Cybersecurity

RESEARCH

Open Access

DEFENDIFY: defense amplified with transfer learning for obfuscated malware framework

Rodrigo Castillo Camargo[1], Juan Murcia Nieto[1], Nicolás Rojas[4], Daniel Díaz-López[1*], Santiago Alférez[3], Angel Luis Perales Gómez[2], Pantaleone Nespoli[2], Félix Gómez Mármol[2] and Umit Karabiyik[5]

Abstract

The existence of malicious software (malware) represents a potential threat to users who connect to a large set of services provided by multiple providers. Such malware is capable of stealing, spying on, encrypting data from users, and spreading, provoking impacts that are beyond a single citizen's device and reaching critical information systems. To detect malware families, Machine Learning and Deep Learning techniques have been employed recently, demonstrating promising results. However, these techniques lack in detecting more advanced malware that employs obfuscation techniques. In this paper, we present DEFENDIFY, a novel framework, empowered by Computer Vision, Deep Learning, and Transfer Learning techniques, that is able to detect completely obfuscated malware with high performance in terms of accuracy and computational consumption. DEFENDIFY comprises three modules: Dataset Creation, Binary Obfuscation, and Model Generation. These modules work together to detect both obfuscated and non-obfuscated malware. The core module, i.e., the Model Generation, employs an entropy tester that determines whether a sample is obfuscated or not. Then, a Deep Learning model powered by Transfer Learning is employed to determine if it is malware or goodware. We validated our framework using real data gathered from malware repositories and legitimate software. The proposed framework was configured to test four Convolutional Neural Network architectures: ResNet18, ResNet34, EfficientNetB3, and EfficientNetV2S. Among them, the ResNet18 architecture obtained the best performance in detecting both non-obfuscated and obfuscated samples with an F1-score of 99.34% and 97.5%, respectively.

Keywords Malware detection, Malware obfuscation, Computer vision, Transfer learning, Deep learning, Networking system of artificial intelligence

Introduction

With the rise of new technologies such as the Internet of Things (IoT) (Glaroudis et al. 2020) and Big Data (BD) (Iaksch et al. 2021), cities have become increasingly automated, thus embracing the smart cities paradigm (Ristvej et al. 2020). This has undoubtedly provided a clear advantage for all citizens, enabling access to unimaginable services just ten years ago. However, there arises an urgent need for safeguarding the digital backbone. As we immerse ourselves in the transformative potential of IoT and BD within city infrastructures, the vulnerability to cyber threats becomes increasingly pronounced, and therefore, new dangers loom over the inhabitants of a smart city.

*Correspondence:
Daniel Díaz-López
danielo.diaz@urosario.edu.co
[1] School of Engineering, Science and Technology, Universidad del Rosario, Kr 6 12c-16, Bogotá 111711, Cundinamarca, Colombia
[2] Faculty of Computer Science, University of Murcia, Campus de Espinardo, 30100 Murcia, Murcia, Spain
[3] Department of Mathematics, Barcelona East Engineering School, Polytechnic University of Catalonia, 610101 Barcelona, Catalonia, Spain
[4] School of Engineering, Pontifical Xavierian University, Bogotá 610101, Cundinamarca, Colombia
[5] Department of Computer and Information Technology, Purdue University, West Lafayette, IN 610101, USA

One of the critical threads in smart cities is cybersecurity and, especially, the proliferation of malware (Ma 2021). Specifically, this term is used to refer to malicious programs designed to perform harmful actions on user's devices (Useche-Pelaez et al. 2018). These actions may include spying on users, stealing sensitive information, damaging the victim's device, or executing other malicious actions, to cite some examples (Martínez Martínez et al. 2021). Different smart city services could be affected by a malicious campaign, e.g. traffic management, pollution monitoring, water quality control, public safety support, etc. Furthermore, there are also organized campaigns designed to spread malware and target different victims, as dissemination and exploitation of malware have become a profitable business (Hakon et al. 2020).

As new forms of malware are constantly being created and spread, the anti threat industry has sought to mitigate the risk of a machine being infected with a malicious program through various malware detection techniques. These techniques include the use of different types of signatures, such as host-based, malware, and network signatures (Pelaez et al. 2018; Nespoli et al. 2019), or ML algorithms such as support vector machine (SVM) and Random Forest, achieving promising results (Joshi et al. 2018).

Due to the advancement in antivirus industry, malware evasion techniques have also been developed to reduce the effectiveness of the proposed detection methods. Among others, a well-known evasion technique leverages the use of encoders or obfuscators, which are coding functions that change the appearance of programs, thus making them unrecognizable to antiviruses and allowing attackers to bypass protection layers when delivering malware (Kim et al. 2018). Nevertheless, it is important to note that not all encoded or obfuscated binaries are malicious programs. There is also well-intentioned software, commonly referred to as goodware, that is encoded by its developers to protect it from industrial espionage or tampering (Toffalini et al. 2019). In this paper, the words encoded/obfuscated and non-encoded/non-obfuscated will be used indistinctly.

In this scenario, some studies proposed the use of artificial intelligence (AI), and especially machine learning (ML) and deep learning (DL) techniques to detect malware. Among these techniques, DL models focused on the computer vision (CV) field are gaining prominence due to their ability to automatically extract features that can capture subtle patterns in obfuscated code that might be invisible to traditional analysis methods.. The procedure is to consider malware as images and use image classification to detect malware by identifying key features of the image (Pereira-Kohatsu et al. 2019; Xue et al. 2019; Bensaoud et al. 2020). The most critical limitation of these approaches and those based on traditional ML algorithms or signatures is that they do not allow one to detect completely obfuscated malware. Furthermore, the procedures to train these models are highly expensive regarding the computation time since they need to train an entire DL model from scratch, preventing the wide adoption of these systems in current technological scenarios or in future smart city architectures.

One technique that can improve the result obtained in this field is the transfer learning (TL) (Weiss et al. 2016), which consists on apply the knowledge acquired in one domain such as CV on another new domain such as malware classification. By transferring learned representations from pre-trained models on image datasets, it is possible to adapt these insights to the specific task of malware detection. This strategy not only reduces the training time required but also significantly enhances detection performance, making it a more efficient and effective solution for identifying complex, hidden threats.

The major motivation for our research stems from three critical challenges in current malware detection approaches. First, there is a significant gap in effectively detecting completely obfuscated malware using static analysis techniques, which is essential for identifying threats before execution. Second, the substantial computational resources required for training deep learning models from scratch make many current solutions impractical for widespread deployment. Third, there is a lack of comprehensive frameworks that integrate different components necessary for effective detection of both obfuscated and non-obfuscated malware in real-world scenarios.

Additionally, another outstanding component in detecting malware samples is the entropy tester (Bat-Erdene et al. 2017). In particular, the entropy-based obfuscation tester can be described as a fundamental component as part of the static analysis process, allowing for the classification of both obfuscated and non-obfuscated binaries without needing dynamic analysis.

Last but not least, a critical challenge for AI-powered malware detection is the lack of diverse and comprehensive datasets (Allix et al. 2015), particularly when it comes to handling encoded or obfuscated samples. Without access to a variety of datasets, including non-encoded, XOR-encoded, and Shikata Ga Nai-encoded goodware and malware, AI models are unable to fully learn the complexities of both obfuscated and non-obfuscated threats. This gap severely limits the ability of existing solutions to generalize across different encoding techniques used by malware authors to evade detection.

These challenges underscore the urgent need for a novel, integrated approach that can effectively detect obfuscated malware while remaining computationally

efficient. Our research is driven by the vision of creating a practical and deployable framework that addresses these limitations by leveraging the strengths of both computer vision and transfer learning, while incorporating specialized components to handle different types of malware, whether obfuscated or not.

In light of this, this paper proposes DEFENDIFY, a novel framework empowered by CV, DL, and TL (Ribani and Marengoni 2019), to detect obfuscated malware samples. DEFENDIFY may be applied to protect user's devices, or particular smart citizens' devices under a smart city scenario. In this sense, TL refers to the use of knowledge acquired from one task to improve performance on a related but different task without training again the whole DL model. That is, the models are pretrained on various image datasets, and the knowledge acquired is then used as baseline to classify obfuscated and non-obfuscated malware, avoiding a potentially expensive training phase. With this in mind, and in order to make use of the knowledge obtained from the pretrained CV artifacts, the ingested binaries must be transformed into images compatible with the original network architecture. The reason for this is that, although TL is a powerful learning tool, it has some restrictions to use the same type of data format with which the "transferred" knowledge was generated.

Specifically, DEFENDIFY consists of three modules: Dataset Creation, Binary Obfuscation, and Model Generation. These modules work together to detect completely obfuscated malware with a high performance in terms of accuracy and computational consumption. The use of CV techniques and convolutional neural networks (CNN) is particularly well-suited for image classification tasks and has shown promising results in various CV applications. In our proposal, the binary code of obfuscated malware is converted into image, aiming to provide a new perspective for malware detection, which can overcome the limitations of traditional detection methods and improve the detection of obfuscated malware.

To summarize, the specific contributions of this paper are as follows:

1. A novel and generic framework powered by DL and TL technologies that consists of three modules to detect both obfuscated and non-obfuscated malware. To be specific, the framework incorporates an entropy tester to differentiate obfuscated and non-obfuscated samples, and two DL models powered by TL that classify goodware and malware.
2. The construction of three datasets of non-encoded, XOR-encoded and Shikata Ga Nai-encoded goodware and malware samples. These datasets were generated by the framework itself thanks to the Dataset Creation module. In later steps, these datasets were converted into images by the Binary Obfuscation module using a specific codec.
3. The validation of DEFENDIFY in terms of evaluation performance and computational consumption. DEFENDIFY was configured to test four CNN architectures: ResNet18, ResNet34, EfficientNetB3, and EfficientNetV2S. In terms of evaluation performance, the best architecture was RestNet18 that reached an F1-score of 99.34% and 97.24% for non-obfuscated and obfuscated malware detection, respectively. In terms of computational consumption, the most efficient architecture was RestNet18 followed by RestNet34, EfficientNetB3, and finally EfficientNetV2S.
4. A functional full-fledged prototype of DEFENDIFY, offered as an IA-enabled smart service for the detection of advanced malware, which is accessible to the scientific community who desire to test it and validate its practicality and easiness-to-use. The prototype is accessible at: https://huggingface.co/spaces/esab/malware_detection

The remainder of the manuscript is structured as follows. "Leveraging computer vision for cybersecurity" section contains a overview of how CV may support cybersecurity challenges. "State of the art" section compares the main works proposed in this field, highlighting its contributions and techniques. Then, "DEFENDIFY framework" section describes the details of the DEFENDIFY framework. In "Experiments", the experiments that were carried out are shown in detail. "Discussion" section introduce a brief discussion about the work presented in this paper. At last, "Conclusions and future work" section concludes the work, summarizing the results and showing the future directions of this research.

Leveraging computer vision for cybersecurity

Computer vision (CV) has emerged as a transformative technology in the field of cybersecurity due to its ability to analyze and interpret complex data patterns. By converting data into visual representations, CV techniques offer innovative approaches for identifying and mitigating cyber threats, particularly in scenarios involving malware detection and analysis.

One of the key advantages of CV is its applicability to static malware analysis. By transforming binary files into grayscale images, convolutional neural networks (CNNs) can be employed to detect patterns indicative of malicious code. These models excel at recognizing features associated with obfuscation techniques or specific malware families, enabling high-accuracy classification even in challenging cases (Li et al. 2022). For example, CV-powered classifiers can analyze code structures

or entropy patterns to identify hidden malicious binaries, supporting static code analysis of logs and network frames collected from information systems or network devices.

CV can also be utilized for dynamic threat detection. In network security, CNNs can analyze traffic patterns to identify anomalies, such as unusual port activity or data exfiltration attempts. These methods enhance the ability to detect both known and zero-day attacks, contributing to the protection of critical systems (Li et al. 2022).

Additionally, CV-based solutions can support phishing detection by analyzing visual and structural features of email campaigns. CNNs trained on phishing patterns can filter suspicious content, providing a proactive layer of defense against social engineering attacks. This is particularly relevant for organizations exposed to high volumes of communication data, where manual analysis would be infeasible. Examples of similar approaches include applications by Mimecast CyberGraph and Cofense Cyberfish (Mimecast 2025; Cyberfish 2025).

Another application of CV in cybersecurity is its role in countering adversarial attacks. Attackers often attempt to bypass machine learning models by introducing subtle perturbations in input data. CV techniques can enhance model robustness by employing advanced architectures and data augmentation strategies, reducing the effectiveness of adversarial mechanisms (Xi 2020).

Finally, CV is instrumental in monitoring physical behaviors that may indicate malicious intent. Techniques such as micro-expression analysis, posture recognition, or facial stress detection can be used to identify deceptive activities, which are critical in scenarios like border security or physical access control (Rehman et al. 2022).

By integrating CV into cybersecurity workflows, researchers and practitioners can develop robust defenses against evolving cyber threats. Its ability to extract meaningful insights from complex data makes it a valuable tool in advancing malware detection and enhancing overall cybersecurity, as will be seen in the paper at hand through the proposal of DEFENDIFY framework.

State of the art

In this section, we review the available literature in the field of malware detection. The traditional approach to detect them can be classified into signature-based, anomaly-based and specification based (Talukder and Talukder 2020). It is worth mentioning that with the arrival of ML and DL techniques, this field has experienced a huge change and many of the proposed works use the anomaly detection (AD) paradigm (Sahin and Bahtiyar 2020; El Merabet and Hajraoui 2019). One of the advantages of this approaches is that they can detect new types of malware without collecting its signatures.

Due to the widespread use of x86 architectures, most work on malware detection focuses on this architecture. Thus, the first question that needs to be answered is about the features used to feed ML and DL models for malware detection in such architecture. These features need to be sufficiently representative of the samples to discriminate between malware and goodware. Until now, there were four main groups of features to use in malware detection: Opcode sequence, portable executable (PE) header, Strings, and API sequence (Guo and Fan 2019). Besides, the study presented in (Xue et al. 2019) shows a taxonomy of ML methods for Binary Code Analysis, including an exhaustive list of the features used in malware detection.

In this context, many of the ML techniques currently applied for malware detection are based on these features. For example, Rezaei et al. (2021) propose a novel method that combines DL and ML techniques to learn different embedding representations of both malware and goodware. In particular, the raw bytes of PE header are embedded into the dense neural network (DNN), and the output is passed to a k-means model. The authors highlight the low computational overhead of its solution due to the lightweight network used and the low time required to extract bytes from the PE header. In the same context, several ML-based approaches have been proposed for malware detection by leveraging features extracted from PE headers. For instance, Hussain et al. (2022) introduce a detection system that applies various ML models, including Random Forest, support vector machine (SVM), and Gradient Boosting, to classify executables as clean or malicious based on PE header features. The authors conduct a comparative analysis of these models, highlighting that Random Forest achieves the highest accuracy (99.44%), making it a promising candidate for real-world malware detection applications.

In terms of opcode sequence approaches, several works have been proposed. For example, Lee et al. (2023) proposed a method that extracts fixed-length and low-dimensional features from opcode category information to distinguish between benign and malicious application binaries. The extracted features are evaluated using multiple ML models, including SVM, Decision Tree, and Random Forest, achieving an accuracy of over 98% for both malware detection and classification. The authors highlight the robustness of their approach, demonstrating its effectiveness in identifying different malware families. In Kakisim et al. (2022), the authors proposed Sequential Opcode Embedding-based Malware Detection (SOEMD), a method that captures common malicious patterns using a Random Walk approach for edge and node selection. By constructing a low-dimensional vector space with opcode embeddings, SOEMD

enhances detection efficiency. The model tested in the architecture were K-Nearest Neighbour (k-NN), CNN and LSTM. Experimental results show that the proposed method outperforms baseline approaches, achieving a 100% malware detection rate.

In terms of feature comparison, Balram et al. (2019) presented a detailed study where different ML techniques are tested using PE header and Strings. To be specific, the ML models tested were SVM, Linear Regression (LR), Random Forest (RF), XGBoost. Besides, based on these models, the authors also tested two ensemble models : LR/XGBoost and LR/RF/Naïve Bayes. The authors concluded that models that use string-based features outperform models that use PE-based features. The model that achieved the best performance was the ensemble of LR/XGBoost with an accuracy of 0.980 for string-based features and 0.915 for PE-based features.

Although both classical ML models and traditional features-based DL methods achieve good results, new approaches based on DL techniques are explored in the literature. In particular, convert binary files into images and using them as input of a CNN is achieving good results. In this context, Shaukat et al. (2023) proposed a deep learning-based method that visualizes PE files as colored images and extracts deep features using a fine-tuned deep learning model. These features are then classified using an SVM, eliminating the need for extensive feature engineering. The proposed method achieves 99.06% accuracy on the Malimg dataset and demonstrates superior performance over state-of-the-art approaches, with an average accuracy improvement of 16.56%. This new approaches based on DL models allow the detection of obfuscated malware. This is clearly show by Mercaldo et al. (2023) that proposed a method that converts system call traces from legitimate, malicious, and obfuscated Android applications into images for classification using a CNN. Their experiments demonstrate the resilience of deep learning models in detecting obfuscated malware, highlighting the effectiveness of dynamic analysis combined with CNN-based classification. Additionally, the authors employ explainability techniques to analyze model predictions, ensuring interpretability and robustness. Another example is exposed by Han et al. (2024), where they proposed a deep learning-based method that leverages depth-wise CNN with a spatial attention mechanism to classify malware obfuscated by virtual machine (VM) code. Using a dataset generated with VMProtect, the proposed model is trained on real-world obfuscated malware samples. Experimental results show that the approach achieves nearly 100% accuracy on regular malware classification and 93.55% on obfuscated malware, demonstrating its effectiveness in handling complex obfuscation techniques. Ravi et al. (2022) propose a Multi-View attention-based DL framework for malware detection, leveraging features from PE headers, imports, images, and API calls. Their approach outperforms non-attention-based ML models, achieving 95% accuracy. Additionally, they evaluate malware detection using grayscale images from binary files, obtaining 98% accuracy on a Windows dataset and 97% on an Android malware dataset. Finally, Conti et al. (2022) explore DL models for detecting and classifying obfuscation in Android applications. They evaluate classical ML methods alongside NLP and image recognition techniques on binaries obfuscated using four strategies: Trivial (T), String Encryption (SE), Reflection (R), and Class Encryption (CE). Among the tested models (SVM, DNN, and CNN), CNNs using RGB images achieved the highest F-scores, with a peak of 0.994 for CE. A hybrid approach combining multiple models further improved performance, reaching F-scores of 0.972 (T), 0.991 (SE), 0.980 (R), and 0.998 (CE).

Table 1 compares the solutions discussed in the previous paragraphs together with the solution that we propose. From this comparison, it is evident that our approach is the only one offering a comprehensive framework for malware detection. Specifically, our contribution introduces several key innovations that distinguish it from prior work:

- Use of Transfer Learning (TL) for obfuscated malware detection: Our method applies TL (Ribani and Marengoni 2019) to address the challenge of detecting fully obfuscated malware. This approach enables us to transfer learned representations from pre-trained models on image datasets to the specific task of malware detection, thereby reducing training time and improving overall performance.
- Integration of an entropy-based obfuscation tester: Our framework is unique in adopting a static analysis approach that classifies both obfuscated and non-obfuscated binaries without requiring execution of the binaries, thus mitigating the risks associated with dynamic analysis. Prior to feeding the samples into the CNN model, we perform an entropy test to distinguish between obfuscated and non-obfuscated samples. This step is novel, as no previous work integrates an obfuscation detection module as part of the classification pipeline.
- Detection of fully obfuscated malware using real-world obfuscators: While the works proposed by Conti et al. (2022) and Han et al. (2024) bears some resemblance to our method, there are two major distinctions: i) their solution does not present a complete framework, and ii) they do not leverage TL techniques. Additionally, unlike previous works, including Conti et al. (2022), our approach is the first

Table 1 Comparison of different related works that propose malware detection using ML/DL

Work	Can detect completely obfuscate malware by means of static analysis?	Dataset features	ML/DL model	Year
Balram et al. (2019)	×	Strings PE header	SVM LR RF XGBoost Ensemble (LR/XGboost) Ensemble (LR/RF/NB)	2019
Rezaei et al. (2021)	×	Raw bytes of PE header	DNN K-Means	2021
Ravi et al. (2022)	×	PE header PE imports API calls Bytes of PE header as images	attention-based CNN, DNN, and LSTM	2022
Conti et al. (2022)	✓	Binaries as gray-scale/RGB images Features extracted from the strings Natural language-based features	SVM DNN CNN	2022
Hussain et al. (2022)	×	PE Header	RF SVM Gradient Boosting	2022
Kakisim et al. (2022)	×	OP code	RF K-NN CNN LSTM	2022
Lee et al. (2023)	×	OP code	RF Decision Tree SVM	2023
Shaukat et al. (2023)	×	Binaries as RGB images	CNN as feature extractor and SVM	2023
Mercaldo et al. (2023)	×	system calls traces encoded as images	CNN	2023
Shaukat et al. (2023)	×	Binaries as RGB images	CNN as feature extractor and SVM	2023
Han et al. (2024)	✓	Binaries as RGB images	CNN with attention mechanism	2024
DEFENDIFY	✓	Binaries as grey-scale images	CV architectures powered by TL	2025

to validate the detection of fully obfuscated malware using obfuscators that are widely employed by real-world attackers. Specifically, our solution demonstrates robustness against Shikata Ga Nai, an obfuscation tool frequently used in actual attack scenarios, which has not been addressed in prior studies.

- Some related works use TL in malware detection labors but for different proposals. Jian et al. Jiang et al. (2020) proposed a TL-based solution that aims to detect variations between the representation of an original obfuscated malware and some of its variations, considering that both the original and the obfuscated malware have an image representation. Marastoni et al. Marastoni et al. (2021) proposed a solution that intentionally applies code transformations to malware to build an augmented malware dataset, which is then used to construct a TL-based model to differentiate between obfuscated and no-obfuscated malware samples. Thus, these last related works are focused on classifying representations of malware, instead of a full-fledged solution that not only detects obfuscation but also detects if the sample is or is not malicious as DEFENDIFY does.

In summary, our proposal is the first to offer a fully integrated framework for malware detection that combines TL, a static analysis approach, and an entropy-based obfuscation tester, ensuring robust performance even against fully obfuscated malware, as validated using real-world obfuscation tools.

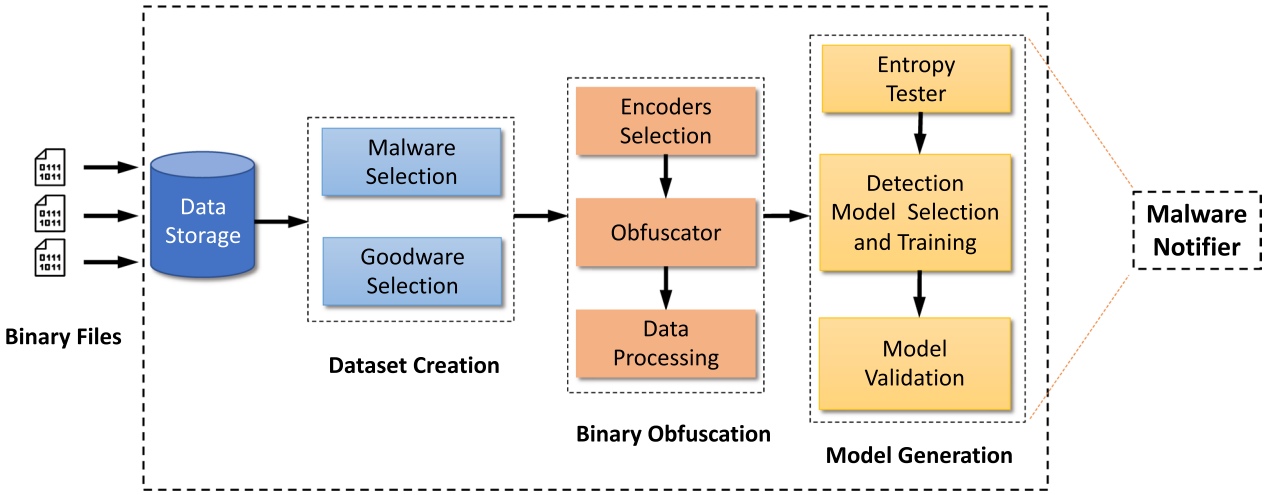

Fig. 1 High-level view of DEFENDIFY, the proposed malware detection framework

DEFENDIFY framework

In this section, the components and modules that comprise the DEFENDIFY framework are detailed. As previously mentioned, the proposed framework utilizes DL and TL technologies to detect obfuscated malware. DEFENDIFY consists of three modules: Dataset Creation ("Dataset creation" section), Binary Obfuscation ("Binary obfuscationy" section), and Model Generation ("Model generation" section). Each of these modules is further divided into various components that work in coordination to efficiently detect obfuscated malware. A high-level description of these modules and components can be observed in Fig. 1.

Dataset creation

In order to perform its detection activities, DEFENDIFY needs to recollect and store a large number of samples as input. To achieve this, the Dataset Creation module comprises a Malware Selection and a Goodware Selection component. The first is responsible for gathering malware binaries, while the second is in charge of collecting goodware binaries, as detailed next. Although the current section describes the generic procedure for the dataset composition, a description the datasets used in our experiments which impact directly the obtained performance metrics is detailed in "Datasets used in the experiments" section.

Malware selection

The Malware Selection component is in charge of one of the most important actions in DEFENDIFY, as it collect, filter and select the final set of malicious binaries that will be used in the next training steps. Even if there are so many repositories where malware may be collected, it is important to consider key features as the volume of samples, diversity in the types of malware, trust in the source, integrity of the samples, among others.Thus, there is different malware repositories (e.g., Malware The Zoo,[1] vxvault,[2] vxunderground[3] or VirusShare[4]) which may be a good sources to be consumed by the Malware Selection component. From the previously mentioned repositories, VirusShare is particularly interesting as it is an open repository of malware samples that is intended to provide security researchers with samples of real malicious code. This repository was created to provide access to samples of live malicious code with the aim of helping researchers, forensic analysts, and similar professionals, however, to prevent the spread of the samples distributed in this repository, the access to those samples is delivered under individual request. As we will see in "Datasets used in the experiments" section, a dataset of 15821 malware samples was employed in our experiments.

Goodware selection

The Goodware Selection component provides DEFENDIFY the capacity to collect, filter and select the final set of non-suspicious binaries which will be used in the next training steps. There is a big diversity of goodware available online, downloadable through the original software publisher or third parties, addressed for different purposes, and types of public. Regarding the purpose, there are many e.g. libraries and handles used by operating

[1] https://github.com/ytisf/theZoo.
[2] http://vxvault.net/ViriList.php.
[3] https://vx-underground.org/.
[4] https://virusshare.com.

systems, end user utilities, server software, middleware, etc. Thus, acquisition of goodware samples may be achieved through from the most naive approach based on a manual download of software typically used by users, until more automated ways that collect large amounts of goodware using application stores such as the one available at Microsoft Store.[5]

One interesting way of composing a proper goodware dataset, usable in DEFENDIFY, is including binaries that are found typically in an operating system, i.e. system files of Windows 10/11, and also including binaries associated to end user utilities, i.e. office applications. As we will see in "Datasets used in the experiments" section, a dataset of 15628 goodware samples was employed in our experiments.

Binary obfuscation

Once the malware and goodware samples have been recollected, the subsequent step entails the obfuscation of the samples. To this extent, the Binary Obfuscation module is responsible for obfuscating the samples gathered by the Dataset Creation module. In particular, this module comprises the Encoders Selection, Obfuscator, and Data Processing components.

Encoders selection

This component is in charge of selecting the encoders used to obfuscate both malware and goodware samples. In this regard, note that obfuscation may be achieved with different types of encoders, mainly metamorphic and polymorphic ones. Among the metamorphic ones, the block-based XOR encoder (Ceschin et al. 2021) is one of the most popular for its simplicity, because it reuses the idea of a mathematical XOR operation to perform transposition encoding, defeating in that way, the regular anti-malware mechanism based on signatures. This encoder uses the property of an XOR operation, i.e., $A \oplus B \oplus B = A$, and processes the binary per block. Thus, the steps to implement block-based XOR are resumed next:

1. Define a key k_j to be used in the current block encoding.
2. Read and identify each character c_i of the software shell code.
3. XOR each c_i using k_j.

One of the goals of an encoder is to obfuscate the key shellcode operations existing in a binary. However, as XOR encoding is metamorphic, it causes known suspicious shellcode instructions, such as the corresponding to *exec*/*bin*/*bash* or /*tcp*/ *ip port*, become encoded to well-known outputs. From an offensive perspective, a well-known output represents a bad signature because it can be easily identified by anti-malware solutions.

Metamorphic encoders evolved to polymorphic ones, meaning that two equal files received as input by the encoder will not produce the same output file, being Shikata Ga Nai one of the most representative and used polymorphic encoders (Farley and Wang 2014). Shikata Ga Nai is a polymorphic XOR additive feedback encoder that uses metamorphic techniques, e.g., reordering and substitution, in conjunction with a chained self-modifying key to produce a different output each time it is applied, bypassing detection mechanisms based on signature recognition (Graham 2021). Given that Shikata Ga Nai uses a chained self-modifying key through additive feedback, in case the input to be decoded or the keys are incorrect at any iteration, all subsequent outputs will be incorrect. The steps to implement Shikata Ga Nai are resumed as follows:

1. Define a key k_j
2. Get the value of the Extended Instruction Pointer *EIP* register using Floating-Point Unit (FPU) instructions.
3. Enter in a loop that runs over a new instruction *EIP*+`0xF`.
    (a) Replace instruction at *EIP*+`0xF` applying an XOR operation between *EIP*+`0xF` and k_j.
    (b) Change the key k_j by adding it with the result of the previously modified instruction at *EIP*`0xF`.

Although DEFENDIFY is generic enough to select any encoder in the literature, this work is focused on XOR and Shikata Ga Nai encoders. The reason for this decision was to cover both metamorphic and polymorphic encoders, and because they are encoders employed commonly Burita and Le (2021).

Obfuscator

Once goodware and malware samples are collected by the Dataset Creation module, these samples must be obfuscated for the encoders selected for DEFENDIFY: XOR and Shikata Ga Nai. There are different tools to execute software obfuscation, being the Metasploit Framework (MF)[6] one of the most recognized ones. MF is an open-source penetration testing framework used by several cybersecurity experts, including white-hat and black-hat

[5] https://apps.microsoft.com/store/apps/windows.

[6] www.metasploit.com/.

Table 2 Image width according to the sample size

Binary file size (KiB)	Image width (px)
Less than 10	32
10–30	64
30–60	128
60–100	256
100–200	384
200–500	512
500–1000	768
More than 1000	1024

hackers, that allows executing security testsand developing and testing payloads and exploit codes. MF contains multiple encoders, Shikata Ga Nai and XOR encoders being the most representatives. Also, MF provides different tools such as `msfvenom`, an open-source payload generator that combines other tools like `msfpayload` and `msfencode` to create an obfuscated binary.

Data processing

DEFENDIFY relies on computer vision DL algorithms;thus, the primary processing that data require is their transformation into images, which can be consumed by convolutional models employing TL. This component can be implemented through different programming languages, following a procedure similar to the next one: (1) the sample is loaded and read byte by byte, storing each byte-associated numerical value in a list; (2) a default image size is calculated (*width, height*) depending on the list length, i.e., the file size; (3) a gray-scale image is created by copying the bytes one by one into an empty image, with every byte corresponding to one image pixel. For example, suppose that we have a 4-byte binary file, and after reading its bytes, the following values are obtained: [0, 64, 128, 255]. If we want to arrange these values into a 2×2 size image, we will obtain an integer matrix as the one shown in Eq. (1), which represents an image that is saved and used later to feed the models.

$$Img = \begin{bmatrix} 0 & 64 \\ 128 & 255 \end{bmatrix} \tag{1}$$

The size of the image associated with each sample is determined using a custom function that uses the number of bytes in the sample as a parameter, as shown in Table 2. Based on the received list of bytes, the function defines a corresponding image width, and its height is calculated by dividing the file size by the image width as proposed in Bensaoud et al. (2020).

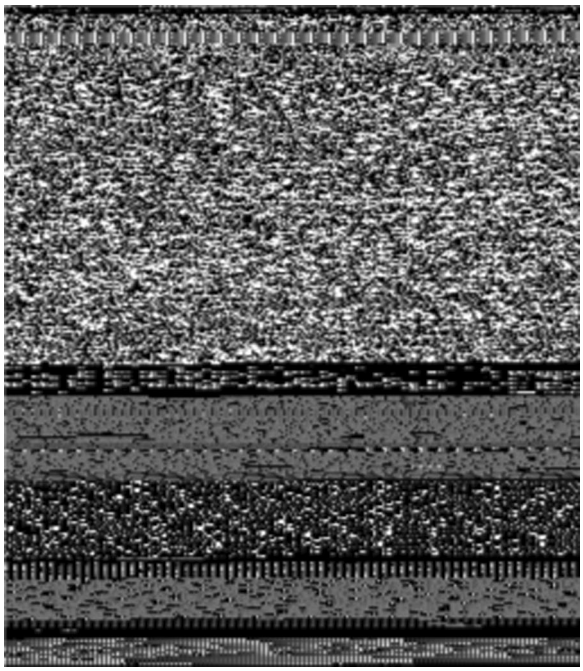

Fig. 2 Example of gray-scale image obtained from an encoded sample

A Python library that can generate an image from a Python list of bytes is *Pillow*.[7] Using this library, a binary image was created for each of the files included in the training dataset, as explained in "Dataset creation" section, obtaining three new training datasets: i) an image dataset obtained from samples obfuscated with Shikata Ga Nai, ii) an image dataset obtained from samples obfuscated with XOR, and iii) an image dataset obtained from samples not obfuscated. When DEFENDIFY gets deployed, this procedure would be applied to any sample that needs to be classified after passing the entropy tester described in "Entropy tester" section.

For example, the gray-scale image in Fig. 2 was generated using the procedure previously described over a randomly selected encoded binary file and is an example of the possible outputs that can be obtained with the algorithm.

Model generation

This module is responsible for training and validating a model to detect non-obfuscated and obfuscated malware. We propose different components to process and classify new software samples. These components are as follows and described next: Entropy Tester, Detection Model Selection and Training, and Model Validation.

[7] https://python-pillow.org/.

Entropy tester

To determine whether a particular sample is obfuscated or not, regardless of it is a malware or not, DEFENDIFY incorporates an Entropy Tester component that computes the entropy Lyda and Hamrock (2007) to help identify when a sample is allegedly encoded with Shikata Ga Nai, XOR or non-encoded at all. Shannon entropy can be interpreted as describing the average level of uncertainty of a text using an alphabet. Formally, the entropy H of a string x is defined by Eq. (2):

$$H(x) = -\sum_{i=1}^{N} p(i) log_2(p(i)) \quad (2)$$

where

- N is the size of the alphabet.
- $p(i)$ is the frequency of the character i of the alphabet in the string x.

The results of this component are used later in the data flow to choose one of the classifiers available to process the sample. The entropy tester is implemented using a classifier trained with a dataset of entropies calculated for non-obfuscated samples and samples obfuscated with Shikata Ga Nai and XOR. Details on the implementation of the entropy tester used in this paper are described in "Results of the entropy tester" section.

Model selection and training

This component is in charge of selecting the appropiate models to detect both non-obfuscated and obfuscated malware. Since our goal is to detect both types of malware, this component selects two models. The first one is employed when the entropy tester component detects that a particular sample is obfuscated, while the second model is used in the opposite case.

This component is primarily focused on DL models, which are widely used in the CV ecosystem. In particular, these models are based on CNN models that use convolutional layers to extract relevant features from an input matrix (i.e., an image), group layers to reduce computational requirements, and multiply stacks of these types of layers to update the network weights.

Furthermore, the use of these models allows DEFENDIFY to take advantage of the pre-trained learning of such models and,thus, apply an effective TL mechanism. In this context, it is easy to spot that TL offers two main advantages. The first one is that TL allows DEFENDIFY to avoid costly processes to train such CNN models. The second one is that TL does not require a high number of samples for training. Since DEFENDIFY is a generic framework that has the potential to be deployed in a wide range of devices;thus, we must consider scenarios with limited capacities for training. Additionally, using a TL strategy allows DEFENDIFY to require fewer training samples.

To apply TL successfully, DEFENDIFY performs the following actions:

1. Set a CNN pre-trained with the ImageNet-1K[8] database
2. Freeze all layers except the last layer (head)
3. Train the head layer using the Leslie Smith's cycle policy Smith (2017)
4. Unfreeze all layers
5. Perform a new training using Leslie Smith's one cycle policy over the entire network

ImageNet-1K was selected to support the TL strategy used by DEFENDIFY as this is one of the largest organized image databases with more than 1,281,167 training images from 1000 categories, 50,000 validation images, and 100,000 test images.

This Model Selection and Training component considers several architectures to achieve optimal detection results. In this work, two families of architectures were explored:

- ResNet: The ResNet Géron (2019) architecture is a family of CNN architectures, having as its main characteristic the use of at least one Residual Unit (RU), i.e., it uses learning residual functions in every layer instead of the traditional unreferenced functions common in regular neural network models. A RU consists of a pair of consecutive convolutional layers, where the output of the second layer is added to the original input of the first layer, forming a skip connection as observed in Fig. 3. An activation function, based on a Rectifier (ReLU) and a Batch Normalization (BN), is also integrated into each skip. All ResNet architectures use RU's, varying in the amount they use, e.g. ResNet18 has a total of 18 convolutional layers, and ResNet34 has 34. Historically, these residual networks have demonstrated much better results than those obtained using other deep architectures.
- EfficientNet: Another family of deep neural network architectures is named EfficientNet Tan and Le (2019) and its most distinguished feature is the use of a constant set of scaling coefficients to uniformly scale the Width (W), Depth (H), and Resolution (C) with the aim of modifying or adapting a new CNN architecture and optimizing the available computa-

[8] https://www.image-net.org/.

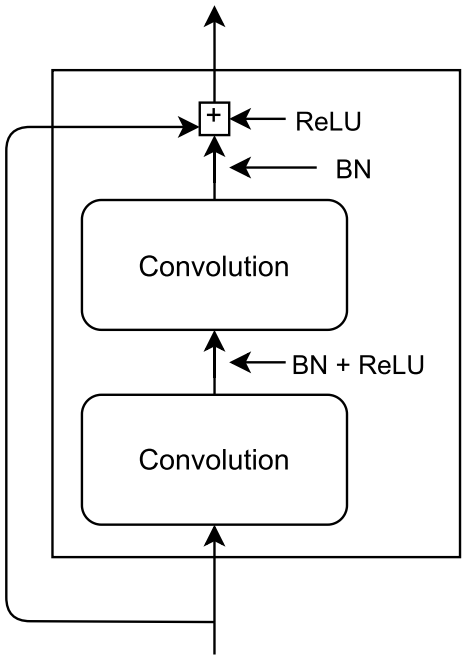

Fig. 3 Residual unit diagram

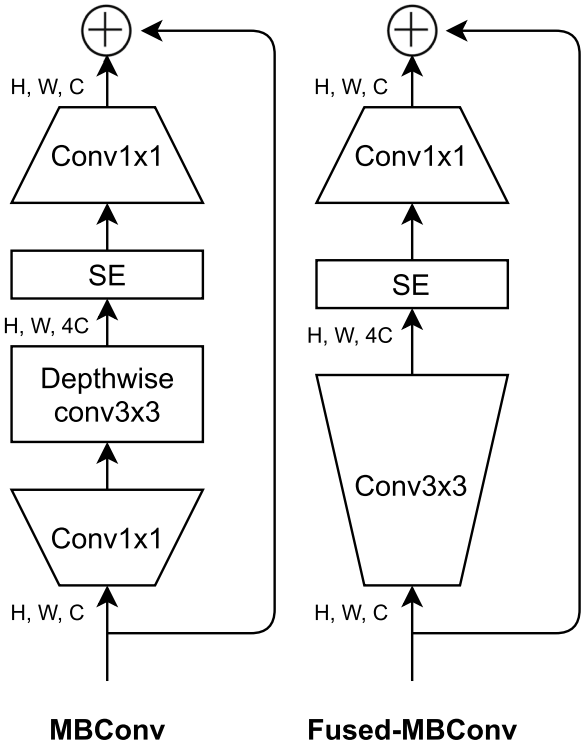

Fig. 4 Diagram for MBConv and Fused-MBConv operators

tional resources. In this regard, Mingxing Tan and Le (2020) indicate that these three aspects (W, H, C) should be modified proportionally, known as compound scaling, consider one with respect to the others. Thus, Tables 3 and 4 indicate the baseline model of proportionality to be used for EfficientNet B0 and EfficientNet V2-S, respectively.

In particular, four different architectures were considered: Resnet18, Resnet34, EfficientNetB3, and EfficientNetV2S. These architectures were selected because they achieve top-tier results in image classification tasks while keeping the number of parameters as small as possible. This last part is essential for our proposal, as the models should be as lightweight as possible to be efficiently deployed in a production environment.

EfficientNetB0 differs from EfficientNetV2S because the first one is purely based on Mobile inverted Bottleneck Convolution (MBConv) operators, while the second one mixes those operators with the Fused-MBConv operator. The MBConv and Fused-MBConv operators are represented in Fig. 4. MBConv differs from Fused-MBConv because it first applies a convolutional 1×1 layer, and then it applies a depth-wise convolutional 3×3 layer, handling each channel independent of the others in comparison to a normal convolution. Meanwhile, Fused-MBConv fuses those 2 first layers, 1×1 and 3×3, into a single convolutional 3x3 layer. Both MBConv and Fused-MBConv integrate a Squeeze-and-Excitation (SE) block

Table 3 EfficientNet B0 architecture

Stage i	Operator $\hat{\mathcal{F}}_i$	Resolution $\hat{H}_i \times \hat{W}_i$	#Channels $\hat{C}_i$	#Layers $\hat{L}_i$
1	Conv3x3	224×224	32	1
2	MBConv1, k3x3	112×112	16	1
3	MBConv6, k3x3	112×112	24	2
4	MBConv6, k5x5	56×56	40	2
5	MBConv6, k3x3	28×28	80	3
6	MBConv6, k5x5	14×14	112	3
7	MBConv6, k5x5	14×14	192	4
8	MBConv6, k3x3	7×7	320	1
9	Conv1x1 & Pooling & FC	7×7	1280	1

that improves channel interdependency. As a reference, EfficientNetB3 is a variation that may be scaled from B0 using the following parameters: depth ($d = \alpha^{\phi}$), width ($w = \beta^{\phi}$), and resolution ($r = \gamma^{\phi}$), such that $\alpha\beta^2\gamma^2 \approx 2$ and $\alpha, \beta, \gamma \geq 1$. The values for α, β and γ are found to solve an optimization problem, and Tan and Le (2020) found that the best values for EfficientNetB0 would be $\alpha = 1.2$, $\beta = 1.1$, and $\gamma = 1.15$. Thus, the Width (W), Depth (H), and Resolution (C) coefficients for EfficientNetB3 would be replacing $\phi = 3$.

Table 4 EfficientNetV2 architecture (V2-S variation) Tan and Le (2021)

Stage i	Operator $\hat{\mathcal{F}}_i$	Stride $\hat{H}_i \times \hat{W}_i$	#Channels $\hat{C}_i$	#Layers $\hat{L}_i$
0	Conv3x3	2	24	1
1	Fuxed-MBConv1, k3x3	1	24	2
2	Fused-MBConv4, k3x3	2	48	4
3	Fused-MBConv4, k3x3	2	64	4
4	MBConv4, k3x3, SE0.25	2	128	6
5	MBConv6, k3x3, SE0.25	1	160	9
6	MBConv6, k3x3, SE0.25	2	272	15
7	Conv1x1 & Pooling & FC	–	1792	1

Model validation

This component is in charge of evaluating the previous trained models to determine which model is optimal for real-world application deployment. The primitive metrics employed to measure the performance of DEFENDIFY are specified in the following list:

- True Positive (*TP*): This value indicates the number of malicious programs that were recognized as such by the model.
- True Negative (*TN*): This value indicates the number of well-intended programs that were recognized as such by the model.
- False Positive (*FP*): This value indicates the number of well-intended programs that were falsely classified as malware by the model.
- False Negative (*FN*): This value indicates the number of malware programs that were falsely classified as well-intended by the model.

These four values will fully describe the test results of every considered model. The definitions imply that the chosen model should minimize *FP* and *FN* as much as possible and maximize *TP* and *TN*. In addition, it is reasonable to use the following classification metrics to measure the performance of deep architectures:

- Accuracy: It is the most straightforward metric used to validate ML models, and has the advantage of considering all previously defined primitive metrics. Accuracy measures the proportion of correctly classified programs against the entire testing set. It can be calculated as follows:

$$Accuracy = \frac{TP + TN}{TP + TN + FP + FN} \tag{3}$$

- Precision: It summarizes the model's performance in terms of malware detection. In other words, it measures the proportion of malicious programs against all programs classified as malware. This can be expressed as follows:

$$Precision = \frac{TP}{TP + FP} \tag{4}$$

Then, maximizing the precision increments the model's probability of correctness when indicating that a given binary contains malicious code.

- Recall: The recall is also referred to as the True Positive Rate (TPR), or sensitivity. It measures the proportion of malware classified as such by the model. This value is calculated as follows:

$$Recall = \frac{TP}{TP + FN} \tag{5}$$

So, the recall indicates the model's probability of correctly detecting a malicious program.

- F_1 Score: This value is a combination (more exactly, the harmonic mean) of precision and recall, and it is one of the most useful metrics when comparing the performance of two different classification models. It is calculated as follows:

$$F_1 = 2 \times \frac{Precision \times Recall}{Precision + Recall} \tag{6}$$

The F_1 score presents high values (close to 1) when both precision and recall are high, so the overall behavior of the classification model is adequate.

Experiments

This section contains the experiments performed to validate the DEFENDIFY framework presented in "DEFENDIFY framework" section. First, the datasets gathered by DEFENDIFY, which were used during the experiments, are described. Then, the results obtained with encoded and non-encoded samples are shown and discussed with their corresponding classification metrics. Finally, a prototype of DEFENDIFY framework as an interactive and public web application is explained.

All the experiments described in the current section were executed in a Nvidia Saturn Cloud T4-4XLarge instance specially dedicated to the experiments execution, with the following features: 64 GB RAM, 16 CPU cores, 1 GPU, and 1 TiB disk space. A project repository[9] is also available that contains all the code developed to

[9] https://github.com/Nirogu/ObfuscatedMalwareDetection.

execute the experiments presented in this paper, as well as detailed instructions on how to obtain the datasets.

Datasets used in the experiments

To create the dataset, the Dataset Creation module gathered samples using publicly available binary files, from multiple online sources. On the one hand, the malware files were obtained from VirusShare. From this repository, a zip file containing 15821 viruses for x86 architecture was downloaded and labeled as malware to train the subsequent DL models.

On the other hand, the goodware files were obtained from two different sources: (1) 15,171 sample files from the folder *System32* from a Windows 10 operating system, which contains many essential binary files for the operating system, (2) 457 sample files from an app installer called PortableApps[10] that automatically downloads selected applications like games, working tools, among others, and extracts their *.exe* and *.dll* files.

Each binary included in the previously composed dataset was obfuscated using the Metasploit Framework, with the `msfvenom` tool, to obtain their obfuscated representation for both encoders, Shikata Ga Nai and XOR. The syntax of the instruction used to execute `msfvenom` is presented in the following two command-line instructions:

```
msfvenom -p generic/custom PAYLOADFILE=virus.exe
-a x86 --platform windows -e x86/shikata_ga_nai
-o virusencoded.exe
```

```
msfvenom -p generic/custom PAYLOADFILE=virus.exe
-a x86 --platform windows -e x86/bloxor
-o virusencoded.exe
```

In this way, three training datasets were conformed in our experiments: (1) dataset of 31,449 samples codified with Shikata Ga Nai, (2) dataset of 31,449 samples codified with XOR, and (3) dataset of 31,449 samples not codified. Table 5 summarizes the number of malware and goodware samples per dataset.

Given that we are interested in identifying as most categories of malware as possible, the DEFENDIFY malware dataset is composed by several variations of computer malicious executable files. In order to obtain the exact malware category for these samples, the AVG antivirus software was used, and 121 different categories were identified. The percentage of existence of each malware category is presented in Table 6. For simplicity's sake, only the 20 most frequent categories are presented in the Table 6 which represent the 75% of the total dataset malware, and the 101 remaining categories are clustered in the category *Others*.

Table 5 Malware and goodware samples per dataset

Encoding	File type	Samples
None	Goodware	15,628
	Malware	15,821
XOR	Goodware	15,628
	Malware	15,821
Shikata Ga Nai	Goodware	15,628
	Malware	15,821

Table 6 Malware categories included in the dataset generated by the Dataset Creation module

Category	Distribution percentage (%)
JS/HiddenLink.B	11.1111
JS/HiddenLink	9.9327
Win32/DH	6.2290
BundleApp_r.AV	5.5556
HTML/Framer	5.2189
JS/HiddenLink.A	4.3771
Generic.C6A	4.3771
InstallBrain.BH	4.0404
JS/Redir	4.0404
Generic.834	3.8721
AdInject	2.6936
JS/Exploit	2.6936
Generic.715	2.3569
AdPlugin.BRP	2.1886
JS/Clicker	1.8519
Generic_r.VD	1.1785
Win32/Cryptor	1.1785
Generic.AF2	1.0101
Downloader.DDT	0.8418
Win32:Adware-DIW	0.8418
Others	24.4103

Results of the entropy tester

Before detecting whether a given binary file contains malicious code, it is necessary to first detect if it was obfuscated or not. This can be seen as a classification problem where the input is an executable file and the output is a label indicating that it was, in fact, obfuscated (i.e., XOR, Shikata Ga Nai), or a label noting that the file is not obfuscated.

[10] https://portableapps.com.

Table 7 Classification metrics for the entropy test set with different classifiers

Algorithm	Accuracy (%)
Logistic regression	72.60
LDA	72.96
QDA	68.35
SVM	74.04
KNN	89.45
Naïve Bayes	68.35
Random forest	87.08
LightGBM	89.81

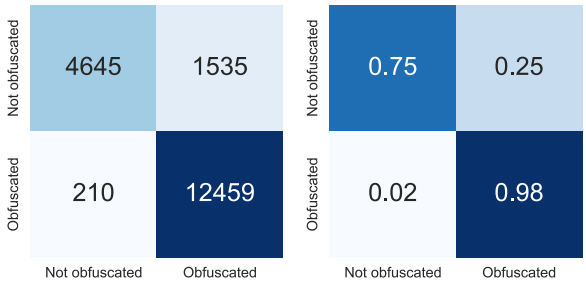

Fig. 5 Obfuscation classifier confusion matrix. Raw results (Left). Results normalized to the row values (Right)

To achieve this, the Entropy Tester component used the entropy of the files bytes to identify the obfuscation type. This gives us one input variable to predict the file label with. In this way, we can test different well-known classification algorithms in order to find out which one presents the best results. Therefore, we partitioned the whole dataset into a training set (80% of the data) and a testing set (20%) of the data, which we used to check the binary accuracy (Obfuscated vs Non-obfuscated) of the following classifiers: logistic regression Cox (1989), linear discriminant analysis (LDA) Fisher (1936), quadratic discriminant analysis (QDA), support vector machine (SVM) Platt (2000), k-nearest neighbors (KNN) Cover and Hart (1967), Gaussian Naïve Bayes Ben-Bassat et al. (1980), Random Forest Breiman (2001), and LightGBM Ke et al. (2017). The results are presented in Table 7.

As seen, the KNN classifier outperforms the other algorithms (except for LightGBM) by achieving high accuracy in spite of its simplicity. It manages to present better results than those by widely used classifiers like SVM or Random Forest, and has a similar accuracy to that of LightGBM, which is a much more complex model. Therefore, in order to improve the classification results even more, it is possible to adjust the KNN algorithm hyper-parameters to better fit the training data. In particular, we reviewed the number of neighbors to use (k), and whether to weight the neighbor points in an *uniform* manner, where all observed points have the same importance, or taking the *distance* into account, where closer points are more important. Since this is an uni dimensional dataset with the entropy as its variable, the only distance used is the absolute value. To find the best hyper-parameter set, we ran a search using a tree-structured Parzen Estimator (TPE) Bergstra et al. (2011), with which we obtained the optimal hyper-parameters $k = 27$ and uniform weights by achieving a test accuracy of 90.74%, which is the final test result for the entropy tester. The detailed classification results are presented in Fig. 5 where the X axis represents the actual class and the Y axis represents the class predicted by the entropy tester over the 18,869 samples that composes the testing dataset ($31,449 * 3 * 20\% = 18,869$). The entropy tester could not processed 20 of the 18,869 samples originally included in the testing dataset, resulting in a total number of classified samples of 18,849.

In this way, we can know the presence of obfuscation for a given file, before classifying it as goodware or malware.

Results with non-obfuscated samples

The four models described in "DEFENDIFY framework" section (ResNet18, ResNet34, EfficientNetB3 and EfficientNetV2S) were trained with 80% of the available non-encoded data and tested with the remaining 20%. The results of these tests can be seen in the confusion matrices presented in Fig. 6, where the X axis represents the actual class and the Y axis represents the class predicted by the models over the 6289 samples that composes the testing dataset ($31,449 * 20\% = 6289$). The models were unable to process 40 of the 6289 samples originally included in the testing dataset, resulting in a total number of classified samples of 6249.

In addition, the classification results obtained for each model are summarized in Table 8, which indicates the relation in terms of percentage for each predicted class, dividing the number of samples of each "predicted class" by the number of samples of each "actual class", per each tested model.

The results for each metric defined in "Model validation" section, calculated for each of the models considered, are presented in Table 9.

As seen in the testing results, all the models present noticeable good results. However, ResNet18 stands out because it is the smallest network in terms of components, compared to other models considered, while maintaining a 99.34% of accuracy. For such a reason, ResNet18 could be considered as the best option to build

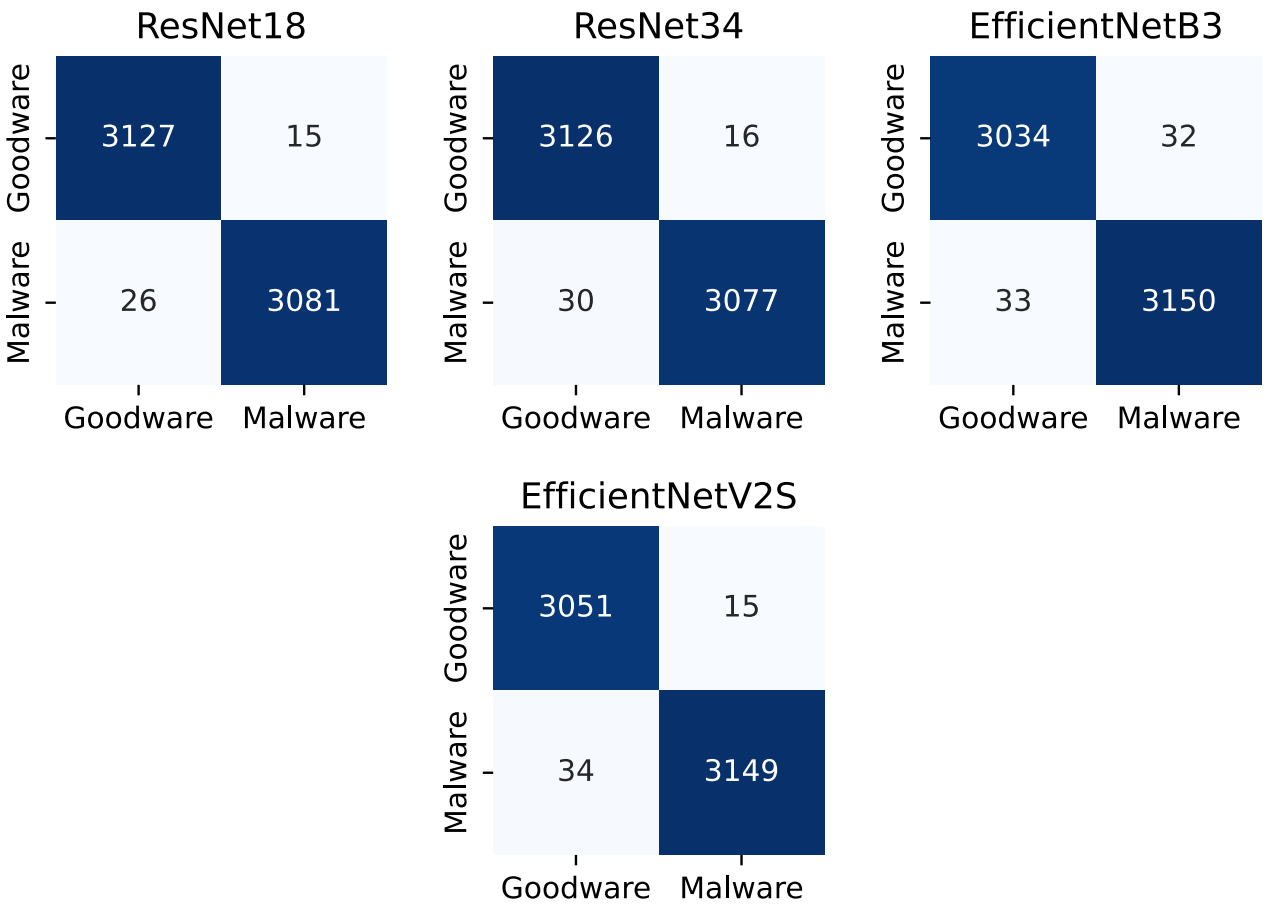

Fig. 6 Confusion matrices for not obfuscated samples

Table 8 Results obtained in the classification of non-obfuscated samples

	Predicted class			
Actual class.	**Goodware**		**Malware**	
Goodware	ResNet18	99.52%	ResNet18	0.48%
	ResNet34	99.49%	ResNet34	0.51%
	EfficientNetB3	98.96%	EfficientNetB3	1.04%
	EfficientNetV2S	99.51%	EfficientNetV2S	0.49%
Malware	ResNet18	0.84%	ResNet18	99.16%
	ResNet34	0.97%	ResNet34	99.03%
	EfficientNetB3	1.04%	EfficientNetB3	98.96%
	EfficientNetV2S	1.07%	EfficientNetV2S	98.93%

a classifier able to differentiate malware from goodware with non-obfuscated binaries.

Table 9 Metrics obtained in the classification of non-obfuscated samples

Model	**Metric**			
	Accuracy (%)	**Precision (%)**	**Recall (%)**	**F1 (%)**
ResNet18	**99.34**	99.52	**99.16**	**99.34**
ResNet34	99.26	99.48	99.03	99.26
EfficientNetB3	98.96	98.99	98.96	98.98
EfficientNetV2S	99.22	**99.53**	98.93	99.23

Bold values are the highest values obtained per each metric

Experiments with XOR and Shikata Ga Nai obfuscated samples

The training and testing procedure was repeated with the XOR and Shikata Ga Nai samples combined for every model. The results of these tests can be seen in the confusion matrices presented in Fig. 7, where the *X* axis represents the actual class and the *Y* axis represents the class predicted by the models over the 12,579 samples that composes the testing dataset

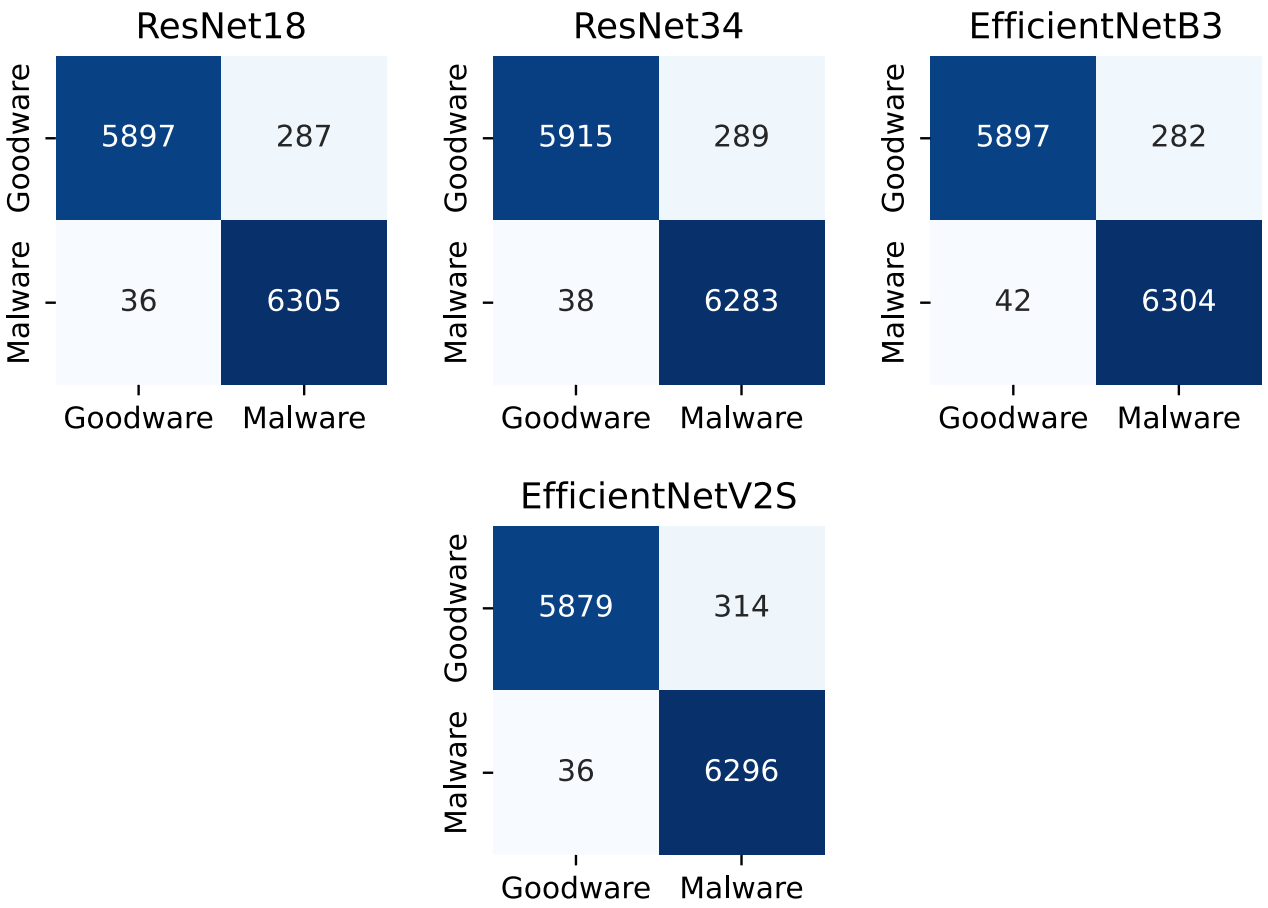

Fig. 7 Confusion matrices for XOR and Shikata Ga Nai samples

Table 10 Results obtained in the classification of XOR and Shikata Ga Nai samples

	Predicted class			
Actual class.	**Goodware**		**Malware**	
Goodware	ResNet18	95.36%	ResNet18	4.64%
	ResNet34	95.34%	ResNet34	4.66%
	EfficientNetB3	95.44%	EfficientNetB3	4.56%
	EfficientNetV2S	94.93%	EfficientNetV2S	5.07%
Malware	ResNet18	0.57%	ResNet18	99.43%
	ResNet34	0.60%	ResNet34	99.40%
	EfficientNetB3	0.66%	EfficientNetB3	99.34%
	EfficientNetV2S	0.57%	EfficientNetV2S	99.43%

Table 11 Metrics obtained in the classification of XOR and Shikata Ga Nai samples

Model	**Metric**			
	Accuracy (%)	**Precision (%)**	**Recall (%)**	**F1 score (%)**
ResNet18	**97.42**	95.65	**99.43**	**97.5**
ResNet34	97.39	95.6	99.4	97.46
EffientNetB3	97.41	**95.72**	99.34	97.49
Efficient-NetV2S	97.21	95.25	**99.43**	97.3

Bold values are the highest values obtained per each metric

$(31,449 * 2 * 20\% = 12,579)$. The models were unable to process 54 of the 12,579 samples originally included in the testing dataset, resulting in a total number of classified samples of 12,525.

In addition, classification results obtained for each model are summarized in Table 10, which indicates the relation in terms of percentage for each predicted class, dividing the number of samples of each "actual class" by the number of samples of each "predicted class", per each tested model.

The results for each metric defined in "Model validation" section, calculated for each of the models considered are presented in Table 11.

As seen in the testing results, all the models present noticeable good results. However, ResNet18 stands out

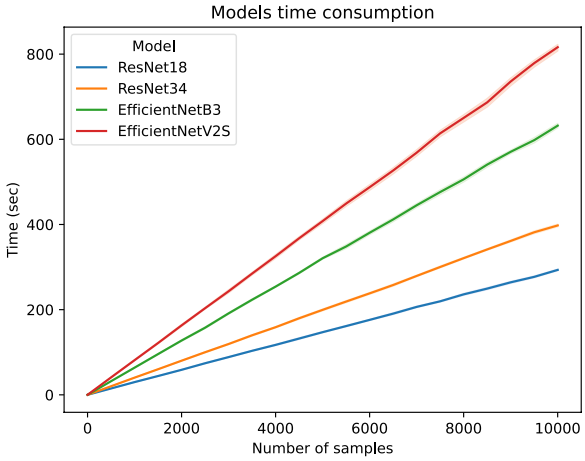

Fig. 8 Time consume, in seconds, registered by each CNN model

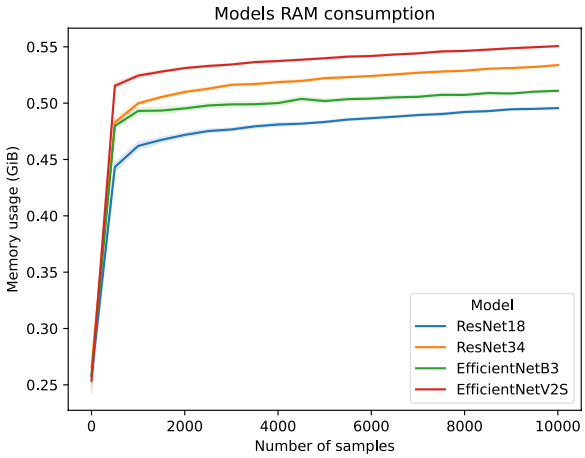

Fig. 9 RAM consume, in GiB, registered by each CNN model

because it is the smallest network in terms of components, compared to other models considered, while maintaining a 97.42% of accuracy. For such a reason, ResNet18 could be considered as the best option to build a classifier that can differentiate malware from goodware with XOR and Shikata Ga Nai obfuscated binaries.

Computational costs

An analysis of computational costs for our proposed solution was done for each of the four CNN architectures presented before. The experiments consist of classifying a batch of images with a size varying between 1 samples and 10^4 Kb samples, while measuring the time and memory used to fulfill the task. To reduce randomness in the results, each batch size was tested with every model 20 times. In Figs. 8 and 9, the measurements of time and RAM are presented, with the mean of all 20 repetitions highlighted as the darker lines, and the 95% confidence interval for these repetitions is drawn as the lines' light contours.

Figure 8 shows the inference time as a linear function with the number of processed samples as the independent variable, and the slope depends on the CNN architecture used. In particular, we can observe that the smallest line slope is associated with ResNet18, which means that this model is the fastest. Besides, the linear behavior indicates stability, because feeding more samples to the program will not make it run indefinitely for many seconds.

Figure 9 shows memory consumption as a logarithmic-like function, although when the number of samples is large enough (more than 1000 approximately) it can be considered constant in practice, with a value depending on the CNN architecture. For example, ResNet18 has the lowest RAM usage with a value of approximately 0.47 GiB. As the growth of memory consumption is too slow to be relevant, the proposed solution is also stable in relation to RAM usage.

Finally, it is worth noting that the 95% confidence interval is too small even to be clearly seen in the graphics. This means that the experiment results are trustworthy and can be easily reproducible in a similar computational environment.

Prototyping DEFENDIFY

Modern and visionary societies need to be built over strong foundations, and in this regard, NSAI (Network System of Artificial Intelligence) is one of the most prominent keystones (Song et al. 2022). NSAI is a paradigm where AI is immersive in all components of a network system, including edge devices, communication nodes and on-premise or cloud-based servers. NSAI architecture is composed of 4 tiers: physical network (PN), service-customized network (SCN), generalized smart service (GSS), and application (APP).

In particular, GSS is one of the most critical tiers, as it may be seen as an intermediate tier that facilitates applications to operate without being aware of the complexity of communication and computing below. To achieve the previous goal, GSS provides application programming interfaces (API) to be consumed by applications in the upper tier. GSS also has the autonomy to reconfigure upper and lower components, i.e. APP and SCN settings, to guarantee reliable services in a smart city.

DEFENDIDY may be integrated into GSS as a cybersecurity service as it provides an AI-enabled smart service capable of detecting obfuscated malware, independent of its location in a distributed network. This service may work by automating the detection of malware flowing to user devices, or it may also be consumed on demand through a smart service API that allows users to upload files to it and trigger a classification process as

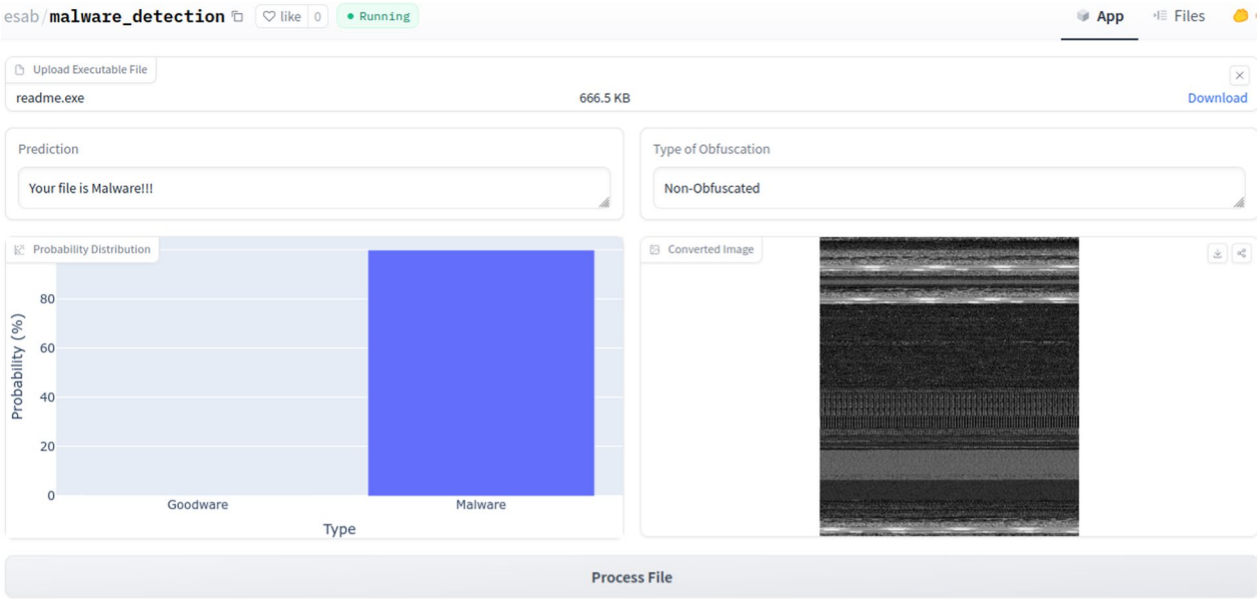

Fig. 10 Web interface with the deployment of DEFENDIFY

described in "DEFENDIFY framework" section. DEFENDIFY would inform users about the analysis results and the probabilities that the provided sample is goodware or malware. Also, it might be appropriate to have the storage available to save new malware samples uploaded by users, allowing the database to grow over time, improving the detection models.

For the sake of contributing a functional prototype that shows in an interactive way the functioning of our proposal, the deployment of DEFENDIFY was tested with a minimalist private web application made using the Python *dash*[11] library. This web application consists of an on-demand simple interface where the user can upload an executable file, that is then processed as indicated in the data flow depicted in Fig. 1. Figure 10 shows a screenshot of the web application[12] that runs the proposal described in this paper.

Furthermore, DEFENDIFY's lightweight architecture and efficient models, such as ResNet18, make it well-suited for potential deployment on resource-constrained edge devices commonly found in smart city environments. For example, the framework could be adapted to run on platforms like Raspberry Pi 5 or NVIDIA Jetson Orin, which are popular choices for IoT and edge AI applications. Deploying DEFENDIFY on these edge devices would allow real-time malware detection closer to the data source, reducing latency, and enhancing the overall security posture of smart city networks.

Discussion

This work presents DEFENDIFY, a novel framework for detecting obfuscated malware using computer vision and deep learning techniques. The results demonstrate that DEFENDIFY is able to detect both obfuscated and non-obfuscated malware samples accurately.

Compared to prior works, DEFENDIFY is the first framework to incorporate an entropy tester to differentiate between obfuscated and non-obfuscated binaries. This allows the framework to selectively use one of two deep learning models depending on whether obfuscation is detected. Other works such as Conti et al. (2022) attempt to detect obfuscation in Android applications using deep learning, but do not provide a complete framework solution.

The DL architectures explored in DEFENDIFY leverage transfer learning to avoid expensive training procedures and enable high accuracy even with limited training data. The ResNet18 architecture achieved the best results, detecting non-obfuscated and obfuscated malware with F1 scores of 99.34% and 97.5%, respectively. This compares favorably with related work such as Ravi et al. (2022) and Conti et al. (2022), reporting malware detection accuracy between 95–98%

A key advantage of DEFENDIFY is the ability to detect malware obfuscated with real-world encoders such as XOR and Shikata Ga Nai that are commonly used by attackers (Burita and Le 2021; Webroot 2020).

[11] https://pypi.org/project/dash/.

[12] https://huggingface.co/spaces/esab/malware_detection.

Most prior work does not evaluate on such realistically obfuscated data. Testing on these encoders makes the framework more robust to new obfuscation techniques employed by malware creators.

Transfer Learning (TL) offers significant advantages over training deep learning models from scratch for obfuscated malware detection. Recent research using non-TL approaches, such as depth-wise CNN with attention mechanisms (Han et al. 2024) and dynamic analysis CNN (Mercaldo et al. 2023), have demonstrated the viability of traditional deep learning methods. However, our TL-based approach achieves superior results in terms of:

- Reduced malware dataset: DEFENDIFY employs a relative small dataset of 31,449 images (goodware and malware), due to it is inheriting the data features of the ImageNet-1K dataset, which is composed of more than 1.2 millions of images. To train a CNN without TL requires millions of samples to achieve comparable results in terms of accuracy as the ones obtained by DEFENDIFY (99.34% for non-obfuscated samples and 97.42% for obfuscated samples).
- Reduced training times: Using Transfer Learning may require a different training time depending on the number of epochs and the capabilities of the computational infrastructure. In the case of DEFENDIFY, thanks to TL it was possible to train a CNN in less than 5 min with 4 epochs using a Nvidia Saturn Cloud T4-4XLarge instance with 64 GB RAM, 16 CPU, 1 GPU and 1 TiB disk space. This time metric could not be achieved if TL is not used, as the training could last weeks.
- Less computational resources: Training a CNN with TL requires instances with significantly less computational features than training without TL, e.g. mid-range GPUs vs multi-GPUs, Quad-core processors (Intel i5/i7, Ryzen 5/7) vs High-performance CPUs (Intel i9, Ryzen 9, Threadripper), 16 Gb vs 32 Gb of RAM, 512 Gb vs 1-2 Tb of storage.

The effectiveness of TL can be attributed to leveraging pre-trained weights from large-scale image datasets, which provide robust feature extraction capabilities. This transfer of knowledge is particularly valuable in malware detection, where acquiring large-scale labeled datasets is challenging. Our results demonstrate that TL can effectively bridge these practical limitations while maintaining or improving detection performance compared to traditional approaches, offering a compelling balance of performance, efficiency, and practicality for obfuscated malware detection.

The computational analysis demonstrates that DEFENDIFY can scale efficiently thanks to the lightweight ResNet18 model. The throughput exceeds 500 samples per second with under 500 MB memory consumption on a GPU instance. This efficiency enables deployment to production environments for practical malware detection.

At last we may also consider the application of DEFENDIFY in the context of a smart city, where a malware campaign is addressed to some critical smart city service, like a traffic management service, impacting the dynamics of a smart city, due it would decrease the productivity of business that depends on transport like logistics and cargo transport and would increase the pollution, just to mention some effects. Considering a more individual sphere, we could also evaluate the impact that would be generated by a malware campaign addressed to impact a mobile transport application used by smart citizens to consult the schedule of buses or trains or review best routes to reach a destination. As generally, the end user device is the one more vulnerable between the components of a system, probably the malware campaign would try to exploit a vulnerability associated with such device to affect directly the transport application, e.g. through an abuse of the device administration API (T1616) (Mitre 2025a), or to affect services consumed by the transport application, e.g. altering the domain dynamic resolution (T1637) (Mitre 2025b). In this case, DEFENDIFY may detain the implantation of the malware or some of their components through a threat analysis performed directly in the user device or in a centralized service.

Limitations

Our results show the effectiveness of using CNN and TL for malware detection, however, these also bring some limitations in our proposal. First, a limitation coming from the use of CNN architectures is that its hierarchical feature extraction may not perfectly align with the obfuscated malware's structural patterns. Second, TL uses pre-trained weights from natural image datasets like ImageNet, which may not be optimal to capture the unique characteristics of binary-derived images. These two limitations may impact our proposal ability to detect certain obfuscation patterns.

These limitations are addressed by the properties of each one of the CNN architectures tested in the DEFENDIFY design: ResNet and EfficientNet. On the one hand, ResNet architecture uses residual connections to mainly address the vanishing gradient problem; however, such connections also preserve the fine-grained patterns existing in binaries, such as the ones that DEFENDIFY analyzes. On the other hand, EfficientNet uses Compound Scaling mainly to scale a network architecture in

a balanced way in terms of depth, width, and resolution. In that way, Compound Scaling allows a balance between accuracy and computational cost, but it also contributes in the ability of the CNN to extract features at different spacial resolutions, i.e. multi-scale feature extraction, which is crucial to recognize unique malware characteristics and patterns independent of the imagebinary size. Despite these inherent limitations, our experiments depict a robust performance across obfuscated and non-obfuscated samples, ResNet obtains the highest Accuracy (99.34% for Non obfuscated samples and 97.42% for Obfuscated samples), and EfficientNet obtains the highest Precision (99.53% for Non obfuscated samples and 95.72% for Obfuscated samples). Both architectures successfully learn hierarchical binary pattern representations while maintaining efficiency through residual connections (ResNet) and compound scaling (EfficientNet).

Another limitation is the framework's performance against zero-day malware attacks, which by definition use novel techniques not seen in the training data. Thus, the current implementation may struggle to detect completely new obfuscation methods or malware families. Additionally, the framework's adaptability to different computing environments, such as resource-constrained IoT devices or high-performance servers, requires further investigation. To address these limitations, as future work we plan to explore continual learning approaches to adapt the models to new threats over time. Techniques like few-shot learning could also be investigated to improve detection of novel malware families with limited samples. Regarding adaptability, developing lightweight versions of the framework optimized for different hardware targets would enhance its practical applicability across diverse computing environments.

In summary, DEFENDIFY pushes the state-of-the-art in obfuscated malware detection through a robust framework combining an entropy tester, deep transfer learning models, and evaluation on realistically encoded malware data. Key advantages include high detection accuracy, flexibility to different obfuscation techniques, and efficient computation for practical purposes. While challenges remain, particularly for zero-day attacks and diverse computing environments, DEFENDIFY provides a strong foundation for future research in adaptive and robust malware detection systems.

Conclusions and future work

Malware continues to pose a critical threat to the security of computers, mobile phones and IoT devices (Díaz-López et al. 2018). These malicious programs can spy on users, steal sensitive data and even encrypt files on infected devices (Sánchez Venegas et al. 2019). Moreover, in response to advanced protective measures, malware increasingly employs obfuscation techniques that hinder detection, making it challenging to distinguish malicious software from benign programs. In this context, traditional Machine Learning (ML) and Deep Learning (DL) techniques have proven ineffective and the research community must shift focus towards new methods capable of detecting obfuscated malware.

In this paper, we introduced DEFENDIFY, a framework that demonstrates the potential of using computer vision and deep learning techniques for obfuscated malware detection. By converting malware binaries into images and applying transfer learning, DEFENDIFY achieves high accuracy in detecting obfuscated malware samples. This indicates that visual malware analysis combined with deep learning constitutes a promising approach for overcoming the limitations of conventional malware detection methods in handling obfuscated malware.

DEFENDIFY consists of three main components, namely: Dataset Creation, Binary Obfuscation, and Model Generation. Specifically, the Dataset Creation module collects both malware and goodware samples. The the Binary Obfuscation module then selects specfic encoding mehods to obfuscate the malware. After the encoders are chosen, the module obfuscates the samples, transforms them into greyscale images and creates three training sets. The first one contains Shikata Ga Nai obfuscated malware and goodware, the second contains XOR obfuscated samples, and the third includes non-obfuscated samples. Finally, the Model Generation module incorporates an entropy tester to determine whether samples are obfuscated. Leveraging this framework, two DL models powered with transfer learning and previously trained, are employed. The first model detects malware from non-obfuscated samples, while the second model is used for obfuscated samples. To effectively apply the TL strategy, DEFENDIFY utilizes pre-trained weights for the feature extractors layers and trains only the last classification layers. Finally, performance is evaluated in DEFENDIFY using Accuracy, Precision, Recall, and F1-score metrics.

Additionally, we validated the performance and computational efficiency of our approach using real-world malware and goodware samples. To this end, the Model Generation module was configured to test four DL architectures: ResNet18, ResNet 34, EfficientNetB3, and EfficientNetV2S. In terms of evaluation metrics, we found that ResNet18 outperformed the other architectures for both obfuscated and non-obfuscated samples, achieving F1-scores of 99.34% and 97.5%, respectively. Regarding computational consumption, RestNet18 required the least resources, followed by RestNet34, EfficientNetB3, and finally EfficientNetV2S.

As future work, we plan to explore interpretability techniques to identify the specific location of malicious code within the malware samples. These techniques will enhance trust in the model by not only classifying samples as malware or goodware, but also providing explanations for why a sample is categorized in a particular way.

Additionally, we aim to address certain limitations of the framework by improving the diversity of the datasets used, ensuring broader coverage of different obfuscation techniques and malware families. This will enhance the model's adaptability to emerging threats. Furthermore, we plan to investigate methods to strengthen the framework's effectiveness against ever-evolving obfuscation techniques, ensuring its applicability to diverse malware types. An additional limitation of our current work is that it focuses solely on x86 binaries. Exploring cross-architecture compatibility, such as assessing ARM binaries on x86 platforms, is an important direction for future research.

We also plan as future works to develop experiments to perform a comparison between the method (Binary-to-Pixel Mapping) to convert binary files into images employed currently by DEFENDIFY, with other methods like: conversion of instructions (opcodes) to images, visualization based on entropy, generation of colored images using a RGB mapping, conversion of individual Portable Executable (PE) sections, among others. Besides, it could prove useful to implement algorithms that do not rely on binary to image transformation processes, using traditional neural network architectures that can handle unprocessed binary information, in order to compare the results obtained when images are not used. In this way, processing pipelines similar to DEFENDIFY, but that do not use TL or CNNs, may be developed as a future work.

Acknowledgements
We acknowledge all colleagues from School of Engineering, Science and Technology at Universidad del Rosario (Colombia) and Faculty of Computer Science at Universidad de Murcia (Spain) who provided feedback to improve the quality of this paper.

Author contributions
Conceptualization: Rodrigo Castillo Camargo, Santiago Alférez, Daniel Díaz-López; Methodology and validation: Angel Luis Perales Gómez, Pantaleone Nespoli; Data curation, formal analysis and investigation: Juan Murcia Nieto, Rodrigo Castillo Camargo, Nicolás Rojas; Writing - original draft preparation: Juan Murcia Nieto, Rodrigo Castillo Camargo, Nicolás Rojas, Pantaleone Nespoli; Writing - review and editing: Daniel Díaz-López, Félix Gómez Mármol, Umit Karabiyik; Funding acquisition: Daniel Díaz-López, Angel Luis Perales Gómez, Pantaleone Nespoli, Santiago Alférez; Supervision: Santiago Alférez, Daniel Díaz-López.

Funding
This work has been partially funded by the School of Engineering, Science and Technology at Universidad del Rosario (Colombia) through a "Beca de Estancia de Docencia e Investigación - EDI 2022-1", by MCIN/AEI/10.13039/501100011033, NextGenerationEU/PRTR, UE, under Grant TED 2021-129300B-I00, by MCIN / AEI / 10.13039 / 501100011033 / FEDER, UE, under Grant PID2021 - 122466OB - I00, the strategic project DEFENDER from the Spanish National Institute of Cybersecurity (INCIBE), by the Recovery, Transformation and Resilience Plan, Next Generation EU, and by the Spanish Ministry of Universities linked to the European Union through the NextGenerationEU program, under Margarita Salas postdoctoral fellowship (172/MSJD/22). This work has also been partially funded by the Nvidia Academic Grant Program through GPU instances provided by Saturn Cloud.

Availability of data and materials
Instructions on how to obtain datasets used in the experiments are available at the project repository at https://github.com/Nirogu/ObfuscatedMalwareDetection.

Code availability
Code developed to execute the experiments presented in this paper is available at the project repository at https://github.com/Nirogu/ObfuscatedMalwareDetection.

Declarations

Competing interests
The authors declare no competing interests.

Received: 16 May 2024 Accepted: 17 March 2025
Published online: 29 April 2025

References

Allix K, Bissyandé TF, Klein J et al (2015) Are your training datasets yet relevant? In: Piessens F, Caballero J, Bielova N (eds) Engineering secure software and systems. Springer, Cham, pp 51–67

Balram N, Hsieh G, McFall C (2019) Static malware analysis using machine learning algorithms on APT1 dataset with string and PE header features. In: 2019 International conference on computational science and computational intelligence (CSCI). IEEE, pp 90–95

Bat-Erdene M, Park H, Li H et al (2017) Entropy analysis to classify unknown packing algorithms for malware detection. Int J Inf Secur 16(3):227–248. https://doi.org/10.1007/s10207-016-0330-4

Ben-Bassat M, Klove KL, Weil MH (1980) Sensitivity analysis in Bayesian classification models: multiplicative deviations. IEEE Trans Pattern Anal Mach Intell PAMI 2(3):261–266. https://doi.org/10.1109/TPAMI.1980.4767015

Bensaoud A, Abudawaood N, Kalita J (2020) Classifying malware images with convolutional neural network models. Int J Netw Secur 22(6):1022–1031

Bergstra J, Bardenet R, Bengio Y, et al (2011) Algorithms for hyper-parameter optimization. In: Proceedings of the 24th international conference on neural information processing systems. Curran Associates Inc., Granada, Spain, NIPS'11, pp 2546–2554

Breiman L (2001) Random forests. Mach Learn 45:5–32. https://doi.org/10.1023/A:1010950718922

Burita L, Le DT (2021) Cyber security and apt groups. In: 2021 Communication and information technologies (KIT), pp 1–7. https://doi.org/10.1109/KIT52904.2021.9583744

Ceschin F, Botacin M, Lüders G, et al (2021) No need to teach new tricks to old malware: winning an evasion challenge with xor-based adversarial samples. In: Reversing and offensive-oriented trends symposium. Association for Computing Machinery, Vienna, ROOTS'20, pp 13–22. https://doi.org/10.1145/3433667.3433669

Conti M, Vinod P, Vitella A (2022) Obfuscation detection in android applications using deep learning. J Inf Secur Appl 70:103311

Cover T, Hart P (1967) Nearest neighbor pattern classification. IEEE Trans Inf Theory 13(1):21–27. https://doi.org/10.1109/TIT.1967.1053964

Cox D (1989) Analysis of binary data, 2nd edn. Monographs on statistics and applied probability. Chapman and Hall, London

Cyberfish (2025) Zero phishing protection for MSPs and SMB | cofense protect. https://cofense.com/. Accessed 15 Mar 2025

Díaz-López D, Blanco Uribe M, Santiago Cely C et al (2018) Developing secure IoT services: a security-oriented review of IoT platforms. Symmetry. https://doi.org/10.3390/sym10120669

El Merabet H, Hajraoui A (2019) A survey of malware detection techniques based on machine learning. Int J Adv Comput Sci Appl 10(1):366–373
Farley R, Wang X (2014) Codext: automatic extraction of obfuscated attack code from memory dump. In: Chow SSM, Camenisch J, Hui LCK et al (eds) Inf Secur. Springer, Cham, pp 502–514
Fisher RA (1936) The use of multiple measurements in taxonomic problems. Ann Eugen 7(2):179–188. https://doi.org/10.1111/j.1469-1809.1936.tb02137.x
Géron A (2019) Hands-on machine learning with scikit-learn, keras, and tensorflow. O'Reilly Media Inc, Newton
Glaroudis D, Iossifides A, Chatzimisios P (2020) Survey, comparison and research challenges of IoT application protocols for smart farming. Comput Netw 168:107037
Graham D (2021) Ethical hacking: a hands-on introduction to breaking in. No Starch Press, San Francisco
Guo Y, Fan W (2019) Feature collection and selection in malware classification. In: Proceedings of the 2019 international conference on artificial intelligence and advanced manufacturing, pp 1–5
Hakon P, Fareed Y, Sindre G (2020) The ransomware-as-a-service economy within the darknet. Comput Secur 92:101762. https://doi.org/10.1016/j.cose.2020.101762
Han S, Yun H, Park Y (2024) Deep learning for cybersecurity classification: utilizing depth-wise CNN and attention mechanism on VM-obfuscated data. Electronics 13(17):3393
Hussain A, Asif M, Ahmad MB, et al (2022) Malware detection using machine learning algorithms for windows platform. In: Proceedings of international conference on information technology and applications: ICITA 2021. Springer, pp 619–632
Iaksch J, Fernandes E, Borsato M (2021) Digitalization and big data in smart farming—a review. J Manag Anal 8(2):333–349
Jiang Y, Li R, Tang J et al (2020) Aomdroid: detecting obfuscation variants of android malware using transfer learning. In: Park N, Sun K, Foresti S et al (eds) Security and privacy in communication networks. Springer, Cham, pp 242–253
Joshi S, Upadhyay H, Lagos L, et al (2018) Machine learning approach for malware detection using random forest classifier on process list data structure. In: Proceedings of the 2nd international conference on information system and data mining, pp 98–102
Kakisim AG, Gulmez S, Sogukpinar I (2022) Sequential opcode embedding-based malware detection method. Comput Electr Eng 98:107703
Ke G, Meng Q, Finley T, et al (2017) Lightgbm: a highly efficient gradient boosting decision tree. In: Advances in neural information processing systems, vol 30. Curran Associates, Inc., Long Beach
Kim JY, Bu SJ, Cho SB (2018) Zero-day malware detection using transferred generative adversarial networks based on deep autoencoders. Inf Sci 460–461:83–102. https://doi.org/10.1016/j.ins.2018.04.092
Lee H, Kim S, Baek D et al (2023) Robust IoT malware detection and classification using opcode category features on machine learning. IEEE Access 11:18855–18867
Li M, Han D, Li D et al (2022) MFVT: an anomaly traffic detection method merging feature fusion network and vision transformer architecture. EURASIP J Wirel Commun Netw 1:39
Lyda R, Hamrock J (2007) Using entropy analysis to find encrypted and packed malware. IEEE Secur Priv 5(2):40–45. https://doi.org/10.1109/MSP.2007.48
Ma C (2021) Smart city and cyber-security; technologies used, leading challenges and future recommendations. Energy Rep 7:7999–8012
Marastoni N, Giacobazzi R, Dalla Preda M (2021) Data augmentation and transfer learning to classify malware images in a deep learning context. J Comput Virol Hack Techn 17(4):279–297. https://doi.org/10.1007/s11416-021-00381-3
Martínez Martínez I, Florián Quitián A, Díaz-López D et al (2021) Malseirs: forecasting malware spread based on compartmental models in epidemiology. Complexity 2021:5415724. https://doi.org/10.1155/2021/5415724
Mercaldo F, Ciaramella G, Santone A, et al (2023) Obfuscated mobile malware detection by means of dynamic analysis and explainable deep learning. In: Proceedings of the 18th international conference on availability, reliability and security, pp 1–10
Mimecast (2025) Cybergraph datasheet–mimecast. https://www.mimecast.com/resources/datasheets/cybergraph/. Accessed 15 Mar 2025
Mitre (2025a) Abuse elevation control mechanism: device administrator permissions. https://attack.mitre.org/techniques/T1626/001/. Accessed 15 Mar 2025
Mitre (2025b) Dynamic resolution. https://attack.mitre.org/techniques/T1637/. Accessed 15 Mar 2025
Nespoli P, Useche Peláez D, Díaz-López D et al (2019) Cosmos: collaborative, seamless and adaptive sentinel for the internet of things. Sensors. https://doi.org/10.3390/s19071492
Pelaez DU, Díaz-López D, Nespoli P, et al (2018) Tris: a three-rings IoT sentinel to protect against cyber-threats. In: 2018 Fifth international conference on internet of things: systems, management and security, pp 123–130. https://doi.org/10.1109/IoTSMS.2018.8554432
Pereira-Kohatsu J, Quijano-Sánchez L, Liberatore F et al (2019) Detecting and monitoring hate speech in twitter. Sensors 19(21):4654
Platt J (2000) Probabilistic outputs for support vector machines and comparisons to regularized likelihood methods. Adv Large Margin Classif 10:61–74
Ravi V, Alazab M, Selvaganapathy S et al (2022) A multi-view attention-based deep learning framework for malware detection in smart healthcare systems. Comput Commun 195:73–81
Rehman A, Saba T, Khan MZ et al (2022) Internet-of-things-based suspicious activity recognition using multimodalities of computer vision for smart city security. Secur Commun Netw 2022:8383461
Rezaei T, Manavi F, Hamzeh A (2021) A PE header-based method for malware detection using clustering and deep embedding techniques. J Inf Secur Appl 60:102876
Ribani R, Marengoni M (2019) A survey of transfer learning for convolutional neural networks. In: 2019 32nd SIBGRAPI conference on graphics, patterns and images tutorials (SIBGRAPI-T). IEEE, pp 47–57
Ristvej J, Lacinák M, Ondrejka R (2020) On smart city and safe city concepts. Mob Netw Appl 25:836–845
Sahin M, Bahtiyar S (2020) A survey on malware detection with deep learning. In: 13th international conference on security of information and networks, pp 1–6
Sánchez Venegas C, Aguado Bedoya C, Díaz-López D et al (2019) Using reverse engineering to handle malware. Ing Solidar 15(2):1–26. https://doi.org/10.16925/2357-6014.2019.02.02
Shaukat K, Luo S, Varadharajan V (2023) A novel deep learning-based approach for malware detection. Eng Appl Artif Intell 122:106030
Smith LN (2017) Cyclical learning rates for training neural networks. arXiv:1506.01186
Song L, Hu X, Zhang G et al (2022) Networking systems of AI: on the convergence of computing and communications. IEEE Internet Things J 9(20):20352–20381. https://doi.org/10.1109/JIOT.2022.3172270
Talukder S, Talukder Z (2020) A survey on malware detection and analysis tools. Int J Netw Secur Appl 12(2): 21
Tan M, Le Q (2019) EfficientNet: rethinking model scaling for convolutional neural networks. In: Chaudhuri K, Salakhutdinov R (eds) Proceedings of the 36th international conference on machine learning, proceedings of machine learning research, vol 97. PMLR, Long Beach, California, USA, pp 6105–6114. https://proceedings.mlr.press/v97/tan19a.html
Tan M, Le QV (2020) Efficientnet: rethinking model scaling for convolutional neural networks. arXiv:1905.11946
Tan M, Le QV (2021) Efficientnetv2: smaller models and faster training. arXiv:2104.00298
Toffalini F, Ochoa M, Sun J, et al (2019) Careful-packing: a practical and scalable anti-tampering software protection enforced by trusted computing. In: Proceedings of the ninth ACM conference on data and application security and privacy. Association for Computing Machinery, Richardson, Texas, USA, CODASPY '19, pp 231–242. https://doi.org/10.1145/3292006.3300029
Useche-Pelaez DE, Sepulveda-Alzate D, Díaz-López D et al (2018) Building malware classificators usable by state security agencies. Iteckne 15:107–121
Webroot (2020) Webroot threat report. Tech. rep., Webroot. https://mypage.webroot.com/rs/557-FSI-195/images/2020%20Webroot%20Threat%20Report_US_FINAL.pdf
Weiss K, Khoshgoftaar TM, Wang D (2016) A survey of transfer learning. J. Big Data 3(1):1–40

Xi B (2020) Adversarial machine learning for cybersecurity and computer vision: current developments and challenges. Wiley Interdiscip Rev: Comput Stat 12(5):e1511
Xue H, Sun S, Venkataramani G et al (2019) Machine learning-based analysis of program binaries: a comprehensive study. IEEE Access 7:65889–65912

Publisher's Note