Received 7 October 2023, accepted 19 October 2023, date of publication 1 November 2023, date of current version 7 November 2023.

Digital Object Identifier 10.1109/ACCESS.2023.3329112

APPLIED RESEARCH

A Proposed New Endpoint Detection and Response With Image-Based Malware Detection System

TRAN HOANG HAI[1], VU VAN THIEU[1], TRAN THAI DUONG[1], HONG HOA NGUYEN[1], AND EUI-NAM HUH[2], (Member, IEEE)

[1]School of Information and Communication Technology, Hanoi University of Science and Technology, Hanoi 10000, Vietnam
[2]Department of Computer Science and Engineering, Kyung Hee University, Yongin 17104, South Korea

Corresponding author: Tran Hoang Hai (haith@soict.hust.edu.vn)

This research was funded by the Vietnam Ministry of Education and Training under project number BKA-04-2022.

ABSTRACT Due to increased reliance on technology and cloud-based services, cyber risks are more common. Advanced persistent threats make it difficult to detect attacks, hence Endpoint Detection and Response (EDR) was adopted in 2013. EDR uses a scanning application on each endpoint machine to monitor and capture events and logs. However, EDR is vulnerable to attacks by malware, so a lightweight malware detector is needed. Image-based malware classification is a technique for classifying malware based on its representative image, but previous studies have not been integrated with EDR. This research aims to integrate EDR with an image-based malware classifier. A basic EDR implementation named Deep Ocean Protection System (DOPS) has been developed with two pre-trained models (Mobilenet V2 and Inception V3) fine-tuned with MalImg and BODMAS datasets. The models were evaluated with the DikeDataset and Mobilenet V2 fine-tuned with BODMAS 4.0.0 performed best in terms of loading and prediction time with a high AUC score of 0.8615. Inception V3 fine-tuned with BODMAS 4.0.0 also achieved a remarkable AUC score of 0.9392. These results show the potential of integrating image-based malware detection with EDR.

INDEX TERMS CNN, deep learning, endpoint detection and response, fine-tuning, malware classification, malware detection, malware visualization.

I. INTRODUCTION

Cyber threats are increasing due to the growing reliance on technology and cloud-based systems. In 2021, data breaches reached a high record, with cybercriminals stealing 5.9 billion user records [1] with much greater frequency compared to 2020 [2]. Following studies by IBM, 83% organizations reported more than one data breach, resulting in an increase in prices for users [3]. Cybersecurity failure is considered one of the most threatening risks for the world economy during the COVID-19 pandemic [4].

Most cyber-attacks begin with penetrating an organization's internal networks, which take an average of two days [5]. Advanced persistent threats (APTs) are particularly dangerous as they use multiple techniques to control compromised networks without being detected [6]. In 2013, A. Chuvakin brought up the term "Endpoint Detection and Response" (EDR) [7], which is a security process that detects attacks in a timely manner by staying on each endpoint inside the organization's network to capture data and events continuously.

Along with the boom of cyber threats in 2013, malicious software (malware) has been dramatically spreading since 2019, with more than one billion malware samples found since then [8]. To counter the malware, further products and research have been carried out with two major paths: static and dynamic analysis. Both of these techniques consume a great deal of time when applied to a vast number of malware samples. However, the rise of machine learning and deep learning methods in the 1990s has led to significant improvements in malware detection in terms of extracting distinguishing characteristics from malware.

As a result, an EDR solution that can detect malware itself can be more effective, especially when integrated with a

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

lightweight image-based malware detection system. Image processing techniques can be applied to convert a binary to a grayscale image to detect malware, and malware files of the same family often share the same image structure [9].

This paper aims to lay the foundations for the integration of the EDR process with an image-based malware detector. To achieve this, a fundamental structure for EDR will be constructed in an extendable way so that any service, even an image-based malware detector, can be integrated. This implementation also needs to accommodate the minimum features of an EDR solution. Moreover, to capture the system data in a real-time manner, a kernel driver is developed to support the system. As a result, the tested models for the malware detector will have high prediction performance while minimizing disk space and prediction time.

This paper makes three contributions:

- Establishing a basic implementation called Deep Ocean Protection System (DOPS) with essential EDR features.
- DOPS is equipped with a kernel driver, named DODriver, to capture process creation and termination notifications continuously and block any process by its full path if necessary.
- Fine-tuning two pre-trained models (Mobilenet V2 and Inception V3) with customized versions of MalImg [9] and BODMAS [10] dataset and evaluating them with DikeDataset. Throughout the experiment, the most suitable model has been integrated with DOPS.

II. RELATED WORK

The rise of commercial EDR products integrated with antiviruses has led to an increase in research papers on EDR solutions. G. Karantzas et al. have proposed an advanced EDR solution to measure their response time and ability to detect threats [6]. Hassan et al. proposed a graph-based approach to discover dependencies between threat alerts, which are generated from EDR [11]. In this approach, a scoring methodology was constructed to reduce the false alarm rate of EDR. Kaur et al. discussed the application of machine learning techniques on EDR to detect malware threats [12].

Most commercial EDR products are closed source, making a basic implementation of EDR unclear. In recent years, image-based malware classification has become popular, however, the application of this method to solving EDR problems has not received much attention. In the next paragraphs we will discuss what benefits image-based malware classification could bring to malware detection in general and to EDR in particular.

The application of machine learning and deep learning techniques for analyzing and detecting malware has been discussed by Kim et al. [13]. Malware analysis can be divided into two types: static analysis, which analyzes the binary code, and dynamic analysis, which detects suspicious behaviors when the binary is executed in a virtual environment. Dynamic analysis can be more accurate but requires more resources, while static analysis requires the most comprehensive investigation, especially when binaries are packed or obfuscated, but brings more coverage in analysis than dynamic analysis. In 2011, [9] proposed a new approach to analyze malware. This is a visualized-based analysis method which uses image textures obtained from transforming 8-bit integer vectors into 2D arrays for classification. By visualizing a portable executable (PE), different sections inside the PE [14] can be distinguished clearly. In addition, these distinctive image textures are not only found in malware but also in other domains [15].

As mentioned above, applying machine learning and deep learning approaches to malware analysis promise an improvement in the accuracy of malware detection without requiring a great understanding of malware [16]. Naeem et al. proposed IoT-based hybrid models trained with dynamic features, with different image sizes, achieved 98.79% accuracy [17]. Moussas et al. used a malware detection system with two levels achieving an accuracy of 99% against 9339 images in MalImg dataset [18]. Jeyaprakash et al. dealt with unbalanced obstacles by reweighted class-balanced loss function in the final classification layer of the DenseNet model [19].

Malware classification method using CNN model has been studied by El-Shafai et al. [20]. In this research, the authors have used different models, including VGG16, Inception V3, MobileNet V2, ResNet 50, AlexNet, DenseNet 201, DarkNet 53, Places365 GoogleNet. All models were evaluated against the MalImg dataset. They found that fine-tuned VGG16 outperformed other traditional models with a classification accuracy of 99.97%. In study by Rezende et al. [21], the use of the ResNet 50 model achieved 98.62% accuracy on MalImg after 750 epochs of transfer learning. Kumar et al. [22] compared classic machine learning models trained on four image formats: Grayscale, RGB, CMYK, and HSL. They found that the highest accuracy achieved was 91% using a random forest model trained with grayscale images. Awan et al. [16] combined VGG19 with Spatial Convolutional Attention to classify malware in the MalImg dataset, achieving an accuracy above 97% when trained with class balancing on benign class.

Although much research has been proposed to address the EDR problem, most of these studies only evaluated benchmark datasets and did not examine other datasets with a large number of families. In addition, most of the trained models were limited to being malware classifiers rather than malware detectors. In this paper, we propose a basic implementation of an EDR solution integrated with an image-based malware detector, which will be upgraded from the malware classifier trained on BODMAS.

III. BACKGROUND

A. WINDOWS SYSTEM CONCEPT

1) KERNEL DRIVER DEFINITION

Kernel drivers are loadable modules that execute in kernel mode and have a higher privilege level than user-mode applications. They have a ".sys" extension and most are hardware device drivers that work with the I/O manager and relevant hardware. Windows also supports software drivers that execute tasks in the kernel as instructed by user-mode applications [23]. DODriver is a software driver written for gaining process notifications to support ESP and block programs by path if necessary. DDODriver is constructed on top of the kernel-mode driver framework (KMDF) provided by the Windows driver framework (WDF). The entry point of a WDF driver is called DriverEntry, which initializes all prerequisite steps. It returns the status based on initialized steps, undoing previous work if any step fails.

TABLE 1. Common major routines [24, p. 44].

Major function	Operation	Invoked By
IRP_MJ_CREATE	Create	CreateFile, ZwCreateFile
IRP_MJ_CLOSE	Close	CloseHandle, ZwClose
IRP_MJ_READ	Read	ReadFile, ZwReadFile
IRP_MJ_WRITE	Write	WriteFile, ZwWriteFile
IRP_MJ_DEVICE _CONTROL	Control	DeviceControl, ZwDeviceControl

2) KERNEL DRIVER OBJECT AND DEVICE OBJECT

The driver object is allocated and initialized by the kernel before it is provided to the driver's entry point. Dispatch routines are attached to the driver object to handle requests from user-mode applications. These routines are specified by an IRP Major Code and most common ones are listed in Table. 1 [24]. The IRP_MJ_CREATE and IRP_MJ_CLOSE functions are necessary for the driver to communicate with user-mode applications. The IoCreateDevice function creates a new device object and stores it as a member of the driver object.

3) MEMORY ALLOCATION

The memory manager creates two types of memory pools: a paged pool and a non-paged pool. The paged pool can be paged out to save memory but causes page faults, which can lead to system crashes if it is not handled properly. The non-paged pool ensures that memory always remains in RAM. In the context of the DODriver program, a non-paged pool is used to maintain a blocklist of program paths to prevent prohibited processes from running.

4) CLIENT AND DRIVER COMMUNICATION

The Driver module in ESP consists of a driver residing in kernel mode interacting with a user-mode application. The driver responds to the application's requests with kernel information. This common type of interaction in an operating system is called client and driver communication. Dispatch routines are used to receive and process I/O request packets (IRPs). This driver investigates these IRPs and executes any action necessary.

By examining IRP and IO_STACK_LOCATION, we can find information about user-mode requests, such as command code and input buffer. The IOCTL code is constructed using the CTL_CODE macro with four parameters: DeviceType, Function, Method and Access. The Method parameter is the most important because it describes how data is transferred between the client and the driver.

Buffered I/O and Direct I/O methods are used for input/output buffers. The caller's input buffer has a copy version in system space (Buffered I/O) and the caller's output buffer is locked, providing an equivalent system address with a Memory Descriptor List (Direct I/O). While METHOD_IN_DIRECT allows the driver to read the caller's output buffer, METHOD_OUT_DIRECT only provides the writing permission to the driver.

Without Buffered I/O and Direct I/O, the driver has to handle the memory allocation, page faults, and other memory-related problems.

5) PROCESS NOTIFICATION

Windows provides Event Tracing for Windows (ETW) for both user-mode applications and kernel drivers to receive notifications when processes are created or terminated. However, these notifications are delayed by 1-3 seconds due to performance reasons. This raises a problem when applications want to keep up with process notification. Starting from Windows 2000, process notifications became a reality for any security software. Whenever a process is created or destroyed, its notification will be sent to interested drivers in a real-time manner. This was simply achieved by registering a callback via the PsSetCreateProcessNotifyRoutineEx API [23].

B. CONVOLUTION NEURAL NETWORKS

1) CONVOLUTIONAL NEURAL NETWORK DEFINITION

The Convolutional Neural Network (CNN) is a type of deep neural network that has been very successful in image analysis. It contains convolutional layers and fully connected layers, which mimic the layered sub-regions in the visual natural cortex [25]. In particular, an activation function called Rectified Linear Unit (ReLU [26]), which is generically used in CNNs, is inspired by ON/OFF ganglion cells [25].

CNNs have three main types of layers: convolutional layer, fully connected layer, and pooling layers. The convolution and pooling layer are used to detect low-level patterns in data. While convolution layers extract features from data, pooling layers decrease the size of the input to wipe out noise. The output is fed to the fully connected layers for fusing all extracted features and bringing out a specific output.

2) MOBILENETV2

To deploy a deep learning application on devices with limited resources, one of the most lightweight that is widely used is the Mobilenet V2.

Mobilenet V2 uses Depthwise Separable Convolutions with the addition of residual bottleneck layers. A Depthwise Separable Convolution is used to replace a full convolutional layer with two separate layers. The first layer, called depth wise convolution, works by applying a single convolutional

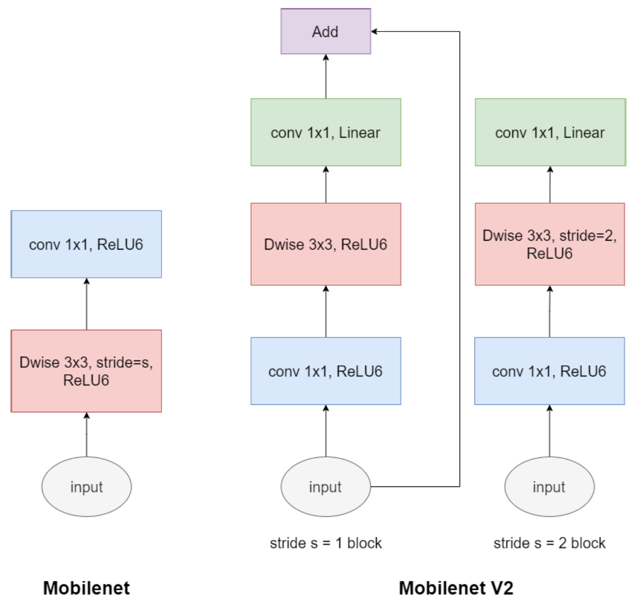


FIGURE 1. Comparision of convolutional blocks for mobilenet and mobilenet V2 [27].

filter per input channel, performs a lightweight transformation. The other layer called pointwise convolution is a 1 × 1 convolution. This layer computes linear combinations of the output by the first layer.

Mobilenet V2 differs significantly from Mobilenet V1 by using two types of blocks: stride 1 residual blocks and stride 2, which aim to reduce size of input. Both these blocks consist of three layers: a 1 × 1 convolution followed by depth wise and pointwise layers. The first two layers in the blocks are associated with the ReLY activation function, which is robust with low-precision computation. The last layer comes with a linear activation function to prevent non-linearities from losing too much information in a low-dimensional subspace of the input space [27].

The stride 1 depth wise separable convolution block integrates with two special terms: "inverted residual" and "linear bottleneck". Number of channels in the input and the output of this block is much less than in the middle. The last convolution layer uses linear activation function (linear bottleneck). While storing all the essential information in the bottleneck, the block adds a shortcut to reduce information loss. The residual connections in Mobilenet V2 are more memory-efficient than traditional designs due to the middle channels having fewer channels. For this reason, Mobilenet V2's residual blocks are called inverted residual bottlenecks.

The Mobilenet V2 consists of 32-filter fully convolution layers and contains 19 residual bottleneck layers, as shown in Fig. 1 [27]. Mobilenet V2 has been demonstrated to be a high-performing, lightweight convolutional neural network architecture.

3) INCEPTIONV3

The demand for less computational cost and a lower number of parameters exist frequently in various scenarios. The Inception family of models, which consists of multiple parallel filters of different sizes, to balance the width and depth of the network. Especially Inception V3, has successfully responded well to such requirements. Additionally, 1 × 1 convolutions are used to decrease computational costs. As stated in [28], by balancing both sizes of the network, higher accuracy can be achieved.

Inception V3 includes multiple inception modules and four important modifications. The first technique is to factorize large convolutional layers into smaller ones to reduce computational cost. The second technique replaces convolution layers with asymmetric convolutions which works well on feature maps with medium grid-sizes. The auxiliary classifier acts as a regularizer to improve quality when the network reaches high accuracy. Finally, the last technique expands the activation dimension of the network filters to reduce the grid size of feature maps while avoiding a representational bottleneck. This is achieved through two parallel stride 2 blocks: a P (pooling layer) and a C (convolution layer), which are concatenated at the end.

Inception V3 is a deep convolutional model with 48 layers that has achieved over 78.1% accuracy on the ImageNet dataset.[1] Its efficiency is the result of not only the inception module, but also other valuable ideas that have been researched over the years.

4) TRANSFER AND FINE-TUNING LEARNING

Transfer learning involves using the parameters of a neural network trained on one dataset to solve a different problem with a different dataset. Common features learned in the early layers of the network can be applied to many datasets in other domains, making it an effective approach for training with a smaller dataset. Fine-tuning is an optional step in transfer learning that involves retraining a portion of the pretrained model with data from the applied dataset, producing significant improvements.

IV. PROPOSED EDR SYSTEM

A. OVERVIEW OF PROPOSED EDR SYSTEM

This paper describes the integration of a naive implementation of EDR, called Deep Ocean Protection System (DOPS), with an image-based malware detection system for further study and analysis.

DOPS contains two main parts: Endpoint Service Pack (ESP) and Endpoint Microservice. As shown in Fig. 2, ESP is installed on each user machine and continuously collects data from the user machine. This data is transferred to Endpoint Microservice via REST APIs and then stored in a database for later query purposes.

By using REST APIs, a user machine is not required on the same network as the server. DOPS is suitable for any scope. However, due to its complexity, DOPS are most suitable for organizations. For companies controlling various APIs,

[1]https://cloud.google.com/tpu/docs/inception-v3-advanced

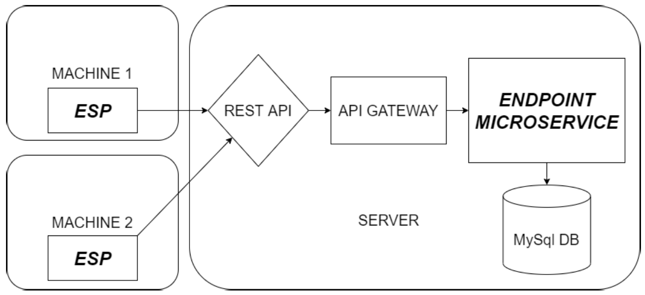


FIGURE 2. Overview architecture of deep ocean protection system.

a gateway is sufficient to map the ESP requests to an Endpoint Microservice.

A web user interface for DOPS is designed to monitor and visualize the collected data, allowing administrators to set up policies or take immediate actions if necessary. To ensure privacy, the organization should construct the service, database, and user interface.

B. ENDPOINT SERVICE PACK

ESP is a collection of executables designed to support EDR on every endpoint. Its architecture is based on the microservice model, where each executable module is considered as a self-operating module that implements a single business capability. ESP includes the Launcher, Scanner, Deep Ocean Malware Detector (DOMD), and Driver module, which consists of the Deep Ocean Client Go (DOCG) and Deep Ocean Driver (DODriver), as illustrated in Fig. 3.

The Launcher is activated upon machine startup and spawns the Scanner, followed by DOMD and DOCG. When Scanner, DOMD, and DOCG are launched successfully, Launcher can now be terminated as if its mission is only spawning other services in the ESP.

Scanner is responsible for collecting data from the endpoint and obtaining system data from the Driver module. It sends data to the server and executes commands when needed, serving as the "Response" in EDR. Scanner also includes a vulnerability scanning plugin and can request DOMD to scan input files for maliciousness.

The Driver module comprises two components: DODriver and DOCG. DODriver is a kernel driver that collects data from the deep kernel, while DOCG acts as an interpreter that transfers system data to the Scanner and Scanner's commands to DODriver. DOMD is an image-based malware classifier that labels files requested by Scanner as malware or benign.

C. LAUNCHER

The Launcher's primary purpose is to start necessary services based on settings. This process is called the service mode and runs continuously while the machine is on. The Launcher is designed to work on Windows 10 $\times$ 64 and can be configured to launch automatically using the Windows Task Scheduler by creating a periodic task to start the Launcher process.

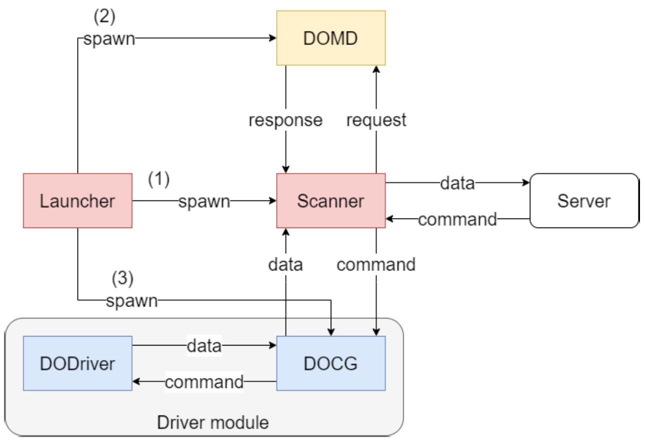


FIGURE 3. Overview architecture of endpoint service pack.

The Launcher is capable of updating all components of ESP, including itself. However, if an attacker gains access to the Launcher, they could replace any component with malware, then the machine will no longer be in the protection of DOPS. To prevent this, ESP is installed in a location called "C:ProgramData.deep_ocean", which is only accessible by machine administrators. This path is set to the "C" drive, which is where Windows system files are stored, and ESP is designed to find a suitable location if the drive letter is different.

The Launcher's service mode is only enabled after administrators authenticate ESP on that endpoint by sending a key to the server for verification. The server sends a permanent token to the Launcher, identifying the machine and saving it to the local database. On next runs, the Launcher loads the saved secret and authenticates with the server.

D. SCANNER

Scanner's main mission is to scan the machine. It scans user mode events, system events, logs, vulnerabilities, and malware using a knowledge base [29] to store data. The Scanner runs discovery plugins to gather basic user mode information, e.g., operating system version, service list, etc.

When running in service mode, the scanner scans vulnerabilities using audit plugins which are proposed in this paper for vulnerability sweeping. This plugin is being developed for each vulnerability reported in Metasploit, and more authors can work independently to add more audit plugins.

The Scanner communicates with the Driver module and DOMD via Named Pipe for IPC. Its second mission is to periodically send captured data to the server and receive commands via REST APIs. Because data collected from a machine is very diverse, Scanner and Endpoint Microservice must commit to the same protocol for each type of data.

An EDR system's power comes from the ability to automate and respond quickly to threats. With the careful setup policies and cautious analysis from the blue team, EDR can provide more protection for the entire organization's network.

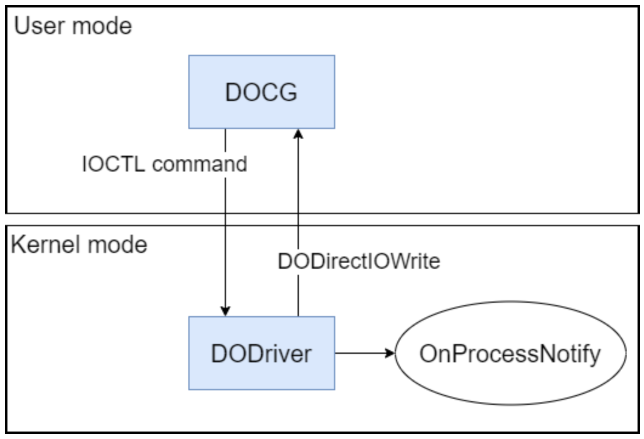


FIGURE 4. Overview architecture of deep ocean driver module.

E. DRIVER

The DOPS system aims to capture OS events and process notifications in real-time. Various methods exist for finding process creation and termination, including polling, ETW, and the "PsSetCreateProcessNotifyRoutineEx" API. Unfortunately, from a security solution's perspective, this method is not enough. In this paper, the Driver module is developed for retrieving process creation and termination notifications and blocking any process creation based on the blocklist sent from the server.

As shown in Fig. 4, the Deep Ocean Driver module in ESP consists of two submodules: the user mode Deep Ocean Client Go (DOCG) and the kernel driver. The DOCG transfers data along with the server command with the Scanner module. The kernel driver receives notifications from the system by passing the "OnProcessNotify" callback to the "PsSetCreateProcessNotifyRoutineEx" API for gathering creation and termination notifications and blocking any process creation if necessary.

The communication between the DODriver and the DOCG is done by the Direct I/O method and the "IRP_MJ_DEVICE_CONTROL" dispatch routine. The Direct I/O method is used to send process notifications to DOCG. The dispatch routine helps DOCG send IOCTL commands to DODriver. The client side cannot force the driver to do what the driver cannot or is not permitted to.

The Scanner running in service mode will open a two-way named pipe for communicating with other services. DOCG connects to this pipe to transfer and receive data while kernel data retrieved by DODriver is sent to the Scanner.

The traditional model of using a driver in the kernel and a client in user mode is frequently used in the Windows system, usually written in C or C++. However, in this paper, the client was rewritten in Golang because it is more user-friendly and has various library support.

F. DEEP OCEAN MALWARE DETECTOR

As shown in Fig. 5, DOMD consists of two main components, Preprocessor and Detector. The Preprocessor is responsible for converting the input binary to an image before it is passed to the detector for the final label, as described in Algorithm 1 [30, p. 196].

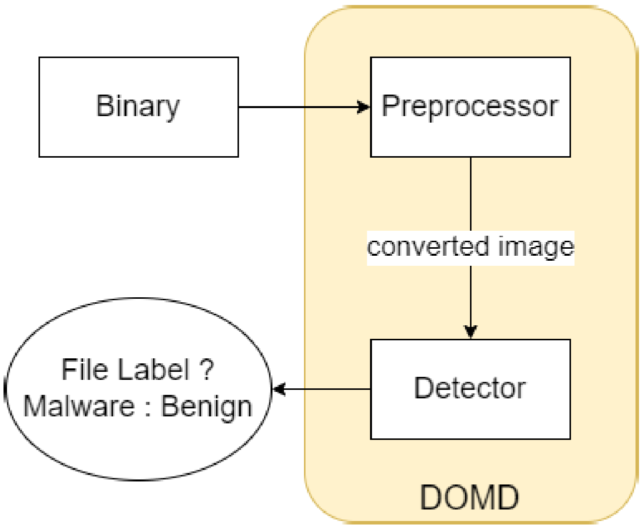


FIGURE 5. Overview architecture of deep ocean malware detector.

The algorithm receives byte data from an executable and a demanded width as inputs. The caller reads the byte data from the executable before passing to this algorithm. Second input is width, which, for instance, is set to 256px. The output image has a fixed width of 256px and a height depending on the size of the original binary. The final grayscale image has the last byte removed since it is typically padded with zero bytes.

While Algorithm 1 explains the way to construct image data from an executable, Algorithm 2 extracts sections from a binary and writes them as a single image. It uses the "pefile" library to read the executable as an object and "get_data()" method to get the raw byte data for each section. The raw data is then added to the "saved_data" variable, which is passed to Algorithm 1. The final step is writing the aligned byte to the image. The algorithm also allows the user to choose specific sections to extract.

After receiving an image from Preprocessor, the Detector will produce a probability P for the malware family with the highest probability. If P is higher than a predefined threshold T, the file is labeled as malware, otherwise it will be tagged as benign. In the case of benign family, If P is higher than 0.5 but the label is benign, the probability is subtracted from 1 before being compared to T.

Overall, the big picture of EDR in general and DOPS in particular has been depicted in detail with each component's missions and their stringent relationships. While the Driver module is developed to support Scanner in capturing data from the kernel, an image-based malware detector, called DOMD, is integrated to help Scanner verify executables as malware or benign. In particular, DOMD has to be trained and evaluated first before being ready for deployment with ESP. In terms of memory, DOMD is expected to take only less than 1GB. Therefore, the next section illustrates how simulation methods are set up to find the optimal image-based models. Moreover, the method of finding the mentioned threshold T is also revealed after this chapter.

Algorithm 1 Align Byte By Width [30, 196]

```
Input: data, width
Output: aligned_byte, byte data had been aligned by
        width
1 size ← len(data) ;   /* get length of byte
    data */
2 if size is 0 then
3   return null;
4 rem ← size mod width;
5 aligned_byte ← data[0 : size − rem];
6 return aligned_byte
```

Algorithm 2 Write Sections From PE To Image

```
Input: input_path, output_path, width,
       section_names
Output: error if any
1  saved_data ← [ ] ;  /* initialize an empty
     list */
2  pe ← pefile(input_path) ;   /* get data from
     PE */
3  nb_sections ← len(pe.sections) ;  /* get number
     of sections */
4  if nb_sections is 0 then
5    return Number of sections is 0;
6  foreach section ∈ pe.sections do
7    if len(section_names) == 0 or
      (len(section_names) > 0 and
      section.Name ∈ section_names) then
8      add section.get_data() to saved_data ;
        /* add all data if no specific
        section is requested or only
        add data of section that is
        demanded */
9  if len(saved_data) is 0 then
10   return Extracted sections have size of 0;
11 aligned_byte ← Algorithm 1 (saved_data, width);
12 write aligned_byte to output_path;
13 return ; /* no error -> return empty */
```

V. PROPOSED IMAGE-BASED MALWARE DETECTOR

A. SIMULATION METHOD

This collection of simulations aims at developing a model that is able to detect whether a file is malware or not. This project trains a malware classifier first and then detects files' maliciousness. The DOMD project involves two phases, as can be seen in Fig. 6. The first phase involves training a

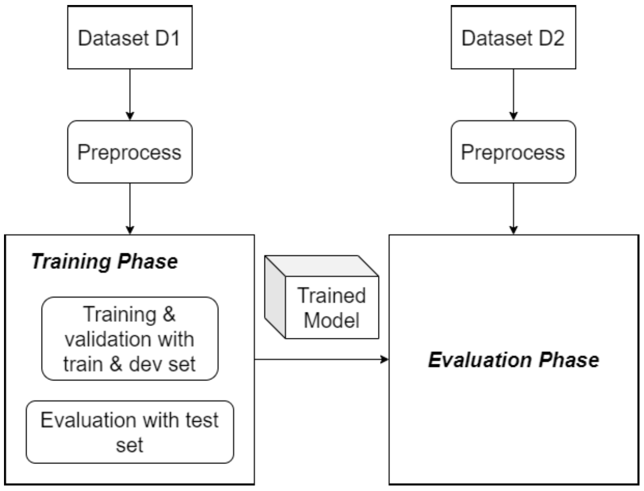


FIGURE 6. Simulation phases.

multiclass malware classifier to recognize the family of an input malware sample. The second phase tests the model's ability to predict malware and benign files that have not been seen before.

The project uses two types of datasets: D1 for training the model and D2 for evaluating the model's generalizability in real-life scenarios. While the MalImg [9] and BODMAS [10] involve the first phase. The DikeDataset is used for evaluating the generalizability of trained models in the second phase. The D1 dataset is divided into three subsets for training, development, and validation.

Studies showed that fine-tuned and transferred learning achieve good results in image-based malware classification [21], [31], [32] but also outperform other traditional networks [33]. Instead of creating new neural networks or training from scratch, the main strategy is to fine-tune with models that have already been trained.

Mobilenet V2 model was born as a lightweight model but performed well in the image classification task. Inception V3 model achieved state-of-the-art performance in grayscale malware image classification [34]. Two models were chosen for training a malware detector because they need to be small enough to integrate with ESP and detect malware in a timely manner.

To form a malware classifier, pre-trained models are fine-tuned and adjusted with additional layers. Both models are fine-tuned using weights learned from ImageNet [35]. Inspired by [36], models are adjusted with some additional layers as in Fig. 7. MobileNet V2 is appended with a global average pooling layer, a fully connected layer with 512 nodes using ReLU activation, a dropout layer, and a final layer for prediction.

The existence of the pooling layer and the dropout layer reduces overfitting significantly. Applying the result that a grayscale image achieves better performance than an RGB image [37]. The models receive grayscale images as input. A convolution is prepended to convert the grayscale input to a 3-channel input. Inception V3 is also constructed in the same

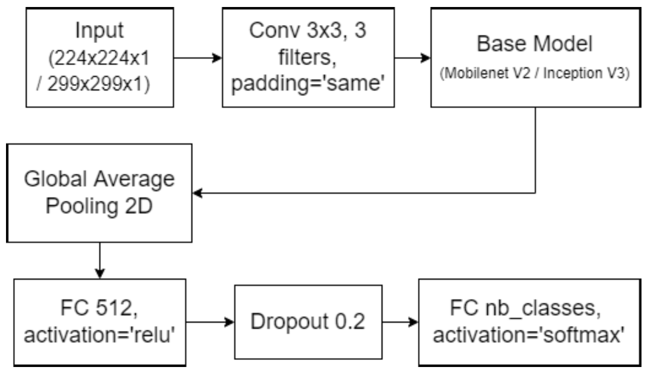

Input (224x224x1 / 299x299x1)
Conv 3x3, 3 filters, padding='same'
Base Model (Mobilenet V2 / Inception V3)
Global Average Pooling 2D
FC 512, activation='relu'
Dropout 0.2
FC nb_classes, activation='softmax'

FIGURE 7. Mobilenet V2 / inception V3 construction diagram.

TABLE 2. Fine-tuning model parameters.

Model	Input Shape	Epochs	Batch size	Learning Rate	Frozen layer
Mobilenet V2	224x224x1	20	16	0.0001	96
Inception V3	299x299x1	20	16	0.0001	249

way as Mobilenet V2 except that it uses an image with a shape of 299 × 299x1 as the input.

In general, both models are fine-tuned within 20 epochs with a training batch size of 16. Mobilenet V2 and Inception V3 freeze the first 96 and 249 layers, respectively to avoid training from scratch. Both models use Categorical Cross Entropy as the loss function and Adam as the optimization algorithm. Parameters that gain better loss on the dev set are saved to the cloud during the training process. The imbalance dataset is addressed by giving higher weight to the data in minority classes.

The fine-tuned models are evaluated based on the parameters listed in Table. 2, and in the evaluation phase, all fine-tuned models are measured in comparison to models fine-tuned with MalImg. This is due to the fact that MalImg has become a competition benchmark dataset in the Kaggle Microsoft Malware Classification Challenge [38].

In the first phase, the training step is done on a Colab free plan with GPU enabled, and the evaluation step is also done on Colab but without GPU. Besides, preprocessing in the first phase and the evaluation phase are conducted by the Intel Xeon CPU E5-2667 2.90 GHz with 16 GB of RAM with Windows 10 Pro 21H1 64-bit installed. The rest of the section describes the experiment scenarios for each dataset.

1) FINE-TUNING WITH MALIMG

The MalImg dataset is a widely used dataset in image-based malware analysis that contains 9339 grayscale images divided into 25 malware families. As shown in Fig. 8, the dataset has an imbalance issue, with two families having more than 15% of the total number of samples, and a significant gap

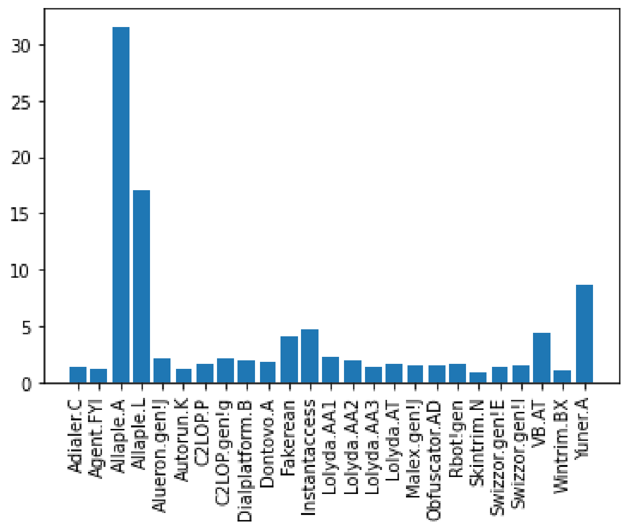


FIGURE 8. Data distribution of MalImg dataset.

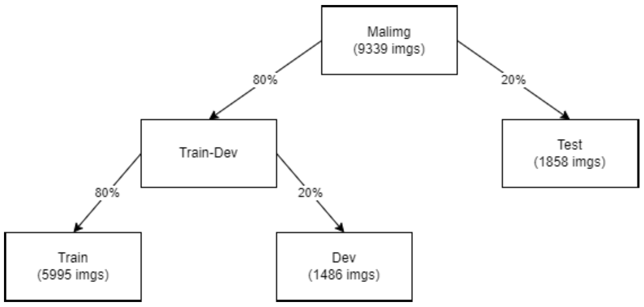


FIGURE 9. Splitting ratio of MalImg dataset.

between the number of samples in the family with the most (Allaple.A with 2949 images) and the least (Skintrim.N with 80 images) number of specimens. The dataset is used to prove the learnability of Mobilenet V2 and Inception V3.

The training phase consists of training and evaluating the model on different subsets of the dataset. MalImg is split into three subsets: train, dev, and test sets. The train set provides training data, the dev set is used to determine optimal parameters, and the test set is used to evaluate model accuracy.

The ratio of train, dev, and test sets is 64, 16, and 20, respectively, with 20% of MalImg set aside for testing. The rest of it is split into the train and dev set with a ratio of 80 to 20, respectively. The splitting process, shown in Fig. 9, is done using the ''validation_split'' attribute of ''ImageDataGenerator'', resulting in 5995, 1486, and 1858 grayscale images for the train, dev, and test sets, respectively. This ratio is kept constant for later scenarios.

MalImg contains a small number of samples, which might not be enough for training a good malware classifier. Therefore, two selected CNN networks are also trained separately with the BODMAS dataset in the next scenarios.

2) FINE-TUNING WITH BODMAS

The BODMAS dataset has a huge number of malware samples and families, which were gathered from August 2019 to September 2020. The dataset includes disarmed executables for 57,286 malware samples within 580 families, feature vectors for both malware and 77,142 benign samples. To prevent accidental execution of malware, each binary is disarmed by setting the OptionalHeader.Subsystem and the FileHeader.Machine to zero, and all binaries are provided with the ".exe" extension. With about 302 GB in storage, one of the main targets of BODMAS is producing more malware samples spanning over years when combined with other famous datasets, such as Ember [39] and SOREL-20M [40]. Indeed, the BODMAS dataset is larger than the MalImg dataset and provides more information for training a malware classifier.

To apply BODMAS with image-based classifiers, malware binaries must be converted to grayscale images before training. One of the biggest problems is finding the optimal size for images. Executables with a huge size might produce an image with different alignments of bytes, which makes it harder to classify malware correctly. Moreover, without the use of the Spatial Pyramid Pooling Network [41], CNN networks are only able to learn with a fixed image shape. However, in the scope of this paper, images are only reshaped with a fixed size of 256 in width. In the training step, each image is resized again to a suitable size corresponding to each model. Another challenge in training with BODMAS is the heavily imbalanced properties of BODMAS 2. In particular, a great number of families have much fewer samples compared to the rest. As a result, new subsets of BODMAS are created to address these challenges.

The first new subset is BODMAS 3, which applies two strategies: preserving the most important samples and removing outliers. The first technique is extracting data from only families with the greatest number of samples. The second one is removing outliers in size and extracting the most important signatures in malware. The order of applying these techniques results in different subversions of BODMAS 3, as can be seen from Fig. 10. With the integration of filters, BODMAS 3 aims at reducing the effects of outliers and the tremendous imbalance features of the original BODMAS. Most of the families obtained in all subversions are well known, e.g., autoit, autorun, wacatac, etc. This ensures that the dataset will not omit famous malware families.

In general, to deploy a deep learning model in a real-life scenario, the model itself needs data from the application field. In order to achieve good performance on prediction, input data must fall within the range of what the model has learned. In this case, the model, after training from BODMAS 3 or earlier, only learns about malware classes. Hence, it might perform badly when encountering benign samples. Therefore, to achieve high accuracy, a benign class is supplied to each subversion in BODMAS 3,

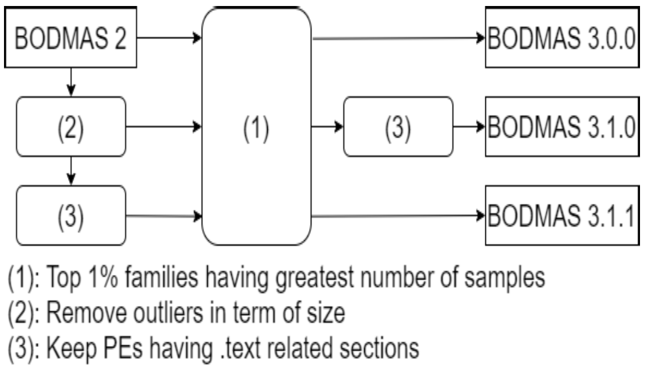

BODMAS 2
(2)
(3)
(1)
(3)
BODMAS 3.0.0
BODMAS 3.1.0
BODMAS 3.1.1
(1): Top 1% families having greatest number of samples
(2): Remove outliers in term of size
(3): Keep PEs having .text related sections

FIGURE 10. **BODMAS 3 construction diagram.**

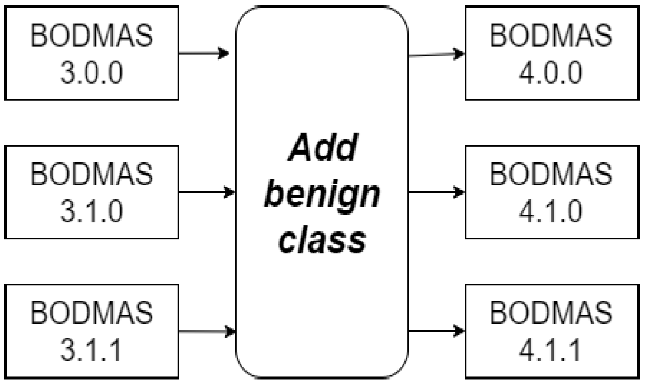


FIGURE 11. **BODMAS 4 construction diagram.**

as in Fig. 11. In detail, the added benign files are obtained from the "Malware Detection PE-Based Analysis Using Deep Learning Algorithm Dataset"[2] with 1000 samples. In fact, these samples were later coincidentally found to be a subset of DikeDataset. This points out the reason for the high performance of models trained with BODMAS 4, which will be discussed in a later section. After the preprocessing step, BODMAS 4.0.0, 4.1.0, and 4.1.1 are provided with 980 all-sections converted grayscale images and 965 ".text-related-sections" converted grayscale images.

In addition, due to the great performance of models trained with BODMAS 4.0.0, an extra dataset called "BODMAS 4.0.0 full" was created to find the best ability reached by selected models. This dataset is the same as BODMAS 4.0.0 but omits the evaluation step of the training phase. This means 80% of samples are used for training, with the remainder saved for validation.

Overall, data of MalImg and different versions of BODMAS are summarized in Table. 3. BODMAS 1 and 2 have the highest number of classes, whereas BODMAS 3.1.0 and BODMAS 4.1.0 have the least number of families. Table. 3b shows that datasets containing all sections have a

[2]https://figshare.com/articles/dataset/Malware_Detection_PE-Based_Analysis_Using_Deep_Learning_Algorithm_Dataset/6635642

TABLE 3. Statistics on Datasets.

Dataset	Samples	Classes	Train set (samples)	Dev set (samples)	Test set (samples)
MalImg	9339	25	5995	1486	1858
BODMAS 1	57293	581	37030	8999	11264
BODMAS 2	57292	581	37030	8999	11263
BODMAS 3.0.0	43936	26	28137	7023	8776
BODMAS 3.1.0	22177	22	14209	3542	4426
BODMAS 3.1.1	24017	25	15390	3834	4793
BODMAS 4.0.0	44916	27	28765	7179	8972
BODMAS 4.1.0	23142	23	14827	3696	4619
BODMAS 4.1.1	24982	26	16008	3988	4986
BODMAS 4.0.0 full	44916	27	35944	8972	0

(a) Components of Datasets

Dataset	Preprocessing time (hours)	File Size (GB)
MalImg	0.008	1.11
BODMAS 1	5.299	9.04
BODMAS 2	3.878	7.92
BODMAS 3.0.0	2.404	13.70
BODMAS 3.1.0	1.126	3.14
BODMAS 3.1.1	1.374	3.46
BODMAS 4.0.0	2.426	13.85
BODMAS 4.1.0	1.435	3.30
BODMAS 4.1.1	1.470	3.61
BODMAS 4.0.0 full	2.426	13.85

(b) Metadata of Datasets

preprocessing time less than the original one, and datasets containing ".text" sections consume the least time for the preprocessing step. On the other hand, the MalImg dataset has samples already in the grayscale image format, hence its preprocessing time is only for splitting samples into train-dev and test sets. Even though BODMAS 1 and 2 have the most samples, their disk sizes are not as big as those of datasets with all sections because of the way they are compressed.

3) EVALUATION WITH DIKEDATASET

Models are required to be evaluated by samples that they have never seen. DikeDataset contains 10,841 malware files (PE and OLE) and 1,082 benign files. The benign files are legitimate Windows software downloaded from Cnet, while the malware files are collected from VirusShare and Malicia Project. Unlike BODMAS, DikeDataset has a relatively small number of malware families. The performance of each model is evaluated based on its ability to detect a file as malware or not.

Only PE samples in DikeDataset are used in this evaluation phase. As a result, with scenarios involved with BODMAS 3.0.0 and 4.0.0, there are 8970 malware and 962 benign samples. Whereas, there are 8903 malware and 947 benign samples used for experiments with models trained by BODMAS 3.1.0, 3.1.1, 4.1.0, and 4.1.1.

B. EVALUATION PARAMETERS

There are two types of evaluated parameters. The first one is for the evaluation step in the first phase, and the second one is for assessing the evaluation step with the DikeDataset.

The macro-averaged F1 score is used as the main metric for evaluating the models. As stated in (1d), it is computed based on the unweighted mean of all F1 scores for N classes, where each F1 score is demonstrated via (1c). F1 score is defined as the harmonic mean of precision and recall. Precision is the proportion of positive labels that are correct. As in (1a), precision is calculated as the number of true positives (TPs) over the number of TPs plus the number of false positives (FPs). Recall presents the proportion of actual positives that were labeled precisely, as shown in (1b). Both precision and recall must be high to produce a high F1 score. In the multi-class classification task, F1 score for each class is calculated with the One-vs-rest approach.

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

$$Recall = \frac{TP}{TP + FN} \tag{1b}$$

$$F1\_score = \frac{2PrecisionRecall}{Precision + Recall} \tag{1c}$$

$$macro\_F1\_score = \frac{1}{N}\sum_{i=0}^{N-1} F1\_score_i \tag{1d}$$

The macro F1 score is a useful metric to consider for imbalanced datasets, as it treats all classes equally and can provide insights into the classifier's performance on minority classes.

In the evaluation phase, malware classifiers are used for a binary classification task, and ROC curves are plotted to visualize the model's ability to detect malware. The ROC curve shows the true positive rate (TPR) (x-axis) and false positive rate (FPR) (y-axis) at various classification thresholds, and a good performing model is represented by a curve that progresses from the bottom left to the top right of the plot.

The area under the ROC curve (AUC) is used as the optimizing metric to judge the performance of the malware detector. Despite its limitations with imbalanced datasets, it is still chosen because of the equal concern for predicting malware files and avoiding false positives.

The best threshold T is determined by calculating the G-Mean for each model's prediction on DikeDataset, which compares the classification performance of both majority and minority classes based on TPR and FPR, as shown in (2). TPR is the same as recall, while FPR is the miss rate.

$$G\_mean = \sqrt{TPR(1 - FPR)} \tag{2}$$

Apart from the macro F1 score and AUC score as main evaluation metrics, model loading time and predicting time are the "satisficing" metrics in the decision on choosing the most suitable models for deployment with ESP. In this paper, model loading time and predicting time per sample's upper bounds are 30s and 0.1s, respectively.

C. RESULT OF EVALUATION WITH DIKEDATASET

As described in Section V-A, fine-tuned models were brought to the second phase to be evaluated by DikeDataset. In this evaluation phase, models are compared based on the prediction of whether a file is malware or benign. Therefore, the problem is simplified as a binary classification task.

For each scenario, ROC curves are used for visualizing the performance of models. In each chart, a blue line connecting point (0, 0) and point (1, 1) represents a no-skill classifier. Any line below this line is considered a terrible classifier since it cannot make a prediction better than a random choice. The other line acts as the ROC curve of the classifier. It also has a red point indicating the coordination where the optimal threshold was found. At this point, TPR is high and FPR is low enough to obtain the highest value for G-Mean.

For each main version of the selected datasets, a table is provided with various metrics recorded from the prediction stage for the comparison between Mobilenet V2 and Inception V3. This table not only outputs the AUC but also shows the optimal threshold and other metrics captured based on this threshold. In detail, the threshold is used by classifiers to make a decision on whether a file is malware or not. If the output probability for a malware class is higher than this threshold, the classifier will tag it as malicious.

1) MALIMG

In the section V-A, models fine-tuned with MalImg were chosen as base line models. Hence, their evaluations were taken first. Figure 12 depicts the ROC curves of two selected models. Both models achieved ROC curves on the left-hand side of the blue line. This proves that these baseline models perform better than random choices. On the other hand, the area under the curve is bigger in the case of Inception V3 compared to the other cases.

Specific details are listed in table 4. The optimal thresholds of Mobilenet V2 and Inception V3 are found to be 0.8299 and 0.7301, respectively. The G-Mean calculated from this threshold is also presented in the table. In this scenario, Inception V3 obtained a higher G-Mean, FPR, Accuracy, Precision and F1 score. However, Mobilenet V2 also performed better than the other one in terms of TPR and Recall. Overall, although Inception V3 got the better AUC, which was 0.6558, it is still not considered a good classifier because its AUC is too far from the ideal AUC, which is 1.

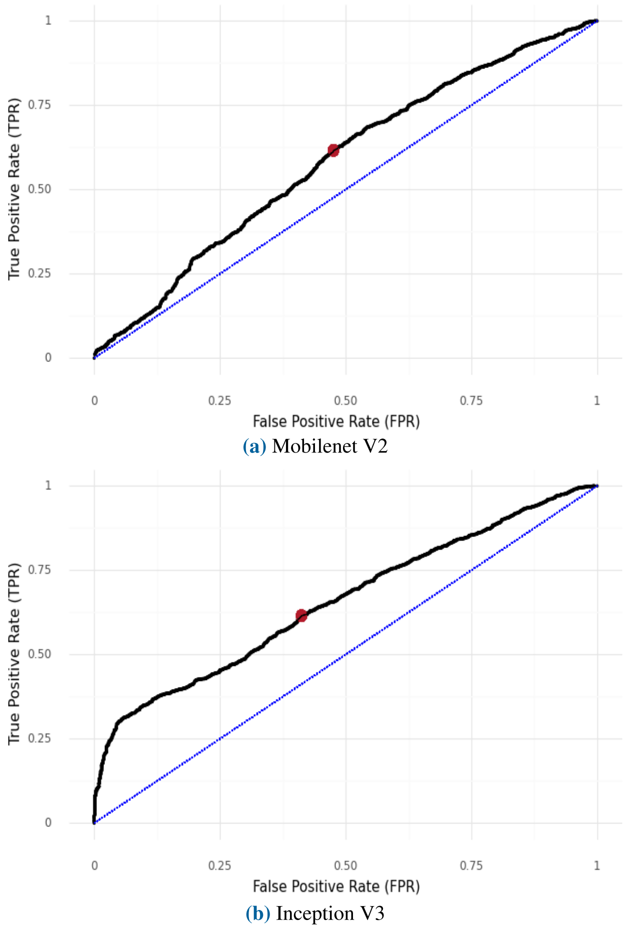


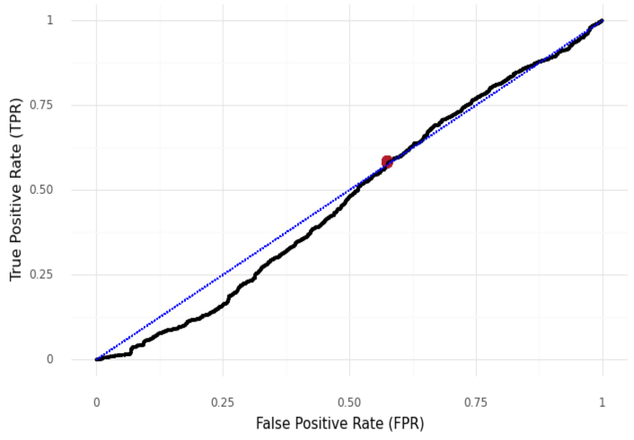
FIGURE 12. ROC curve of mobilenet V2 and inception V3 trained with malimg in evaluation phase with DikeDataset.

TABLE 4. Metric values of Mobilenet V2 and Inception V3 trained with MalImg in Evaluation Phase with DikeDataset.

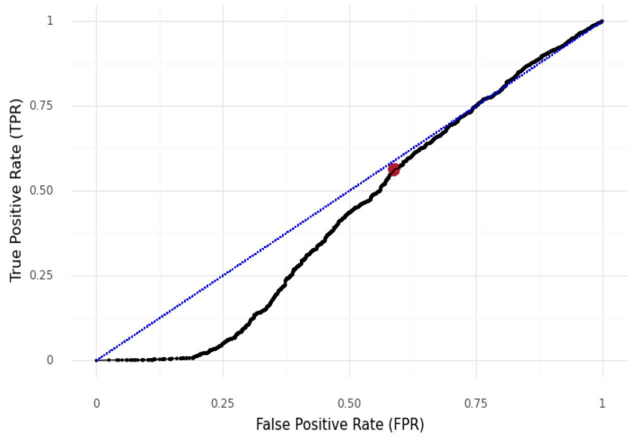
Metrics	Mobilenet V2	Inception V3
AUC	0.5838	**0.6558**
Threshold	0.8299	0.7301
G-Mean	0.5682	**0.6014**
TPR	**0.6161**	0.6153
FPR	0.476	**0.4122**
Accuracy	0.6076	**0.6128**
Precision	0.9284	**0.9373**
Recall	**0.6159**	0.6153
F1 Score	0.7405	**0.7429**

2) BODMAS 3

Figure 13 depicts the bad performance of both chosen models. In both experiments, the ROC curve is under the no-skill line. Any points located beneath the blue line have an extremely low TPR. As can be seen, Inception V3 had a bigger region than Mobilenet V2. This means it does not have the ability to recognize malware.



(a) Mobilenet V2



(b) Inception V3

FIGURE 13. ROC curve of Mobilenet V2 and Inception V3 trained with BODMAS 3.0.0 in Evaluation Phase with DikeDataset.

As shown in table 5, AUC scores of both models fine-tuned with BODMAS 3.0.0 are lower than 0.5. Contrary to the previous scenario, all metrics of Mobilenet V2 based on the found optimal threshold are all better than Inception V3. Even so, its FPR is about 0.57, which is too high for a good malware detector.

Similar results happened to the models fine-tuned with BODMAS 3.1.0 and 3.1.1. The models in this scenario had poor performance with DikeDataset. Both Mobilenet V2 and Inception V3 obtained an AUC that was lower than 0.5, and their optimal thresholds were also found to be pretty high. Other metrics based on the optimal threshold are also around 0.5, as in table 5. Overall, for all models fine-tuned with all subversions of BODMAS 3, Mobilenet V2 models attained pretty close metric values to their Inception V3 corresponding models.

3) BODMAS 4

In experiments with models fine-tuned with subversions of BODMAS 4, the recorded AUC scores in table 6 are relative good compared to previous scenarios.

TABLE 5. Metric values of Mobilenet V2 and Inception V3 trained with BODMAS 3 in Evaluation Phase with DikeDataset.

Metrics	BODMAS 3.0.0		BODMAS 3.1.0		BODMAS 3.1.1	
	(1)	(2)	(1)	(2)	(1)	(2)
AUC	**0.4751**	0.4298	0.4698	**0.479**	**0.4957**	0.4732
Threshold	0.6709	0.7839	0.9612	0.8961	0.842	0.982
G-Mean	**0.4981**	0.4814	**0.512**	0.4913	0.5019	**0.5028**
TPR	**0.5835**	0.563	**0.5695**	0.4591	**0.4868**	0.4458
FPR	**0.5748**	0.5884	0.5396	**0.4741**	0.4826	**0.4329**
Accuracy	**0.5682**	0.5483	**0.5589**	0.4654	**0.4895**	0.4571
Precision	**0.9044**	0.8992	**0.9084**	0.901	0.9046	**0.9063**
Recall	**0.5835**	0.563	**0.5694**	0.4589	**0.4866**	0.4454
F1 Score	**0.7094**	0.6924	**0.7**	0.6081	**0.6328**	0.5972

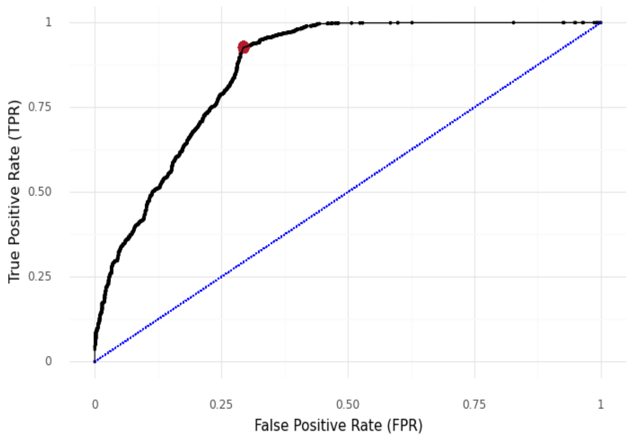
Note: (1) Mobilenet V2, (2) Inception V3

With the help of the benign class in the dataset, models trained with BODMAS 4.0.0 gained high AUC scores: 0.8615 and 0.9392 for Mobilenet V2 and Inception V3, respectively. Although most of Inception V3's recorded metrics based on the found optimal threshold are higher than Mobilenet V2's, its optimal threshold is 0.1817, which is quite low. On the other hand, Mobilenet V2 achieved a 0.4995 optimal threshold, which might bring balance in classifying a sample as malware.

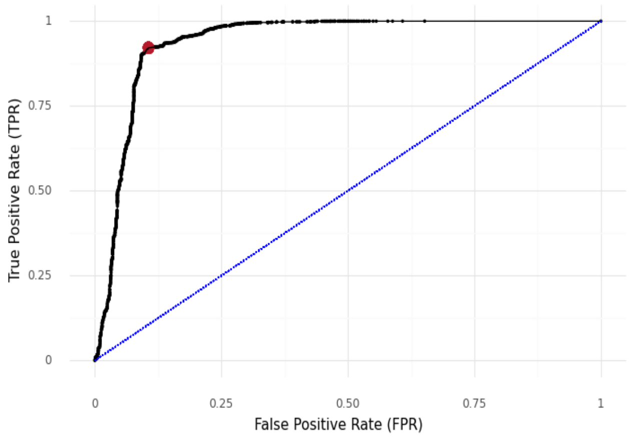
As illustrated in figure 14, ROC curves of both models in this scenario were found to be better than models' in all previous scenarios, even the baseline models'. Both the ROC curves of the selected models are far higher than the no-skill line. In particular, Inception V3 has its upper left corner (the red point) close to the ideal (0, 1) point.

Models fine-tuned with BODMAS 4.1.0 and 4.1.1 also obtained better AUC scores than the baseline models'. For instance, the AUC scores of Inception V3 trained with BODMAS 4.1.0 and 4.1.1 are 0.8763 and 0.825, while the same model trained with MalImg only got 0.6558. On the other hand, as shown by table 6, G-mean of Inception V3 trained with BODMAS 4.1.0 based on the optimal threshold was saved as 0.8109, which is much better than Mobilenet V2's (0.678). However, Mobilenet V2 showed its ability to catch up with Inception V3 when its AUC and other metrics were quite near the other one's. Moreover, the threshold of Mobilenet V2 trained with BODMAS 4.1.0 is 0.5054, which is considered safer when predicting malware. In fact, with a small threshold like the one Inception V3 had in this scenario (0.1939), the detector could get in trouble when it meets an unseen benign file.

In this paper, an extra scenario has been added to measure the ability of BODMAS 4.0.0 models. In particular, models are trained with BODMAS 4.0.0 full (BODMAS 4.0.0 without the test set). With more samples per class, the AUC scores of both models in this scenario were found to be greater than 0.9, which totally outperformed other experimented models. Moreover, other metrics based on the optimal threshold were discovered to be much higher than in other scenarios. For example, the best FPR in this scenario is 0.0634, while the best FPR in the baseline scenario is 0.4122.



(a) Mobilenet V2



(b) Inception V3

FIGURE 14. ROC curve of Mobilenet V2 and Inception V3 trained with BODMAS 4.0.0 in Evaluation Phase with DikeDataset.

However, the optimal thresholds in this scenario are very small. In particular, the optimal thresholds of Mobilenet V2 and Inception V3 are 0.1974 and 0.0441, respectively. The cause of these small thresholds is that fine-tuned models have overfitted the dataset, especially the benign samples in the dataset. As mentioned earlier, with a small threshold, the malware detector might output the wrong class for an unseen normal user application. Therefore, to make use of these models, a great number of benign classes must be fed to the detector for continued learning.

Overall, the evaluation phase brings up various experiments for models fine-tuned with different datasets. Models learned with benign samples outperformed other models, especially the baseline models. On the other hand, models learned with ‘‘.text’’ related sections also produced quite good performance in predicting malware. This shows the ability to combine these models for further deployment in the future.

D. SUMMARY

Through experiments, Mobilenet V2 has successfully shown its potential for classifying malware. It not only achieved a macro F1 score in the training and an AUC score in the evaluation that was close to Inception V3's, but also consumed less disk space and outperformed Inception V3 in terms of prediction time. Mobilenet V2 fine-tuned with BODMAS 4.0.0 is better than other experimental models and is chosen to be integrated with ESP. On the other hand, Inception V3 fine-tuned with BODMAS 4.0.0 could be used on the server to achieve high performance but still not affect any user machine.

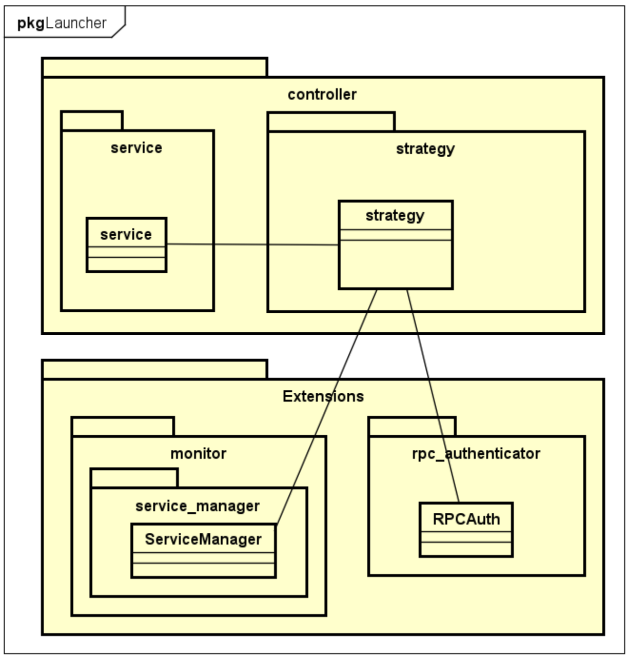


FIGURE 15. Package design of launcher.

VI. APPLICATION BUILDING

A. PACKAGE DESIGN

In DOPS, the Launcher, Scanner, and DOCG are written in Golang. In Go, each folder on the project is considered a package. The Launcher is designed as shown in Fig. 15. The core component is the strategy package, which is responsible for controlling how the Launcher is run. When the Launcher is configured to run in service mode, the strategy package initiates the RPCAuth, opens the port, and waits for the Scanner to connect via the RPC[3] method. Then it calls the ServiceManager to start the Scanner service and can also install or uninstall the Scanner service if necessary. If the Launcher is configured with the Windows Task Scheduler, the service package is called by the strategy package.

As shown in Fig. 16, the Scanner contains two major components: the core and the plugin. Controller package in Scanner aims to find the right strategy to run plugins. In service mode, ServiceStrategy will initialize the data for components in the data package. Then it will

[3]https://www.ibm.com/docs/en/aix/7.1?topic=concepts-remote-procedure-call

TABLE 6. Metric values of Mobilenet V2 and Inception V3 trained with BODMAS 4 in Evaluation Phase with DikeDataset.

Metrics	BODMAS 4.0.0		BODMAS 4.1.0		BODMAS 4.1.1		BODMAS 4.0.0 full	
	(1)	(2)	(1)	(2)	(1)	(2)	(1)	(2)
AUC	0.8615	**0.9392**	0.7652	**0.8763**	0.7373	**0.825**	0.9176	**0.9851**
Threshold	0.4995	0.1817	0.5054	0.1939	0.3819	0.2607	0.1974	0.0441
G-Mean	0.809	**0.9076**	0.678	**0.8109**	0.6813	**0.7943**	0.8772	**0.951**
TPR	**0.9272**	0.9215	0.621	**0.7843**	0.7738	**0.9123**	0.9336	**0.9656**
FPR	0.2942	**0.106**	0.2598	**0.1616**	0.4002	**0.3083**	0.1757	**0.0634**
Accuracy	0.9057	**0.9187**	0.6325	**0.7895**	0.7571	**0.8911**	0.9229	**0.9627**
Precision	0.9671	**0.9878**	0.9574	**0.9786**	0.9479	**0.9653**	0.9802	**0.993**
Recall	**0.9271**	0.9214	0.621	**0.7843**	0.7738	**0.9123**	0.9334	**0.9656**
F1 Score	0.9467	**0.9535**	0.7534	**0.8708**	0.852	**0.938**	0.9563	**0.9791**

Note: (1) Mobilenet V2, (2) Inception V3

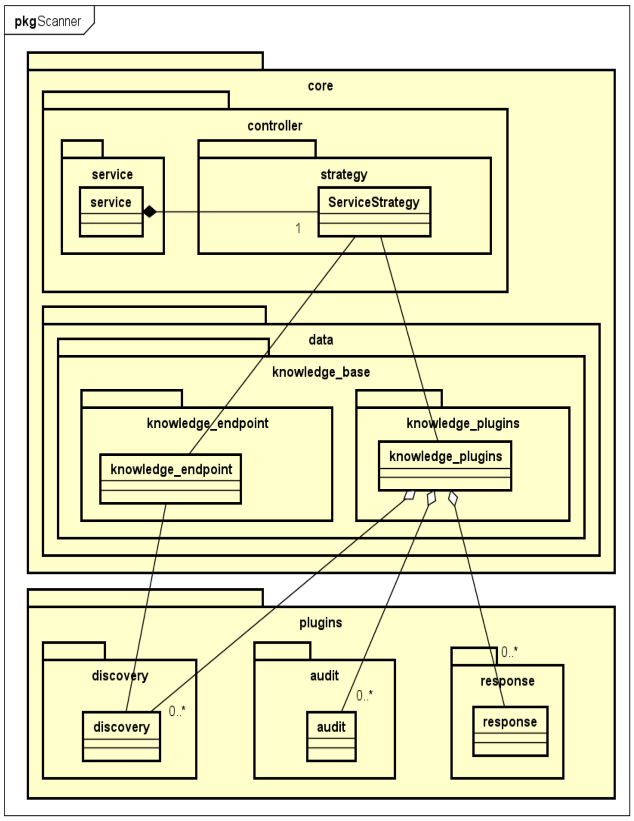


FIGURE 16. Package design of scanner.

select a predefined list of plugins, which are stored in the knowledge_endpoint package, to start scanning.

The data package contains two sub-packages: knowledge_endpoint and knowledge_plugins. The first sub-package stores initialized data about the machine, which is filled by discovery plugins. The second sub-package holds the list of all plugins used in the project. The core package takes care of monitoring, and the plugins package contains all the specific plugins used for sweeping purposes. The discovery package contains plugins used for discovering user mode information about the machine; the audit package contains plugins for scanning vulnerabilities; and the response plugin takes responsibility for receiving commands from administrators and responding to those requests.

B. APPLICATION BUILDING

1) ACHIEVEMENT

The core idea of DOPS is the ESP, which provides a process for continuously monitoring endpoints in an organization's network and executing any protective action. ESP includes four main executables: launcher.exe (Launcher), scanner.exe (Scanner), DOMalwareDetector.exe (DOMD), DOClientGo.exe (DOCG), and a kernel driver: deocdrv.sys (DODriver).

To give an overview of the workload for this project, statistics about each component are summarized in Table. 7.

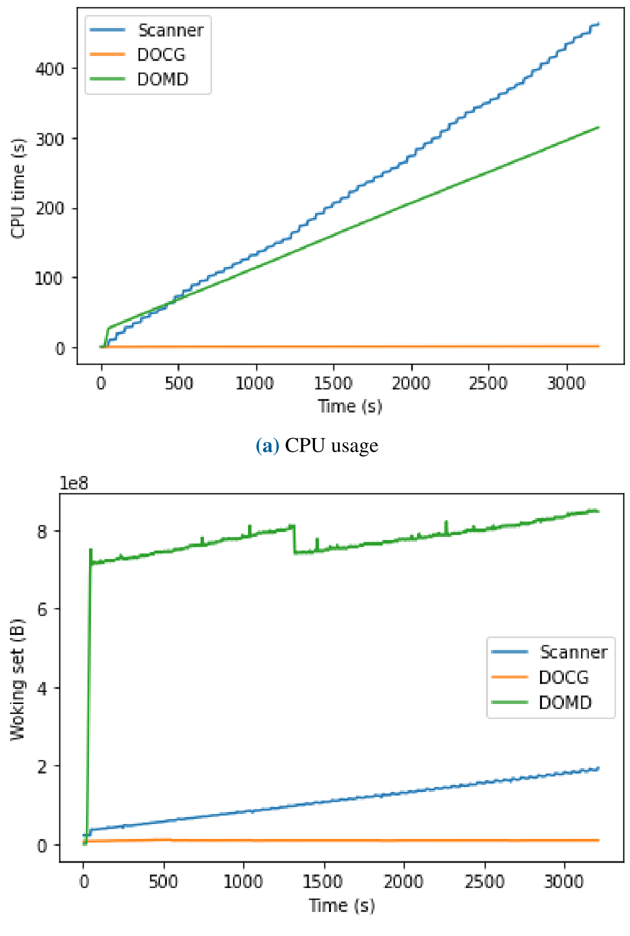
2) DEPLOYMENT

The deployment stage of the DOPS system requires attention from administrators and the blue team. The ESP can be developed by third-party security or the organization's security team, but the protocol between the ESP and DOPS must be unified. ESP must be installed on every user machine to be monitored, and the administrators must start ESP in service mode with an authentication key to prevent its use outside the organization's scope. Additionally, all components can be automatically started on the next run of the machine by running Launcher with the authentication key.

The Launcher and DODriver are configured to auto-run, and the latter is deployed in test signing mode. To deploy a kernel driver on a real machine, the driver must be signed with a paid certificate. However, the system was deployed on a virtual machine running Windows 10 Pro (8GB RAM) with test signing mode, which means that driver signature enforcement is disabled and Microsoft Defender Antivirus's real-time protection is turned off. The REST API used for Scanner in the data transferring step was https://api.cystack.net/v3/endpoint.

TABLE 7. Code statistics of ESP.

Component	Code Lines	Code Size (kB)	Code File Statistics	Code Files	Executables Size (MB)
Scanner	8875	384	96 .go	96	26.6
Launcher	728	23	13 .go	13	8.74
Common	238	8	5 .go	5	
DOCG	652	36	17 .go	17	6.19
Keras Model					35.1
DOMD	344	17	13 .py	13	516
DODriver	767	38	8 .cpp, 10 .h	18	0.02
Scripts	10	0	2 .cmd	2	0.002
Overall	11614	506	131 .go, 13 .py, 8 .cpp, 10 .h, 2 .cmd	164	557.552



(a) CPU usage

(b) Memory usage

FIGURE 17. CPU usage and memory usage of scanner, DOCG and DOMD in one hour.

In the simulation, Scanner's discovery, audit, and response plugins were scheduled to run at different intervals, and Scanner was configured to send process notifications, malware scanning results, and requests to DOMD at specific intervals. In this paper, Scanner only looked up "Download" folders instead of using a filter driver to receive new notifications about programs. The performance of each ESP component was measured in terms of CPU and memory usage for one hour using a tool available in DOPS's side project, which estimated process resources every five seconds.

Fig. 17 shows the CPU and memory usage of each component of ESP over an hour. Scanner's CPU usage increased significantly due to multiple goroutines continuously gathering and sending data to the server. DOMD's CPU usage expanded gradually as it received requests from Scanner and produced predictions. DOCG's CPU time was minimal because it only transferred data. Launcher had the lowest CPU usage and memory usage. DOMD required the first twenty seconds to be loaded successfully, then took up nearly 686 MB to start operating properly. Scanner's memory usage gradually increased over time, while DOMD's memory usage was 847 MB, which is below the expected memory usage.

VII. CONCLUSION

This paper presents a basic implementation of an EDR system called DOPS, which includes the Endpoint Service Pack (ESP), consisting of several components, as well as an image-based malware detection system (DOMD). A mini-project written in Golang has been included as a public contribution, along with the fine-tuning of two CNN networks (Mobilenet V2 and Inception V3) using MalImg and BODMAS datasets. Through simulations, Mobilenet V2 fine-tuned with BODMAS 4.0.0 was found to be the optimal image-based malware classifier, achieving a high AUC score (0.8615) while also meeting the model loading time and predicting time requirements for deployment with ESP.

This paper also describes the construction of an EDR system. However, ESP still needs improvement in dealing with real-life attacks and lacks a process for defending against advanced persistent threats. Furthermore, the image-based malware classification is a static malware analysis, and ESP's protection ability can be enhanced with the combination of both static and dynamic malware detection. In terms of performance, ESP needs better memory management, and DOMD consumes a lot of disk space and physical memory. In the future, ESP requires more applications from the microservice architecture to deal with the expansion of each existing service and new components.

From an academic perspective, image-based malware detection should be analyzed in more detail, along with its integration with dynamic malware detection. The Divide and Merge strategy [42] can also be applied to various sections of the PE file to improve the model's accuracy. Overall, further efforts will be made to evaluate DOPS in a real-life scenario.

REFERENCES

[1] C. Nwosu, "Visualizing the 50 biggest data breaches from 2004–2021," Visual Capitalist, British Columbia, Canada, Jun. 2022. [Online]. Available: https://www.visualcapitalist.com/cp/visualizing-the-50-biggest-data-breaches-from-2004-2021/

[2] J. P. Mello, "Data breaches affected nearly 6 billion accounts in 2021," TechNewsWorld, California, USA, Tech. Rep., Jan. 2022. [Online]. Available: https://www.technewsworld.com/story/data-breaches-affected-nearly-6-billion-accounts-in-2021-87392.html

[3] IBM. (Jul. 2022). *Cost of a Data Breach Report 2022*. [Online]. Available: https://www.ibm.com/security/data-breach

[4] E. G. Franco, M. Kuritzky, R. Lukacs, and S. Zahidi, "The global risks report 2022," World Economic Forum, Cologny, Switzerland, Tech. Rep. 978-2-940631-09-4, 2022. [Online]. Available: https://www3.weforum.org/docs/WEF_The_Global_Risks_Report_2022.pdf

[5] Positive Technologies. (Dec. 2021). *Business in the Crosshairs: Analyzing Attack Scenarios*. [Online]. Available: https://www.ptsecurity.com/ww-en/analytics/pentests-2021-attack-scenarios/

[6] G. Karantzas and C. Patsakis, "An empirical assessment of endpoint detection and response systems against advanced persistent threats attack vectors," *J. Cybersecur. Privacy*, vol. 1, no. 3, pp. 387–421, Jul. 2021.

[7] A. Chuvakin. (Jul. 2013). *Named: Endpoint Threat Detection & Response*. [Online]. Available: https://blogs.gartner.com/anton-chuvakin/2013/07/26/named-endpoint-threat-detection-response/

[8] AV-TEST. *Malware*. Accessed: Feb. 11, 2023. [Online]. Available: https://www.av-test.org/en/statistics/malware/

[9] L. Nataraj, S. Karthikeyan, G. Jacob, and B. Manjunath, "Malware images: Visualization and automatic classification," in *Proc. 8th Int. Symp. Vis. Cyber Secur.*, Jul. 2011, pp. 1–7.

[10] L. Yang, A. Ciptadi, I. Laziuk, A. Ahmadzadeh, and G. Wang, "BODMAS: An open dataset for learning based temporal analysis of PE malware," in *Proc. IEEE Secur. Privacy Workshops (SPW)*, May 2021, pp. 78–84.

[11] W. U. Hassan, A. Bates, and D. Marino, "Tactical provenance analysis for endpoint detection and response systems," in *Proc. IEEE Symp. Secur. Privacy (SP)*, May 2020, pp. 1172–1189.

[12] H. Kaur and R. Tiwari, "Endpoint detection and response using machine learning," *J. Phys., Conf. Ser.*, vol. 2062, no. 1, Nov. 2021, Art. no. 012013.

[13] J.-Y. Kim, S.-J. Bu, and S.-B. Cho, "Malware detection using deep transferred generative adversarial networks," in *Proc. Int. Conf. Neural Inf. Process.*, Oct. 2017, pp. 556–564.

[14] R. Kath, "The portable executable file format from top to bottom," Microsoft Developer Netw. Technol. Group, Redmond, Washington, USA, Tech. Rep., 1997. [Online]. Available: http://www.csn.ul.ie/~caolan/pub/winresdump/winresdump/doc/pefile2.html

[15] G. Conti, S. Bratus, A. Shubina, A. Lichtenberg, R. Ragsdale, R. Perez-Alemany, B. Sangster, and M. Supan, "A visual study of primitive binary fragment types," Tech. Rep., Jun. 2011.

[16] M. J. Awan, O. A. Masood, M. A. Mohammed, A. Yasin, A. M. Zain, R. Damaševičius, and K. H. Abdulkareem, "Image-based malware classification using VGG19 network and spatial convolutional attention," *Electronics*, vol. 10, no. 19, p. 2444, Oct. 2021.

[17] H. Naeem, F. Ullah, M. R. Naeem, S. Khalid, D. Vasan, S. Jabbar, and S. Saeed, "Malware detection in industrial Internet of Things based on hybrid image visualization and deep learning model," *Ad Hoc Netw.*, vol. 105, Aug. 2020, Art. no. 102154.

[18] V. Moussas and A. Andreatos, "Malware detection based on code visualization and two-level classification," *Information*, vol. 12, no. 3, p. 118, Mar. 2021.

[19] J. Hemalatha, S. Roseline, S. Geetha, S. Kadry, and R. Damaševičius, "An efficient DenseNet-based deep learning model for malware detection," *Entropy*, vol. 23, no. 3, p. 344, Mar. 2021.

[20] W. El-Shafai, I. Almomani, and A. AlKhayer, "Visualized malware multi-classification framework using fine-tuned CNN-based transfer learning models," *Appl. Sci.*, vol. 11, no. 14, p. 6446, Jul. 2021.

[21] E. Rezende, G. Ruppert, T. Carvalho, F. Ramos, and P. de Geus, "Malicious software classification using transfer learning of ResNet-50 deep neural network," in *Proc. 16th IEEE Int. Conf. Mach. Learn. Appl. (ICMLA)*, Dec. 2017, pp. 1011–1014.

[22] A. Kumar, K. P. Sagar, K. S. Kuppusamy, and G. Aghila, "Machine learning based malware classification for Android applications using multimodal image representations," in *Proc. 10th Int. Conf. Intell. Syst. Control (ISCO)*, Jan. 2016, pp. 1–6.

[23] P. Yosifovich, M. E. Russinovich, D. A. Solomon, and A. Ionescu, *Windows Internals, Part 1: System Architecture, Processes, Threads, Memory Management, and More*, 7th ed. Redmond, WA, USA: Microsoft Press, 2017.

[24] P. Yosifovich, "Windows kernel programming," Leanpub, Tech. Rep., 2019. [Online]. Available: https://leanpub.com/windowskernelprogramming

[25] J. Kim, O. Sangjun, Y. Kim, and M. Lee, "Convolutional neural network with biologically inspired retinal structure," *Proc. Comput. Sci.*, vol. 88, pp. 145–154, Dec. 2016.

[26] R. H. R. Hahnloser, R. Sarpeshkar, M. A. Mahowald, R. J. Douglas, and H. S. Seung, "Digital selection and analogue amplification coexist in a cortex-inspired silicon circuit," *Nature*, vol. 405, no. 6789, pp. 947–951, Jun. 2000.

[27] M. Sandler, A. Howard, M. Zhu, A. Zhmoginov, and L.-C. Chen, "MobileNetV2: Inverted residuals and linear bottlenecks," in *Proc. IEEE/CVF Conf. Comput. Vis. Pattern Recognit.*, Jun. 2018, pp. 4510–4520.

[28] C. Szegedy, V. Vanhoucke, S. Ioffe, J. Shlens, and Z. Wojna, "Rethinking the inception architecture for computer vision," in *Proc. IEEE Conf. Comput. Vis. Pattern Recognit. (CVPR)*, Jun. 2016, pp. 2818–2826.

[29] F. Hayes-Roth, D. A. Waterman, and D. B. Lenat, *Building Expert Systems*. Reading, MA, USA: Addison-Wesley Longman Publishing Co., 1983.

[30] M. Stamp, M. Alazab, and A. Shalaginov, *Malware Analysis Using Artificial Intelligence and Deep Learning*. New York, NY, USA: Springer, Jan. 2021.

[31] 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.

[32] 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.

[33] P. Prajapati and M. Stamp, "An empirical analysis of image-based learning techniques for malware classification," 2021, *arXiv:2103.13827*.

[34] A. Bensaoud, N. Abudawaood, and J. Kalita, "Classifying malware images with convolutional neural network models," 2020, *arXiv:2010.16108*.

[35] J. Deng, W. Dong, R. Socher, L.-J. Li, K. Li, and L. Fei-Fei, "ImageNet: A large-scale hierarchical image database," in *Proc. IEEE Conf. Comput. Vis. Pattern Recognit.*, Jun. 2009, pp. 248–255.

[36] S. Phiphiphatphaisit and O. Surinta, "Food image classification with improved MobileNet architecture and data augmentation," in *Proc. 3rd Int. Conf. Inf. Sci. Syst.*, Mar. 2020, pp. 51–56.

[37] A. Kumar, K. P. Sagar, K. S. Kuppusamy, and G. Aghila, "Machine learning based malware classification for Android applications using multimodal image representations," in *Proc. 10th Int. Conf. Intell. Syst. Control (ISCO)*, Jan. 2016, pp. 1–6.

[38] Y. Lu and J. Li, "Generative adversarial network for improving deep learning based malware classification," in *Proc. Winter Simulation Conf. (WSC)*, Dec. 2019, pp. 584–593.

[39] H. S. Anderson and P. Roth, "EMBER: An open dataset for training static PE malware machine learning models," 2018, *arXiv:1804.04637*.

[40] R. Harang and E. M. Rudd, "SOREL-20M: A large scale benchmark dataset for malicious PE detection," 2020, *arXiv:2012.07634*.

[41] S. Sriram, R. Vinayakumar, V. Sowmya, M. Alazab, and K. P. Soman, "Multi-scale learning based malware variant detection using spatial pyramid pooling network," in *Proc. IEEE Conf. Comput. Commun. Workshops (INFOCOM WKSHPS)*, Jul. 2020, pp. 740–745.

[42] J. Singh, D. Thakur, F. Ali, T. Gera, and K. S. Kwak, "Deep feature extraction and classification of Android malware images," *Sensors*, vol. 20, no. 24, p. 7013, Dec. 2020.

TRAN HOANG HAI received the B.S. degree from the Hanoi University of Science and Technology, Vietnam, the M.S. degree in computer engineering from Kyung Hee University, South Korea, in 2008, and the Ph.D. degree in computer science from the University of Rennes 1, France, in 2012. Since 2008, he has been with INRIA joint Alcatel-Lucent Bell Laboratory. He is currently an Assistant Professor with the Department of Data Communication and Computer Networks, School of Information and Communication Technology, Hanoi University of Science and Technology. His research interests include network security, routing, and resource allocation mechanisms in the next-generation internet, and applied game theory to communication networks. He has published several papers on those issues.

VU VAN THIEU received the Ph.D. degree in computer science from the Leiden Institute of Advanced Computer Science, Leiden University, The Netherlands, in 2012. He is currently with the Computer Science Department, School of Information and Communication Technology (SoICT), Hanoi University of Science and Technology (HUST), Vietnam. His research interests include AI and high-performance computing.

TRAN THAI DUONG is currently pursuing the degree in information technology with the Hanoi University of Science and Technology. With a background in cyber security and a moderate foundation in developing Windows kernel drivers, he is currently contributing as a Project Manager and a Backend Engineer. He aspires to be an inventor, striving to create better systems that serve humanity. Through continuous learning and dedication, he hopes to make a positive impact in the field and contribute to the advancement of technology for the benefit of all.

HONG HOA NGUYEN is currently pursuing the bachelor's degree with the Hanoi University of Science and Technology, Vietnam. She is working on the bachelor's thesis on malware detection using deep learning models. Her research interests include network security and reverse engineering.

EUI-NAM HUH (Member, IEEE) received the B.S. degree from Busan National University, South Korea, the master's degree in computer science from The University of Texas, USA, in 1995, and the Ph.D. degree from Ohio University, USA, in 2002. He was an Assistant Professor with Sahmyook University and Seoul Women's University, South Korea. He is currently the Director of the Real-Time Mobile Cloud Research Center. He is also a Professor with the Department of Computer Science and Engineering, Kyung Hee University, South Korea. His research interests include cloud computing, screen content coding (cloud streaming), the Internet of Things, distributed real-time systems, security, and big data. He is the Chair of the Cloud/BigData Special Technical Committee for the Telecommunications Technology Association (TTA) and a Korean national standards body of ITUT SG13 and ISO/IEC SC38.