model was more than 38 times faster than those in previous studies and showed a high classification rate of 98.2%. Moreover, the proposed method differentiates the weight of image features by updating the output matrix, so the accuracy of detection can be increased.

As a result, the advantages and disadvantages of our proposed model can be summarized as follows. **Low computational cost**: In exploit kit detection, previous detection methods required feature extraction time. However, our proposed model reduces computational costs by changing the exploit kit code to RGB (converting three RGB to the matrix) and then to a limited grayscale image (one matrix). **High performance**: This ConvNet-based exploit kit detection model can identify an obfuscated exploit kit without clarification via the deobfuscation of the original exploit kit. This provides rapid detection performance. This proposed model is 38 times faster than previous typical machine learning models. Also, the training time is 77.8% faster than prior CNN models, as shown in Figure 13. **Multiclass ConvNet model provides a high detection rate**: Based on each file size of the exploit codes, the size of images that are transformed to grayscale is classified into three different size groups and forwarded to a size-based differentiating model to filter the suspicious images. This approach facilitates the increase of the detection rate. Further, we enhanced classification accuracy by applying improved image processing techniques (RCNN) that recursively update the images. This is designed for sufficiently preserving image properties in order to overcome the vanishing gradient problem.

In contrast, in the case of converting a short attack code into an image, it is difficult to preserve the characteristics of the attack code in the image. (The same problem occurs with other static-based EK detection methods) It does not respond to an adversarial attack that inserts garbage code. This is a common disadvantage of the CNN model.

TABLE 9. Confusion matrix in samples more than 512 image size. In this evaluation, we used 72 test samples.

	p_angler	p_benign	p_blackhole	p_ckvip	p_gondad	p_magnitude	p_neutrino	p_nuclear	p_redkit	p_rig	p_sundown	p_sweet
y_angler	**0**	0	0	0	0	0	0	0	0	0	0	0
y_benign	0	**72**	0	0	0	0	0	0	0	0	0	0
y_blackhole	0	0	**0**	0	0	0	0	0	0	0	0	0
y_ckvip	0	0	0	**0**	0	0	0	0	0	0	0	0
y_gondad	0	0	0	0	**0**	0	0	0	0	0	0	0
y_magnitude	0	0	0	0	0	**0**	0	0	0	0	0	0
y_neutrino	0	0	0	0	0	0	**0**	0	0	0	0	0
y_nuclear	0	0	0	0	0	0	0	**0**	0	0	0	0
y_redkit	0	0	0	0	0	0	0	0	**0**	0	0	0
y_rig	0	0	0	0	0	0	0	0	0	**0**	0	0
y_sundown	0	0	0	0	0	0	0	0	0	0	**0**	0
y_sweet	0	0	0	0	0	0	0	0	0	0	0	**0**

VII. CONCLUSION

Exploit kits are one of the roots of malware contamination. This type of attack has rapidly increased. However, the detection rate is still low. In particular, antivirus products still exhibit low detection rates. Accordingly, we propose a CNN model based on image updates to conserve image features. To do this, scripts with malicious forms were converted into images. The changed image attributes provide useful insight into EK classification via the filter of a CNN model because this model is based on features, which are distinguished from benign HTML/JavaScript codes, through obfuscation, size, unique images, etc. These properties are well emerged when EKs are transformed into limited grayscale images. To make the model more precise, we proposed a limited grayscale, size-based hybrid model, and recursive image update method. This affects classification accuracy. In this paper, we expanded the scope of detection with image features. Above all, this model provides both high detection and high performance, which is a rare achievement in prior models.

APPENDIX

CONFUSION MATRIX IN RCNN

See Tables 7-9.

ACKNOWLEDGMENT

(Suyeon Yoo and Sungjin Kim are co-first authors.)

REFERENCES

[1] M. Hopkins and A. Dehghantanha, "Exploit Kits: The production line of the Cybercrime economy?" in *Proc. 2nd Int. Conf. Inf. Secur. Cyber Forensics (InfoSec)*, Nov. 2015, pp. 23–27.

[2] McAfee. (Feb. 2015). *Mcafee Labs Threat report*. [Online]. Available: https://docplayer.net/7975859-Report-mcafee-labs-threats-report.html

[3] D. Canali, M. Cova, G. Vigna, and C. Kruegel, "Prophiler: A fast filter for the large-scale detection of malicious web pages," in *Proc. 20th Int. Conf. World Wide Web (WWW)*, 2011, pp. 197–206.

[4] B. J. Kwon, J. Mondal, J. Jang, L. Bilge, and T. Dumitras, "The dropper effect: Insights into malware distribution with downloader graph analytics," in *Proc. 22nd ACM SIGSAC Conf. Comput. Commun. Secur. (CCS)*, 2015, pp. 1118–1129.

[5] X. Yuan, P. He, Q. Zhu, and X. Li, "Adversarial examples: attacks and defenses for deep learning," *IEEE Trans. Neural Netw. Learn. Syst.*, vol. 30, no. 9, pp. 2805–2824, Sep. 2019.

[6] J. Su, D. V. Vargas, and K. Sakurai, "One pixel attack for fooling deep neural networks," *IEEE Trans. Evol. Comput.*, vol. 23, no. 5, pp. 828–841, Oct. 2019.

[7] C. Grier, A. Pitsillidis, N. Provos, M. Z. Rafique, M. A. Rajab, C. Rossow, K. Thomas, V. Paxson, S. Savage, G. M. Voelker, L. Ballard, J. Caballero, N. Chachra, C. J. Dietrich, K. Levchenko, P. Mavrommatis, D. Mccoy, and A. Nappa, "Manufacturing compromise: The emergence of exploit-as-a-service," in *Proc. ACM Conf. Comput. Commun. Secur. (CCS)*, 2012, pp. 821–832.

[8] V. Kotov and F. Massacci, "Anatomy of exploit kits," in *Proc. 5th Int. Conf. ESSOS*. Berlin, Germany: Springer, 2013, pp. 181–196.

[9] B. Eshete, A. Alhuzali, M. Monshizadeh, P. A. Porras, V. N. Venkatakrishnan, and V. Yegneswaran, "EKHunter: A counter-offensive toolkit for exploit kit infiltration," in *Proc. NDSS*, 2015, pp. 1–15.

[10] S. Kim, J. Kim, and B. B. Kang, "Malicious URL protection based on attackers' habitual behavioral analysis," *Comput. Secur.*, vol. 77, pp. 790–806, Aug. 2018.

[11] B. Eshete and V. N. Venkatakrishnan, "WebWinnow: Leveraging exploit kit workflows to detect malicious urls," in *Proc. 4th ACM Conf. Data Appl. Secur. Privacy*, 2014, pp. 305–312.

[12] C. Curtsinger, B. Livshits, B. G. Zorn, and C. Seifert, "Zozzle: Fast and precise in-browser javascript malware detection," in *Proc. USENIX Secur. Symp.*, San Francisco, CA, USA, 2011, pp. 33–48.

[13] L. Xu, Z. Zhan, S. Xu, and K. Ye, "Cross-layer detection of malicious Websites," in *Proc. 3rd ACM Conf. Data Appl. Secur. Privacy*, 2013, pp. 141–152.

[14] S. Yoo, "Two-phase malicious Web page detection scheme using misuse and anomaly detection," *Int. J. Reliable Inf. Assurance*, vol. 2, no. 1, pp. 1–10, Jun. 2014.

[15] S. Kim and B. B. Kang, "Frism: Malicious exploit kit detection via feature based string-similarity matching," in *Proc. Int. Conf. Security Privacy Commun. Syst.* Cham, Switzerland: Springer, 2018, pp. 416–432.

[16] Y. Wang, W.-D. Cai, and P.-C. Wei, "A deep learning approach for detecting malicious JavaScript code," *Security Commun. Netw.*, vol. 9, no. 11, pp. 1520–1534, Jul. 2016.

[17] B. Stock, B. Livshits, and B. Zorn, "Kizzle: A signature compiler for detecting exploit kits," in *Proc. 46th Annu. IEEE/IFIP Int. Conf. Dependable Syst. Netw. (DSN)*, Jun. 2016, pp. 455–466.

[18] I. Yoo, "Visualizing windows executable viruses using self-organizing maps," in *Proc. ACM Workshop Vis. Data Mining Comput. Secur. (VizSEC/DMSEC)*, 2004, pp. 82–89.

[19] P. Trinius, T. Holz, J. Gobel, and F. C. Freiling, "Visual analysis of malware behavior using treemaps and thread graphs," in *Proc. 6th Int. Workshop Vi. Cyber Secur.*, Oct. 2009, pp. 33–38.

[20] K. Han, B. Kang, and E. G. Im, "Malware analysis using visualized image matrices," *Sci. World J.*, vol. 2014, Jul. 2014. Art. no. 132713.

[21] L. Nataraj, S. Karthikeyan, G. Jacob, and B. S. Manjunath, "Malware images: Visualization and automatic classification," in *Proc. 8th Int. Symp. Vis. Cyber Secur.*, 2011, p. 4.

[22] 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.

[23] B. Xiaofang, C. Li, H. Weihua, and W. Qu, "Malware variant detection using similarity search over content fingerprint," in *Proc. 26th Chin. Control Decision Conf. (CCDC)*, May 2014, pp. 5334–5339.

[24] L. Liu and B. Wang, "Malware classification using gray-scale images and ensemble learning," in *Proc. 3rd Int. Conf. Syst. Informat. (ICSAI)*, Nov. 2016, pp. 1018–1022.

[25] K. S. Han, J. H. Lim, B. Kang, and E. G. Im, "Malware analysis using visualized images and entropy graphs," *Int. J. Inf. Secur.*, vol. 14, no. 1, pp. 1–14, Feb. 2015.
[26] 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.
[27] A. Oliva and A. Torralba, "Modeling the shape of the scene: A holistic representation of the spatial envelope," *Int. J. Comput. Vis.*, vol. 42, no. 3, pp. 145–175, 2001.
[28] D. G. Llauradó, "Convolutional neural networks for malware classification," M.S. thesis, Dept. Comput. Sci., Univ. Politècnica de Catalunya, Barcelona, Spain, 2016.
[29] E. K. Kabanga and C. H. Kim, "Malware images classification using convolutional neural network," *J. Comput. Commun.*, vol. 06, no. 01, pp. 153–158, 2018.
[30] W. Wang, M. Zhu, X. Zeng, X. Ye, and Y. Sheng, "Malware traffic classification using convolutional neural network for representation learning," in *Proc. Int. Conf. Inf. Netw. (ICOIN)*, 2017. pp. 712–717.
[31] S. Seok and H. Kim, "Visualized Malware Classification Based-on Convolutional Neural Network," *J. Korea Inst. Inf. Secur. Cryptol.*, vol. 26, no. 1, pp. 197–208, Feb. 2016.
[32] S. Choi, S. Jang, Y. Kim, and J. Kim, "Malware detection using malware image and deep learning," in *Proc. Int. Conf. Inf. Commun. Technol. Converg. (ICTC)*, Oct. 2017, pp. 1193–1195.
[33] *Microsoft Malware Classification Challenge (Big 2015)*, Microsoft, Redmond, WA, USA, Feb. 2015.
[34] *KAIST Cyber Security Research Center*. [Online]. Available: http://csrc.kaist.ac.kr
[35] Wikipedia. (May 2018). *Information Theory*. [Online]. Available: https://en.wikipedia.org/wiki/Information_theory
[36] J. Ma, L. K. Saul, S. Savage, and G. M. Voelker, "Beyond blacklists: Learning to detect malicious web sites from suspicious URLs," in *Proc. 15th ACM SIGKDD Int. Conf. Knowl. Discovery Data Mining (KDD)*, 2009, pp. 1245–1254.
[37] A. Fass, R. P. Krawczyk, M. Backes, and B. Stock, "JaSt: Fully syntactic detection of malicious (obfuscated) JavaScript," in *Proc. Int. Conf. Detection Intrusions Malware, Vulnerability Assessment*. Cham, Switzerland: Springer, 2018, pp. 303–325.
[38] C. Seifert, I. Welch, and P. Komisarczuk, "Identification of malicious Web pages with static heuristics," in *Proc. Australas. Telecommun. Netw. Appl. Conf.*, Dec. 2008, pp. 91–96.
[39] G. Tan, P. Zhang, Q. Liu, X. Liu, C. Zhu, and L. Guo, "MalFilter: A lightweight real-time malicious URL filtering system in large-scale networks," in *Proc. IEEE Intl Conf Parallel Distrib. Process. Appl., Ubiquitous Comput. Commun.*, Dec. 2018, pp. 565–571.

SUYEON YOO received the B.S. degree in industrial engineering from Pusan National University, Busan, South Korea, and the M.S. degree in industrial and systems engineering from the Korea Advanced Institute of Science and Technology, Daejeon, South Korea, where she is currently pursuing the Ph.D. degree with the Graduate School of Information Security. Her current research interests include web security, big data analysis, machine learning, and malware detection and analysis.

SUNGJIN KIM received the B.S. degree in computer science from Ohio State University, Columbus, USA, and the M.S. degree in computer science from Sogang University, Seoul, South Korea, respectively, and the Ph.D. degree from the Korea Advanced Institute of Science and Technology (KAIST), Daejeon, South Korea, in 2019. He is currently an Assistant Professor with Cheju Halla University. His current research interests include network security, machine learning, big data analytics, social network analysis, web security, and malware detection and analysis.

BRENT BYUNGHOON KANG (Member, IEEE) received the B.S. degree from Seoul National University, the M.S. degree from the University of Maryland, College Park, and the Ph.D. degree in computer science from the University of California at Berkeley. He is currently an Associate Professor with the Graduate School of Information Security, Korea Advanced Institute of Science and Technology (KAIST). Before KAIST, he has been with George Mason University as an Associate Professor. He has been working on systems security area including botnet defense, OS kernel integrity monitors, trusted execution environment, and hardware assisted security. He is currently a member of the USENIX and the ACM.