PROUD-MAL: static analysis-based progressive framework for deep unsupervised malware classification of windows portable executable

Syed Khurram Jah Rizvi^{1,3} · Warda Aslam¹ · Muhammad Shahzad¹ · Shahzad Saleem^{1,4} · Muhammad Moazam Fraz^{1,2}

Received: 30 January 2021 / Accepted: 24 September 2021 / Published online: 12 October 2021
© The Author(s) 2021

Abstract

Enterprises are striving to remain protected against malware-based cyber-attacks on their infrastructure, facilities, networks and systems. Static analysis is an effective approach to detect the malware, i.e., malicious Portable Executable (PE). It performs an in-depth analysis of PE files without executing, which is highly useful to minimize the risk of malicious PE contaminating the system. Yet, instant detection using static analysis has become very difficult due to the exponential rise in volume and variety of malware. The compelling need of early stage detection of malware-based attacks significantly motivates research inclination towards automated malware detection. The recent machine learning aided malware detection approaches using static analysis are mostly supervised. Supervised malware detection using static analysis requires manual labelling and human feedback; therefore, it is less effective in rapidly evolutionary and dynamic threat space. To this end, we propose a progressive deep unsupervised framework with feature attention block for static analysis-based malware detection (PROUD-MAL). The framework is based on cascading blocks of unsupervised clustering and features attention-based deep neural network. The proposed deep neural network embedded with feature attention block is trained on the pseudo labels. To evaluate the proposed unsupervised framework, we collected a real-time malware dataset by deploying low and high interaction honeypots on an enterprise organizational network. Moreover, endpoint security solution is also deployed on an enterprise organizational network to collect malware samples. After post processing and cleaning, the novel dataset consists of 15,457 PE samples comprising 8775 malicious and 6681 benign ones. The proposed PROUD-MAL framework achieved an accuracy of more than 98.09% with better quantitative performance in standard evaluation parameters on collected dataset and outperformed other conventional machine learning algorithms. The implementation and dataset are available at <https://bit.ly/35Sne3a>.

Keywords Unsupervised classification · Progressive learning · Malware detection · Static analysis · Feature attention

Introduction

Cybersecurity has become an irresistible concern for enterprises across the globe keeping in view the sensitivity of the information as most valuable asset. In this information age, organizations are facing an ever expanding and sophisticated

malware-based cyber threat spectrum. The Malware is an abbreviated form of “malicious software” and it is a set of instructions intended to bring fatal damages to enterprises, infrastructure, industrial processes and digital systems. It is a lethal cyber weapon used for unauthorized access, cyber espionage, cyber terror, identity theft, data exfiltration or corruption, service interruption or failure, data hostage for ransom etc. In 2018, more than 430 million unique samples of malware were detected in with an annual increase of 36% [1]. The significant annual increase of 25% and 1000% is also observed in use of malwares and malicious PowerShell scripts, respectively. According to Kaspersky Labs (2019), more than 100 million different hosts were attacked by mid-2019. In 2018, more than 889,452 internet banking users were targeted by of banking Trojans with an increase of 15.9% in comparison with previous year. Accord-

✉ Syed Khurram Jah Rizvi
srizvi.phd15seecs@seecs.edu.pk; syed.rizvi@warwick.ac.uk

¹ National University of Sciences and Technology (NUST), Islamabad, Pakistan

² The Alan Turing Institute, London NW1 2DB, UK

³ Department of Computer Science, University of Warwick, Coventry CV47AL, UK

⁴ Department of Computer Science, University of Jeddah, Jeddah, Kingdom of Saudi Arabia

ing to a recent analysis by Juniper Research, the financial impact of data breaches will increase by 11% per year and will reach a level from \$3 trillion to \$5 trillion in 2024. Therefore, it is the utmost requirement of every business to protect its information-based assets, since even a single attack can result in critical data loss. There are several classes of malware including [11] Ransomware [11], Trojan [14], Key Logger [3], Backdoor [21], Launcher [13], Remote Access toolkits (RAT) [33], Spam-Sending malware [34] etc. The approaches for Malware detection are either signature-based [2] or behavior-based [31]; while first approach is good for identification of known attacks without producing an overwhelming false alarm [3] but requires frequent manual updates of the database with rules and signatures. On the other hand, later approach can be used to generalize signatures related to host and network used to identify the presence of an unwanted piece of code or activity on victim computers or networks. The use of packers [46], encryption [5], polymorphism [31] and obfuscation [28] techniques can easily bypass signature-based detection as they only perform pattern or string matching [11]. Behavior-based [36] approaches that focus on pattern identification including file activity, registry activity and API call [8]; are either based upon static [7] or dynamic analysis [6]. The latter form of analysis requires execution of the malicious code [35] in a controlled setup, i.e., sandbox and is often slow, resource intensive and not suitable for the deployment in the production environment which are also discussed in by [22]. Moreover, due to geometric rise in zero-day malware, existing approaches have become less efficient for detection of zero-day attacks and there is a dire need of automated malware detection and classification system equipped with the machine learning techniques [9]. The machine learning can be either supervised or unsupervised, i.e., supervised learning or discriminative deep architectures conducts the training over labelled data, i.e., classification, regression or predictive analytics whereas unsupervised learning or so called generative architectures draws inferences from datasets consisting of input data without labels [43].

Keeping in view the ongoing huge growth in number of malwares, time-based complexity for malware analysis, acute number of domain experts and demand of earliest detection, considerable research on machine-learning-based techniques is being conducted for automated malware analysis and classification [10, 19] but most of the static analysis-based approaches are supervised in nature. The availability of updated malware dataset along with the labels is also a major hurdle for malware analysts. The aforementioned limitations and gap motivated the development of automated unsupervised malware analysis system for investigation of portable executable to make a classification decision based on static analysis. Moreover, it is essential to have a suitable representation of feature vectors to make decision regarding

malware classification. This paper proposed a progressive deep unsupervised framework (PROUD-MAL) for classifying Windows PE using static analysis of executables. The major contributions are described as follows:

- (a) The purpose of research is to present a framework for unsupervised classification of Portable Executables (PEs) using static features. We term this framework as PROUD-MAL To this end, we propose a two-phase cascaded formulation of progressive unsupervised clustering followed by an attention-based deep neural network for static feature-based malware classification.
- (b) Moreover, it is worth mentioning that attention models have shown promising outputs in various domains such as image analysis and natural language processing but to best of our knowledge, there is no research on applying the attention-based mechanism for malware classification using unsupervised clustering over static features. To this end, we propose a Feature Attention-based Neural Network (FANN) architecture for malware classification. The Attention Block (AB) considers correlation of a feature to target or other features, besides considering feature weight. It puts relatively more weights to feature that contributed more to minimize the validation loss and maximize the classification accuracy.
- (c) We also collected a novel real-time malware dataset comprises 15,457 (25 GB) PE samples collected over a period of seven (07) months (200 days). The novel dataset is collected by deploying low and high interaction honeypots as well as enterprise endpoint security solution over a large organizational network. It is available publicly for research community.
- (d) The quantitative assessment reflects that the proposed model achieved superior performance and outperformed state-of-the-art supervised approaches as well unsupervised one. The high yield of classification accuracy demonstrated the significance and utility of the proposed framework.

The rest of the paper organization is as follows: section “[Background and context](#)” describes background and structure of windows-based PEs. Section “[Related work](#)” narrates the related work. Section “[Methodology and architecture](#)” describes the dataset acquisition, data pre-processing, feature extraction and proposed framework, i.e., PROUD-MAL followed by FANN architecture. Section “[Experiments and results](#)” narrates the implementation details including the experimental setup, results obtained and discussion. Section “[Conclusion and future direction](#)” narrates the concluding remarks followed by the future direction.

Background and context

Malware can be an executable or a non-executable binary and its classification is based on either dynamic or static analysis. The former approach involves the execution of a PE in a controlled environment to study its behavior including auto-start extensibility points, function calls and parameter analysis, data flow tracking [11] but it is more time consuming and computationally expensive, therefore, the adoption of dynamic analysis in production environment is not appreciated. The static analysis includes source code inspection [12] without any execution in controlled setup that involves decompression/unpacking of PE, if it is encoded by a third-party packer [11] and disassembling for the purpose of obtaining codes residing in memory [14]. The disassembler and memory dumper software packages, e.g., OllyDump and LordPE can be utilized. The windows-based PE file can be an executable, Dynamic Link Library (DLL) [13] or object code and inherits many features from Unix-specific Common Object File format (COFF). The PE content is semantically structured [21] that is important to understand for good analysis. The format is supported by various architectures including Intel, variants of ARM as well as AMD instruction sets. The PE has numerous predefined blocks including a number of headers and sections. The section contains a header that provides information regarding the address and size. The predefined blocks are explained as follows:

- (a) *DOS Header* Defines file as an executable binary or file and also called as MZ header. It provides information about four-byte offset address of PE header.
- (b) *DOS Stub* Small embedded program to display an appropriate message whenever there is an attempt to run a PE file in DOS.
- (c) *PE File Header (Signature)* Defines an executable file as PE. It provides information about machine compatibility, number of sections/symbols, compiler time-stamp and size of optional header, i.e., next unit.
- (d) *Optional Header or Image Optional Header* Mandatory contrary to its title and provides details including entry point address, os version, image and data base, image and subsystem version, the version of linker and size of the code, initialized and uninitialized data, stack and heap
- (e) *Data Directories* Successor of Optional Header. It gives details about directories including export, import, exception, relocation table, global pointer, debug and load configuration.
- (f) *Section Table & Header* Preceded by data directories and provides PE file attributes, instruction to load PE in memory, virtual address, section name, characteristics and size of raw data, etc.

Table 1 PE sections

Name	Description
.text	Executable code
.bss	Uninitialized and static variables
.rdata	Read only data, i.e., constants, string literals
.data	Local and global variables except the automatic variables
.rsrc	Read only information, i.e., resource name, type of icon, bitmap, dialog box
.edata	Export data for an exe or DLL
.idata	Functions imported by an exe or DLL, import directory, import address table
.pdata	Array of function table entries used for exception handling
.debug	Compiler-generated debug information

- (g) *Sections* Contains executable code, resources and operands for PE unlike headers that provide information about executable. There are nine predefined sections. The names and description of each section are listed in Table 1. All sections may not be present in a PE. The missing idata does not mean there is no import table as it may be in .data or .edata section.

Related work

In literature, several approaches for malware detection based on machine learning techniques have been proposed. Some of the research work based on machine learning algorithms specific to PE file malware classification is discussed here. The Malware is a set of instructions developed to bring harmful consequences to organizations, their process, networks as well as infrastructure. The Malware can be an executable or a non-executable entity and its detection is based on either static or dynamic analysis. In 2001, a machine learning framework was proposed for classification of PE files using static analysis and the utilization of data mining techniques for the extraction of strings and byte sequence feature from PE [15]. In 2009, Researchers [16] extracted 5-g byte sequences from file header and applied term frequency-inverse document frequency approach for classification. In 2013, a malware detection system was proposed for analysis of PE files using byte sequence alternatively known as n-gram sequence that is less efficient and computationally expensive [17]. In 2015, researchers [18] proposed heuristic-based detection technique for metamorphic malware while using used static features for PE analysis. In the proposed model, file was disassembled using IDA pro to extract the features. Multiple classification algorithms (j48, j48graft, LADTree, NBTree, Random Forest, REPTree) were used for analysis and classification of PE files. It was highlighted by the

researchers that the classification accuracy is based on the model applied as well as disassembler chosen. In 2018, it has also been shown that machine learning model can learn from sequence of raw bits without explicit feature extraction based on conventional practices of malware classification [19]. The use of machine learning-based classifiers for malware intrusion detection is a well-known approach for network analysis [25]. In addition to string extraction, researchers [30] have also used statistical approach such as raw byte and byte entropy histogram. In [20], researchers presented an approach using static analysis of the features from the PE-Optional Header fields by employing Phi (ϕ) coefficient and Chi-square (KHI2) score. In [23], features were extracted from system calls and submitted to neural network for classification using 170 samples and obtained 0.96 for Area under Curve (AUC). In [24], experiments were performed to identify the critical point to quarantine the activity of malicious code related to its communication with remote command and control server. Researchers [26] presented a framework that ensures the protection of application programs against malware for mobile platform. In 2017, researchers used static analysis to extract key information, i.e., headers strings and sequence from the metadata of PE files. The model was trained over a dataset of 4783 samples using Random Forest and achieved 96% accuracy. The researchers [42], designed a malware classification method for several malware variants based on signature prediction. The proposed solution was based on the static analysis of features including strings, n-grams and hashes extracted from PE header. In [27], the researchers proposed a malware detection system based on supervised learning. They devised tool for feature extraction from header of PE files. Later, system was trained using supervised machine learning classification algorithms such as Support Vector Machine and Decision Trees. In [47], authors proposed Virtual Machine Introspection a machine learning-based approach for malware detection in virtualized environment. The researcher extracted opcodes using static analysis and trained the classifier with selected features. Later, Term Frequency-Inverse Document Frequency (TF-IDF) and Information Gain (IG) were also applied as classification algorithms. In 2019, researcher [29] proposed a malware detection approach in the IoT environment based on similarity hashing algorithm-based. In proposed technique, scores of binaries were calculated to identify the similarity between malicious PEs. Numerous hashing techniques [21] including PEHash, Imphash and Ssdeep were used. Later, researchers integrated hash results using fuzzy logic. Recently, attention models have shown promising output in tasks such as image analysis, machine translation, computer vision and natural language processing [32]. The attention mechanism supports the model to focus on the most relevant features as required. Therefore, we employed the attention-

based mechanism over static features using unsupervised clustering for malware classification.

Methodology and architecture

In this section, design of our proposed unsupervised framework, i.e., PROUD-MAL for classifying windows-based PE using clustering based on static analysis will be explained. The PROUD-MAL is a custom-built unsupervised framework composed of multiple modules including novel dataset collection, dataset pre-processing and feature extraction and unsupervised clustering of the malicious & benign PE samples as illustrated in Figs. 1 and 2. Moreover, the designed Feature Attention-based Neural Network (FANN) is trained over pseudo labels. The proposed classifier is evaluated over the test dataset which was kept hidden during the testing as depicted in Fig. 3.

Malware dataset acquisition

The first stage of the proposed framework is the indigenous dataset collection. In this research work, a pilot attempt is made to perform the dataset collection including the malware and benign samples which will be extended as future research work. A major obstacle in leveraging machine learning techniques for malware analysis is the lack of sufficiently big, labelled datasets that shall contain the malicious as well as benign samples. Moreover, it is very important to keep updating the malware dataset due to ever changing smart evasion approaches adopted by malware authors. The collection of malicious samples was difficult but the collection of benign samples was also not easy. To this end, we used two (02) different approaches for collecting the malicious and benign samples as illustrated in Fig. 1. First, we deployed low and high interaction honeypots as production unit and intentionally configured them in a vulnerable way to collect malicious files and log unauthorized behavior. The low interaction honeypots, i.e., Honeyd [34] as well as high interaction honey pot, i.e., SMB Honey Pot [4] were deployed over the enterprise organizational network to emulate the services frequently targeted by the attacker and the production systems, respectively. Second, Kippo-Malware collector and Kaspersky endpoint security solution is also deployed over the enterprise organizational network to collect malware as well as benign samples. The benign PE including.exe or.dll is also collected from machines with licensed and updated version of Windows operating system including Windows XP, 7, 8 and 10. Special precautions have been taken into account for compliance of licensing and regulatory requirements while collecting benign samples. Moreover, additional precautionary measures such as establishment and configuration of sandbox environment for dataset collection and

Fig. 1 Malware dataset collection and pre-processing

Fig. 2 Unsupervised framework for windows-based PE malware classification

further processing were also taken into consideration. We collected 19,000 samples (31 GB) over the period of seven months (200 days) but after performing dataset verification, samples were reduced to 15,457 (25 GB) PE samples comprising 8775 (17 GB) malicious and 6681 (8 GB) samples. The reduction in number of samples resulted due to filtering

of corrupt and duplicate samples. The verification of samples and removal of duplicates were done using the hash values. The dataset is divided into 60:20:20 ratios for training, validation and testing, respectively, of the proposed model.

Fig. 3 PROUD-MAL validation

Data pre-processing and feature extraction

To prepare the dataset, a series of pre-processing steps were performed, i.e., identification of file type, removal of corrupt and duplicate samples, unpacking of the binaries and verification of labels to transform the raw data into a meaningful format. It was ensured that the dataset shall not contain any duplicate binaries using MD5. It was also ensured that only unpacked binaries shall be submitted for feature extraction therefore section names were examined using a tool PESTudio [45] to see if any of them contains popular packers [46] such as UPX, ASPack, FSG.

Moreover, verification of labels is a significant activity, which was performed by deploying signatures-based anti-virus solutions in parallel and finally using cloud-based service of Virus Total. We used VirusTotal API [44] as well as VirusTotal web interface to submit the binaries for verification. The VirusTotal API does not require to web interface for file submission. It is pertinent to mention here that labelling of samples in the dataset like text, images or speech is relatively an easy task, but the labelling as well as the verification of labels that whether a sample is benign or malicious was very time intensive task. Handling the malicious files needs extra precautionary measures such as establishment and configuration of sandbox environment. During the process, findings were observed such as existence of overlapping segments, usage of non-standard version details, names for sections and zero size of raw data that also results into high virtual size of section in case of packed PE files. It was also observed that some packers make an attempt to reduce entropy by embedding zero bytes in data to bypass screening. Moreover, in malicious files, the data section is missing or has relatively lower value (if present) and permissions assigned to the section are found to be inconsistent in comparison with standard practices. It was also observed that resource size is relatively small as malicious files are mostly non-GUI. The study of compilation time revealed that malwares are mostly compiled during off working days and also do not have genuine creation time. After the dataset preparation and pre-processing, feature extraction was performed. Features were extracted by parsing headers of Portable Executables (PEs). A custom parser was developed to read PE headers, tokenization of features and their respective values. Finally, tokens are orga-

nized in a csv compatible format. More than 35 features were extracted and below is brief description of selected features.

- MD5 is a cryptographic signature. It is a 32-bit hexadecimal value and each file has its unique MD5 value.
- Machine represents the target machine such as Intel 386, MIPS little endian Motorola, etc.
- Size of optional header is a mandatory feature irrespective of the name and provides information related to PE. It is included only for executable files and not for object files.
- Characteristics represent attributes of the file such as base relocation address, local symbol, user program or system file, little-endian or big-endian architecture or whether file is DLL or not etc.
- Major/Minor Linker Version tells the linker to place a version number in the header of the.dll or.exe file.
- Code Size represents size of code (text) section.
- Size of Uninitialized Data is the size of data section.
- Address of Entry Point is the address where the PE loader will begin execution; this address is relative to image base of the executable. It is the address of the initialization function for device drivers and is optional for DLL.
- Base of Code is the pointer to beginning of the code section, which is relative to image base.
- Image Base is the preferred address of the first byte of the executable when it is loaded in memory.
- Section Alignment: The address assignment to PE requires section loading. The section alignment is set to 0×2000 . This means that the code section starts at 0×2000 and the section after that starts at 0×4000 .
- File Alignment: Just like the section alignment, the data also needs to be loaded. It is set to 512 bytes or 0×200 .
- Major/Minor Operating System Version is the version supported by PE.
- Major Image Version is the major version number of image.
- Size of Image is the size of executable after being loaded into memory. It must be multiple of section alignment.
- Size of headers represents the size of all headers, i.e., PE header, the optional header, DOS header.
- Checksum is used for file validation at load time and to confirm whether a file is undamaged or has been corrupted.

- **Sub System** This field points to user interface type required by operating system.
- **Size of Stack Reserve** is number of bytes allocated for stack and determines the stack region utilized by threads.
- **Size of Stack Commit** is the amount of memory that stack is relegated at startup.
- **Size of Heap** represents the space to reserve for loading.
- **Loader Flags** informs upon loading whether to break upon loading, debug on loading or to set to default.
- **Number of RVA** is the number of relative virtual addresses in rest of the optional header. Each entry describes a location and size. The structures contain critical information about specific regions of the PE file.
- **Load Configuration size** is usually used for exceptions. It is only utilized in Windows NT, 2000 and XP.
- **Section Minimum/Maximum/Mean Entropy value** of specific file is represented using digital values and is used to check whether a file is packed or not. Higher entropy usually means that file is malicious.

PROUD-MAL

The PROUD-MAL framework is a progressive unsupervised framework for malware classification based on static analysis of executables. To this end, an architecture with two-phase cascaded formulation of unsupervised clustering with an attention-based deep neural network is proposed. As 80% of dataset was unlabeled, therefore k-means clustering was employed for prediction of pseudo labels. Subsequently, deep neural network was trained using pseudo labels by applying attention over input features. The trained model was then tested over test dataset against the standard performance metric.

Unsupervised clustering

Clustering is a generally ubiquitous and widely accepted instrument of classification for the categorization of data with diversity of application domains including medical imaging, natural language processing, biotechnology and cyber security, etc. It is used in the manner of data exploration where objective function is to learn from data that is not well defined or understood [41]. Several algorithms are available but in this research work, unsupervised clustering is performed by applying k-means algorithm with the motivation for finding a representative, stable clustering solution, which can be further utilized for classification as per the framework architecture. The cluster prediction using unsupervised formulation, i.e., k-mean clustering is depicted in Fig. 4b. Keeping in view the nature of the subject problem, specifically, if the number of classes is known in advance, it is intuitive to initialize the value of k equal to them. However, we still validated the value of k using the elbow method

given an empirical validation regarding the appropriate selection of numbers of clusters in a dataset that is also depicted by Fig. 4b. Nevertheless, if such information is not known in advance, the applicability of other clustering algorithms, e.g., mean-shift [20] or unsupervised deep embedding [21], etc. can be considered more appropriate. Therefore, the extracted features $F = \{f_1, f_2, \dots, f_N\}$ are submitted to k-means algorithm that clusters the similar features (i.e., the corresponding binaries). Using k-means allows us to obtain a set $C = \{c_1, c_2, \dots, c_k\}$ of $k (= M)$ cluster centroids by keeping the following optimization function at minimum:

$$C \leftarrow \arg_c \min \sum_{i=1}^N \sum_{j=1}^M \|f_i - c_j\|^2, \quad (1)$$

where $c_j = 1$, M represents the M cluster centroids. The k-mean clustering iteratively optimizes an Euclidean objective clustering with a self-training distribution to achieve predicted clusters. This progressive clustering is important to refine the obtained pseudo label to optimize the model classification accuracy and subsequently convergence. It will also help to reduce the incorrect assignment which may be more vulnerable to get stuck in bad local optimum. Moreover, the visualization of clustering performance using silhouette analysis and elbow method is also illustrated in Fig. 4a and b, respectively.

Classification using feature attention-based neural network

To end of malware classification in PROUD-MAL framework, we designed a Feature Attention-based Neural Network (FANN) to learn the patterns within a dataset. The FANN is designed to learn feature representation without ground-truth cluster membership labels and is trained over pseudo labels. The pseudo labels are achieved using k-means clustering which iteratively optimizes a Euclidean objective function with a self-training distribution. The FANN comprises of an input layer, output layer, Attention Block (AB) and three hidden layers; illustrated in Fig. 5. All layers are densely connected. A feature vector is input to FANN and is fed forward through the densely connected layers. The first hidden dense layer contains 38 neurons as equal to number of static feature while using rectified linear unit (ReLU) as an activation function. The output of first hidden layer is propagated to the embedded AB. The proposed Attention Block (AB) encodes contextual information by probing feature weight and results in more refined representation by focusing on features of interest. The AB consists of two parallel attention networks /layers. Each network computes the attention for features subsequence of a PE instance and also incorporates prior knowledge to predict new weights. The attention mechanism is discussed in more detail in follow-

Fig. 4 **a** k-mean clusters of PE binaries. **b** Elbow score for cluster prediction

Fig. 5 Feature attention-based neural network (FANN)'s architecture

ing subsection. The third and fourth layers comprise of 13 neurons each followed by output layer using sigmoid activation for binary classification. The model is further fine-tuned by adjusting the hyperparameters to achieve the optimum results.

Attention mechanism

As we introduced above, PE header has numerous features where some features might have a higher impact on identifying malicious PEs. Therefore, we employ attention mechanism to prioritize significance of important features while penalizing the “noise” fields. The main principle behind proposed Attention Block (AB) is as follows: The selection of significant feature rather than examining entire feature set improves classification. To this end, a feature vector sequence of length n is extracted from PE header. After processing feature vector at first iteration, significant combination of length k is selected based upon attention threshold. Subsequently, this subsequence is uti-

lized as prior knowledge to train the model to predict classification of PEs. The sequence can be represented as $\{F_1, F_2, \dots, F_n\}$. The weighted vector containing W_i of each data point S_i in feature combination sequence is represented as $\{(F_1, W_1), (F_2, W_2), \dots, (F_n, W_n)\}$. Next, we extract subsequence with k highest weights: $\{(F'_1, W'_1), (F'_2, W'_2), \dots, (F'_k, W'_k)\}$. As discussed earlier, the AB connects two parallel attention network/layers of opposite directions to same output. Each network/layers computes the $\text{attn}_{(i,h)}$ for features of a PE instance given as input, where i represents features and h represents number of units. One network processes sequence from top to bottom (forwards) and other processes the sequence from bottom to top (backwards). Let x_t denote current step of input sequence, h_{t-1} denote previous hidden state. The next hidden state h_t can be calculated as follows:

$$h_t = f(Ax_t + Wh_{t-1}), \quad (2)$$

where f is a non-linear activation function. A and W represent weight matrices of current input vector x_t and previous

hidden state h_{t-1} . At each time step t , the forward pass calculates hidden state h_t by considering previous hidden state h_{t-1} and new input sequence x_t . At the same time, backward flow computes hidden state h_t considering future hidden state h_{t+1} and the current input x_t . Afterward, the best output among both forward h_t and backward h_t are selected to obtain refined vector representation. The first network or set of layers in AB used sigmoid function while the other used ReLU function. Finally, the best output is applied to feature importance map while taking the product of learned parameters with respective probabilities. As each layer computes the $\text{attn}_{(i,h)}$ for features of a PE instance given as input, where i represents features and h represents number of units. The feature weights for first layer can be learned as Eq. (3).

$$a_{(i,h)} = \sigma(x_{(i,h)}, W), \quad (3)$$

where x_i is the input to layer and W denotes weights of layer and σ represent sigmoid activation function to feature map $w(i, h)$ for the first attention layer in Eq. (4).

$$b_{(i,h)} = \partial(x_{(i,h)}, W). \quad (4)$$

Similarly, ∂ represent ReLU activation function employed by second attention layer to feature map $w(i, h)$ followed by selecting maximum of Eqs. (3) and (4).

$$w_{(i,h)} = \max(a_{(i,h)}, b_{(i,h)}) \quad (5)$$

The attention is computed by multiplication of $w(i, h)$ with output of sigmoid function as:

$$\text{attn}_{(i,h)} = \left[\frac{\exp(w_{(i,h)})}{\exp(w_{(i,h)}) + 1} \right] \otimes w_{(i,h)}. \quad (6)$$

The feature attention-based layer learns to put relatively more to those features that have contributed more to minimize the validation loss while learning the accurate classification by applying sigmoid function to the feature importance map and subsequently multiplying learned parameters with the respective probabilities. The dataset based on validated predicted clusters is splitted into 60:20:20 ratios for classification training, validation, and testing, respectively. The model is trained over the predicted cluster dataset using classification algorithms including Random Forest, Support Vector Machine (SVM), Gradient Boost, Ada Boost, Naive Bayes and PROUD-MAL. The training is performed for 60 epochs (i.e., approx. 23,185 iterations) and the input was submitted to the network in a batch of 32 with Adam as an optimizer and learning rate was initialized with stepwise decay at 0.001 or 10^{-3} . Binary cross entropy is utilized for loss calculation over the training data. After model training, it is tested to make predictions against validation dataset.

Table 2 Hyper-parameters and associated values

Parameters	Values
Batch size	32
Epochs	60
Dropout	0.5
Learning rate	0.001
Loss function	Binary cross entropy
Optimizer	ADAM
Momentum	0.9

Finally, the empirical validation of proposed PROUD-MAL approach is also performed against standard metrics over test dataset which is kept hidden during the training phase.

Experiments and results

Implementation details

This section narrates the configuration and performance metric used for the experiment to classify Windows-based PE. The run time environment configured for experiments includes a workstation with Intel Core i5-9500 Processor @ 3.0 GHz with 6 cores and 6 logical processors, 32 GB Ram, virtual memory of 20.0 GB with enabled virtualization, graphic card NVIDIA GeForce GTX 1650 with 4 GB Ram and Window 10 Pro 64 operating system. In terms of software, both Keras and Tensorflow were employed at back-end for implementation of our proposed framework. The training is performed for 60 epochs (i.e., approx. 23,185 iterations) and input was submitted to network in a batch of 32 with Adam as an optimizer and learning rate was initialized with stepwise decay at 0.001 or 10^{-3} . Dropout regularization of 0.5 is placed in after fully connected layers which help to prevent overfitting. Generally, dropout removes neurons and its connections randomly. Moreover, we adopted binary cross-entropy loss function, which is minimizing the negative logarithmic likelihood between the prediction and the ground-truth data. The momentum helps accelerating the ADAM in the relevant direction and mitigates oscillations by adding a fraction of the update vector of the past time step to the current update vector. The accuracy and loss parameters provided by Keras are visualized in better manner utilizing tensor board and console logs. The summary of hyper parameters is provided in Table 2.

Results and discussion

We performed comparison of our proposed method with state-of-the-art supervised approaches. Despite this challenging comparison, the utility of our proposed framework is

Table 3 Quantitative assessment of PROUD-MAL with supervised approaches—results

Model	Accuracy	F1 Score	Precision	AUC	TP	FP	FN	TN
Random Forest	94.27	95.06	98.71	98.78	0.91	0.09	0.03	0.97
SVM	93.01	94.07	97.61	97.40	0.87	0.13	0.02	0.98
Gradient Boost	56.71	72.38	88.28	90.99	0.0	1.0	0.0	1.0
Ada Boost	91.91	93.21	88.28	90.99	0.84	0.16	0.02	0.98
Naive Bayes	56.68	39.05	94.75	95.37	0.99	0.01	0.76	0.24
PROUD-MAL	98.09	98.33	97.30	99.55	0.98	0.02	0.02	0.98

Table 4 Quantitative assessment of PROUD-MAL with unsupervised approach—results

Model	Accuracy	F1 Score	Precision	AUC	TP	FP	FN	TN
Hyrum et al. [48]	92.89	93.71	96.31	93.35	0.86	0.14	0.03	0.97
PROUD-MAL	98.09	98.33	97.30	99.55	0.98	0.02	0.02	0.98

well demonstrated by achieving high classification accuracy. To perform model assessment in a quantitate fashion, we used standard metrics of classification accuracy, F1 score, precision, Receiver Operating Characteristic (ROC) curve, area under the ROC curve (AUC), True Positive (TP), False Positive (FP), True Negative (TN) and False Negative (FN) Rate. The accuracy and other parameter results for PROUD-MAL are illustrated in Tables 3, 4. In our experiments, we considered results of Random Forest (RF) as a baseline classification results for comparative study, due to its high classification accuracy. However, other classification algorithms are also employed for detailed comparison including Support Vector Machine (SVM), Gradient Boost (GB), Ada Boost (AB) and Naive Bayes (NB). The experiments include testing of classifier over novel dataset which is kept hidden during training phase against the standard evaluation metrics. Tables 3, 4 show quantitative results of comparative analysis of PROUD-MAL and other classification algorithms including Random Forest, Support Vector Machine, Gradient Boost, Ada Boost and Naive Bayes algorithms over the collected dataset. It can be seen in Tables 3, 4 that the best performance is achieved by PROUD-MAL with a classification accuracy of 98.09%. The RF, SVM and AB also showed good performance by achieving classification accuracy of 94.27%, 93.01% and 91.91%. However, GB and NB achieved lowest classification accuracy, i.e., 56.71 and 56.68%, respectively.

A detailed analysis of confusion matrix shows that the proposed PROUD-MAL framework with Feature Attention-based Neural Network (FANN) demonstrated best classification accuracy of 98.09% against standard evaluation metrics on our indigenously collected novel dataset. However, RF, SVM and AB also showed good performance by achieving classification accuracy of 94.27%, 93.01% and 91.91%. The GB and NB achieved the lowest classification accuracy, i.e., 56.71 and 56.68%, respectively. It is worth mentioning that PROUD-MAL achieved 4%, 5.46%, 72%, 6.72% and 73% higher accuracy than classical machine learning clas-

Fig. 6 Validation and training accuracy as well as loss of PROUD-MAL **Fig. 7** ROC curve of PROUD-MAL

sification models including random forest, SVM, Gradient Boost, Ada Boost and Naive Bayes, respectively. Moreover, our experiments show that FANN demonstrated overall higher AUC of 99.55% as compared to other classifiers which

Fig. 8 t-SNE clusters visualization of PEs

shows better predictive power and can also provide better sensitivity tuning. To the best of our knowledge, this is due to unsupervised clustering cascaded by classifier with embedded attention layers. However, RF, SVM and NB also showed good performance by achieving AUC of 98.78%, 97.40% and 95.37%. The GB and AB achieved relatively lower AUC, i.e., 90.99%. The comparison with unsupervised approach [Hyrum S. Anderson et.al. 2018] also showed superior performance. Our approach demonstrated 5.2% high classification accuracy. The detailed comparative assessment with supervised approaches as well as unsupervised one has shown utility and significance of the proposed architecture. It is also pertinent to mention that for classification of an unknown PE using an anti-virus software, the training time is not important because we can use pre-trained neural network. As test time of FANN model is less than 21 ms per step, the model is appropriate for its subsequent utility in real anti-virus software.

Experiments show the True Positive (TP), False Positive (FP), True Negative (TN), False Negative (FN) rate for FANN is 0.98, 0.02, 0.98 and 0.02, respectively. The quantitative assessment was conducted over 60 epochs with a batch size of 32. The training-validation accuracy as well as training-validation loss is depicted in Fig. 6. The training and validation graphs in Fig. 6 depict that PROUD-MAL is trained quite well enough around 60 epochs. We also employed early stopping criteria to discontinue further training at an appropriate stage. It is worth mentioning that as we increase the number of iterations, the loss or learning rate descends gradually (not showing due to non-significance in figure for later iterations.) Moreover, the graphs for training and validation are also illustrated in Fig. 6. As our dataset has 15,457 binaries comprising 8775 (17 GB) malicious and 6681 (8 GB) samples, therefore, we also calculated the area under the ROC curve (AUC) as illustrated in Tables 3, 4, which is a widely used performance metric for imbalanced datasets. A visual inspection of the Receiver Operating Char-

acteristic (ROC) curve (Fig. 7) shows that our framework shows superior performance compared to other state-of-the-art supervised approaches. PROUD-MAL achieved ROC of 0.99 with small discrepancy of 0.01. The visualization of cluster prediction is generated by applying t-SNE on the dataset and is depicted in Fig. 8. The blue dots represent the malicious binaries and yellow mark represents the benign PEs. The visual exhibit reflects minor overlapping between the malicious and benign samples. There were 38 features in vector space. However, by applying attention mechanism, it is revealed that features with the numerical values, e.g., section entropy, size of sections, image base were given more weight by AB. On the other hand, the features that either represent unique numerical value or fixed length value with a specific format, e.g., MD5, checksum are given relatively less weight than the normal numerical values such as section entropy. But these attributes are given more consideration in comparison with the features having string values, e.g., machine, characteristics, compiler etc. The proposed scheme of using feature subsequence combination by applying attention mechanism resulted in more refined feature representation. Subsequently, quantitative results of comparative assessment have demonstrated the utility of attention mechanism for unsupervised classification of PEs using static features.

Conclusion and future direction

We have proposed and presented a progressive deep unsupervised malware classification framework, i.e., PROUD-MAL with a deep neural network architecture that uses dense layers and an attention block for binary classification of Windows-based PEs based on features extracted from header in a static fashion. Our proposed feature attention mechanism-based neural network for malware classification learns to put relatively more weights to those features that contributed more to

minimize the validation loss while learning the accurate classification. We also collected novel real-time malware dataset by deploying low and high interaction honeypots as well as endpoint security solution on an enterprise organizational computer network for validation of proposed framework. This indigenously collected dataset is novel and is made public for the research community. We also look forward to enhance existing volume of novel dataset. The quantitative assessment reflects that the proposed PROUD-MAL framework achieved an accuracy of more than 98.09% with better quantitative performance in standard evaluation metrics on indigenously collected novel dataset and outperformed other conventional machine learning algorithms. As a way forward, our framework can be enhanced to explore the behavioral analysis based on API calls [49] using reinforcement learning [50] for malware analysis. This includes the transformation of PEs into malware images and performs entropy based semantic segmentation of malware images. This will potentially help malware authors to use malware visualization to perform malware analysis more effectively for zero-day malware samples. The scope of future direction may also include Non-Portable Executable (NPE) files.

Acknowledgements The authors would like to express their gratitude for research grant of Higher Education Commission (HEC) of Pakistan under International Research Support Initiative Program (IRSIP) and the institutional support from the Department of Computer Science, University of Warwick, Coventry, United Kingdom (UK).

Declarations

Conflict of interest The authors declared that they have no conflict of interest.

Open Access This article is licensed under a Creative Commons Attribution 4.0 International License, which permits use, sharing, adaptation, distribution and reproduction in any medium or format, as long as you give appropriate credit to the original author(s) and the source, provide a link to the Creative Commons licence, and indicate if changes were made. The images or other third party material in this article are included in the article's Creative Commons licence, unless indicated otherwise in a credit line to the material. If material is not included in the article's Creative Commons licence and your intended use is not permitted by statutory regulation or exceeds the permitted use, you will need to obtain permission directly from the copyright holder. To view a copy of this licence, visit <http://creativecommons.org/licenses/by/4.0/>.

References

1. Tang Y, Xiao B, Lu X (2011) Signature tree generation for polymorphic worms. *IEEE Trans Comput* 60:565–579. <https://doi.org/10.1109/TC.2010.130>
2. Internet security threat report (2019) <https://www.symantec.com/content/dam/symantec/docs/reports/istr-24-2019-en.pdf>
3. Gandotra E, Bansal D, Sofat S (2014) Malware analysis and classification: a survey. *J Inf Secur* 5:56–64
4. Provos N (2004) A virtual honeypot framework. In: *Proceedings of the 13th conference on USENIX Security Symposium*
5. Sung AH, Xu J, Chavez P, Mukkamala S (2004) Static analyzer of vicious executables (SAVE). In: *Proceedings of the 20th annual computer security applications conference*
6. Wuchner T, Cislak A, Ochoa M, Pretschner A (2020) Leveraging compression-based graph mining for behavior-based malware detection. *IEEE Trans Depend Secure Comput* 16:1
7. Ghafir I, Hammoudeh M, Prenosil V (2018) Defending against the advanced persistent threat: Detection of disguised executable files. *PeerJ Preprints* 6:e2998. <https://doi.org/10.7287/peerj.preprints.2998v2>
8. Alazab M, Layton R, Venkataraman S, Watters P (2010) Malware detection based on structural and behavioral features of API calls, pp 1–10. In: *International cyber resilience conference 2010—Perth*
9. Devesa J, Santos I, Cantero X, Peña YK, Bringas PG (2010) Automatic behavior-based analysis and classification system for malware detection. In *ICEIS* 2:395–399
10. Slam R, Tian R, Batten LM, Versteeg S (2013) Classification of malware based on integrated static and dynamic features. *J Netw Comput Appl* 36(2):646–656
11. Egele M, Scholte T, Kirda E, Kruegel C (2012) A survey on automated dynamic malware-analysis techniques and tools. *ACM Comput Surv* 44:2
12. Christodorescu M, Jha S (2003) Static analysis of executables to detect malicious patterns. In: *Proceedings of the 12th conference on USENIX security symposium*, vol 12, p 12
13. Ye Y, Wang D, Li T, Ye D (2007) IMDS: intelligent malware detection system. In: *Proceedings of ACM international conference on knowledge discovery and data mining (SIGKDD)*, pp 1043–1047
14. Gandotra E, Bansal D, Sofat S (2014) Malware analysis and classification: a survey. *J Inf Secur* 2:56–64
15. Schultz MG (2001) Data mining methods for detection of new malicious executables. In: *Proceedings of the IEEE symp. on security and privacy*, pp 38–49
16. Shabtai A, Moskovitch R, Elovici Y, Glezer C (2009) Detection of malicious code by applying machine learning classifiers on static features: a state-of-the-art survey. *Inf Secur Tech Rep* 14:16–29
17. Eskandari M, Khorshidpour Z, Hashemi S (2013) Hdm-analyser: a hybrid analysis approach based on data mining techniques for malware detection. *J Comput Virol Hack Tech* 9(2):77–93
18. Khodamoradi P, Fazlali M, Mardukhi F, Nosrati M (2015) Heuristic metamorphic malware detection based on statistics of assembly instructions using classification algorithms. In: *18th CSI international symposium on computer architecture and digital systems (CADSD)*, IEEE, pp 1–6
19. Raff E, Sylvester J, Nicholas C (2017) Learning the PE header, malware detection with minimal domain knowledge. In: *Proc. 10th ACM workshop artificial intelligence secure*, New York, ACM, pp 121–132
20. Belaoued M (2015) A real-time PE-malware detection system based on CHI-Square test and pe-file features. In: *IFIP international conference on computer science and its applications CIIA 2015: computer science and its applications*, pp 416–425
21. Pietrek M (1994) Peering inside the PE: a tour of the Win32 portable executable file format
22. Rossow C et al (2012) Prudent practices for designing malware experiments: status quo and outlook. In: *Proc. IEEE symp. secur. privacy (SP)*, pp 65–79
23. Tobiyama S, Yamaguchi Y, Shimada H, Ikuse T, Yagi T (2016) Malware detection with deep neural network using process behavior. In: *Proc. IEEE 40th annu. comput. softw. appl. conf. (COMPSAC)*, vol 2, pp 577–582

24. Shibahara T, Yagi T, Akiyama M, Chiba D, Yada T (2016) Efficient dynamic malware analysis based on network behavior using deep learning. In: Proc. IEEE Global Commun. Conf. (GLOBECOM), pp 1–7
25. Mutz D, Valeur F, Vigna G (2006) Anomalous system call detection. *ACM Trans Inf Syst Secur* 9(1):61–93
26. Zhauniarovich Y, Russello G, Conti M, Crispo B, Fernandes E (2014) Moses: supporting and enforcing security profiles on smart-phones. *Depend Secure Comput IEEE Trans* 11(3):211–223
27. Raff E, Nicholas C (2017) An alternative to NCD for large sequences, Lempel-Ziv Jaccard distance. In: Proceedings of the 23rd ACM SIGKDD international conference on knowledge discovery and data mining (ACM, 2017), pp 1007–1015
28. Rastogi V, Qu Z, McClurg J, Cao Y, Chen Y (2015) Uranine: real-time privacy leakage monitoring without system modification for android. *Springer Int. Pub, Cham*, pp 256–276
29. Namanya AP, Awan IU, Disso JP, Younas M (2019) Similarity hash-based scoring of portable executable files for efficient malware detection in IoT. *Fut Gen Comput Syst* 110:824–832. <https://doi.org/10.1016/j.future.2019.04.044>
30. Merkel R (2010) Statistical detection of malicious PE executables for fast offline analysis. In: Springer, Berlin, Heidelberg, ISBN 978-3-642-13241-4, pp 93–105. https://doi.org/10.1007/978-3-642-132414_10
31. Catak FO, Yazı AF, Elezaj O, Ahmed J (2020) Deep learning based Sequential model for malware analysis using Windows exe API Calls. *PeerJ Comput Sci* 6:e285. <https://doi.org/10.7717/peerj-cs.285>
32. Rush AM, Harvard SEAS, Chopra S, Weston J (2015) a neural attention model for sentence summarization. In: Proceedings of the international conference on empirical methods in natural language processing, Lisbon, Portugal
33. Collberg C, Thomborson C (2002) Watermarking, tamperproofing, and obfuscation - tools for software protection. *IEEE Trans Software Eng* 28(8):735–746
34. Koniaris I, Papadimitriou G, Nicopolitidis P, Obaidat M (2014) Honeypots deployment for the analysis and visualization of malware activity and malicious connections. In: 2014 IEEE international conference on communications (ICC), Sydney, pp 1819–1824. <https://doi.org/10.1109/ICC.2014.6883587>
35. Zhou Y, Jiang X (2012) Dissecting android malware: characterization and evolution. In: IEEE symposium on security and privacy, pp 95–109
36. Wan M, Shang W, Zeng P (2017) Double behavior characteristics for one-class classification anomaly detection in networked control systems. *IEEE Trans Inf Forens Secur* 12(12):3011–3023
37. Hearst M, Dumais S, Osman E, Platt J, Scholkopf B (1998) Support vector machines. *IEEE Intell Syst Appl* 13(4):18–28. <https://doi.org/10.1109/5254.708428>
38. Seiffert C, Khoshgoftaar TM, Van Hulse J, Napolitano A (2010) RUSBoost: a hybrid approach to alleviating class imbalance. *IEEE Trans Syst Man Cybern A Syst Hum* 40(1):185–197
39. Caruana A et al (2006) An empirical comparison of supervised learning algorithms. In: ICML '06 proceedings of the 23rd international conference on machine learning, pp 161–168
40. Kanungo T, Mount D, Netanyahu N, Piatko C, Silverman R, Wu A (2002) An efficient K-means clustering algorithm analysis and implementation. *IEEE Trans Pattern Anal Mach Intell* 24:881–892. <https://doi.org/10.1109/TPAMI.2002.1017616>
41. Boutsidis C, Mahoney M, Drineas P (2009) Unsupervised feature selection for the k-means clustering problem. In: Advances in neural information processing systems 22-proceedings of the conference, pp 153–161
42. Gandotra E, Bansal D, Sofat S (2016) Zero-day malware detection. In: Sixth international symposium on embedded computing and system design (IEEE, 2016), pp 171–175
43. Ng CK, Jiang F, Zhang LY, Zhou W (2019) Static malware clustering using enhanced deep embedding method. *Concurr Computat Pract Exper* 2019:e5234. <https://doi.org/10.1002/cpe.5234>
44. Algaith A, Gashi I, Sobesto B, Cukier M, Haxhijaha S, Bajrami G (2016) comparing detection capabilities of antivirus products: an empirical study with different versions of products from the same vendors. In: 2016 46th Annual IEEE/IFIP international conference on dependable systems and networks workshop (DSN-W), Toulouse, pp 48–53. <https://doi.org/10.1109/DSN-W.2016.45>
45. Kozachok AV, Kozachok VI (2018) Construction and evaluation of the new heuristic malware detection mechanism based on executable files static analysis. *J Comput Virol Hack Tech* 14:225–231. <https://doi.org/10.1007/s11416-017-0309-3>
46. Hassen M, Carvalho MM, Chan PK (2017) Malware classification using static after validation analysis-based features. In: 2017 IEEE symposium series on computational intelligence (SSCI), Honolulu, pp 1–7. <https://doi.org/10.1109/SSCI.2017.8285426>
47. Vadrevu P, Perdisci R (2016) MAXS: scaling malware execution with sequential multi-hypothesis testing. In: Proceedings of the 11th ACM on Asia conference on computer and communications security (ACM, 2016), pp 771–782
48. Anderson HS, Kharkar A, Filar B, Evans D, Roth P (2018) Learning to evade static PE machine learning malware models via reinforcement learning. <http://arxiv.org/abs/1801.08917>
49. Wang Y, Stokes J, Marinescu M (2020) Actor critic deep reinforcement learning for neural malware control. *Proc AAAI Conf Artif Intell* 34(01):1005–1012
50. Wu C, Shi J, Yang Y, Li W (2018) Enhancing machine learning based malware detection model by reinforcement learning. In: Proceedings of the 8th international conference on communication and network security (ICCNS 2018), pp 74–78. <https://doi.org/10.1145/3290480.3290494>

Publisher's Note Springer Nature remains neutral with regard to jurisdictional claims in published maps and institutional affiliations.