





FORENSICS: Deciphering and Detecting Malware Through Variable-Length Instruction Sequences

César Augusto Borges de Andrade   [University of Brasília | cesar.andrade@aluno.unb.br]


Geraldo Pereira Rocha Filho  [State University of Southwest Bahia | geraldo.rocha@uesb.edu.br]

Rodolfo I. Meneguette  [University of São Paulo | meneguette@icmc.usp.br]

Ricardo Sant'Ana  [Military Institute of Engineering | santana.ricardo@eb.mil.br]

Julio Cesar Duarte  [Military Institute of Engineering | duarte@ime.eb.br]

André Luiz Marques Serrano  [University of Brasília | andrelms@unb.br]

Clóvis Neumann  [University of Brasília | clovisneumann@unb.br]

Vinícius P. Gonçalves  [University of Brasília | vpvginicius@unb.br]

 Electrical Engineering Dept., University of Brasília, Campus Universitário Darcy Ribeiro, Brasília, DF, 70910-900, Brazil.

Received: 05 October 2024 • **Accepted:** 23 July 2025 • **Published:** 14 October 2025

Abstract The increasing complexity of contemporary malware, driven by the use of advanced evasion and obfuscation techniques, combined with the rapid growth in the number of new variants emerging continuously, undermines the effectiveness of traditional signature-based detection mechanisms. In response to this scenario, this work proposes FORENSICS (Framework fOR malwaRe dEtectioN baSed on InstrUction Sequences), an innovative deep learning-based framework that employs variable-length instruction sequences to detect malware efficiently and accurately. By integrating Natural Language Processing (NLP) techniques with Long Short-Term Memory (LSTM) neural networks, FORENSICS analyzes opcode sequences extracted from real-world malware and benign software artifacts. The framework introduces optimized methods for opcode extraction and representation, significantly reducing computational overhead while preserving detection performance. FORENSICS achieved 99.91% accuracy, 99.99% precision, and detection times ranging from 8 to 17 milliseconds, outperforming several state-of-the-art approaches across multiple metrics. Additionally, the framework demonstrated robustness in identifying zero-day malware samples, confirming its effectiveness in real-world cybersecurity scenarios. A new balanced dataset comprising over 40,000 labeled samples was created and made publicly available, facilitating reproducibility and encouraging further research. These results position FORENSICS as a robust, scalable, and highly effective solution for malware detection in modern threat landscapes.

Keywords: Malware Detection, Opcode, Recurrent Neural Networks (RNNs), Long-Short-Term Memory (LSTM), Natural Language Processing (NLP), Cybersecurity.

1 Introduction

Identifying malware is a critical and complex task in the domain of cybersecurity. Due to the constant advancement of the techniques used to create malware, conventional protection systems often fail to detect and eliminate emerging threats effectively. Advanced malware has the ability to hide, adjust, and evolve to avoid detection by exploiting software and operating system vulnerabilities [Palma Salas *et al.*, 2023]. Furthermore, the increasing variety and amount of malware, fueled by the accessibility of tools to create harmful code, further complicates the detection process [Aslan and Samet, 2020].

Recently, the detection of new malware has shown a substantial increase, as documented in 2023 security reports. According to SonicWall [2023], more than 465,000 previously unknown malware variants were identified, representing a 5% increase over the previous year. The average daily detection of new variants reached 1,279, highlighting the growing sophistication of attacks. Similarly, Kaspersky's report [Kaspersky, 2023] revealed that detection systems discovered

an average of 411,000 malicious files daily in 2023, an increase of nearly 3% over the previous year, with a notable 53% increase in attacks involving malicious Microsoft Office documents. Furthermore, malware infections targeting Internet of Things (IoT) devices increased by 37% in the first half of 2023. These data indicate a shift in cybercriminals' strategies, reflecting their ability to exploit new vulnerabilities and adopt more stealthy and sophisticated techniques. This scenario highlights the need for continuous monitoring and updating of cyber defenses to mitigate emerging threats.

Historically, antivirus companies have relied on malware signatures as the primary method of identifying and addressing cyber threats. Malware signatures are distinct markers obtained from certain attributes of malicious code, allowing antivirus software to effectively identify and avoid known malware [Abusitta *et al.*, 2021]. However, this method is effective only against recognized and previously examined malicious software, exposing systems to potential threats from new variations or polymorphic malware that can modify its attributes to evade signature identification [Botacin *et al.*, 2022]. Moreover, the process of generating and disseminating new

signatures can be slow, resulting in an exposure window for emerging malware [Ferdous *et al.*, 2023]. In light of this situation, there is a need for more sophisticated analysis methods that can decipher the patterns and characteristics of malware [Palma Salas *et al.*, 2023].

With the progress of deep learning techniques, Recurrent Neural Networks (RNNs) have proven to be powerful tools for deciphering complex temporal patterns in large sequential datasets [Xiao and Zhou, 2020]. An RNN is a type of neural network characterized by its ability to maintain and process sequential information over time, that is, mathematical operations applied to data sequences, with the main objective of learning temporal dependencies and patterns in sequential data [Abbaspour *et al.*, 2020]. As an RNN processes a sequence, it generates a set of hidden states indicating where and when the sought feature was detected [Shiri *et al.*, 2023].

The growing adoption of deep learning techniques has opened new possibilities for malware detection, particularly through the use of RNNs, which have proven effective in modeling instruction sequences extracted from executable code [Catal *et al.*, 2022; Gaber *et al.*, 2024; Tayyab *et al.*, 2022]. These models are able to capture temporal dependencies and structural patterns that emerge over time, making them suitable for analyzing sequential representations of malicious behavior.

Recent research efforts have explored a variety of strategies for representing Portable Executable (PE) files in the context of deep learning. Among the most frequently adopted methods are the following:

- Image-based representations derived from raw binary data [Jannat Mim *et al.*, 2024; Hebish and Awni, 2024; El ghabri *et al.*, 2024; Omar, 2022; Aslan and Yilmaz, 2021];
- Sequences of API calls extracted during program execution or emulation [Syeda and Asghar, 2024; Aggarwal and Di Troia, 2024; Owoh *et al.*, 2024; Li and Zheng, 2021; Catak and Yazi, 2019]; and
- Instruction-level abstractions such as mnemonic tokens or opcode sequences [de Andrade *et al.*, 2025; Mehta *et al.*, 2024; Kale *et al.*, 2023; Zhao *et al.*, 2021; Zhang *et al.*, 2019; Lu, 2019].

This work adopts an opcode-based representation due to several technical and practical advantages. Opcodes offer a direct view of the underlying processor instructions, allowing the identification of behavioral patterns independently of high-level programming constructs. Compared to source code or raw binaries, opcodes provide a balanced level of abstraction that remains robust against syntactic obfuscation techniques. In addition, working with short and standardized opcode sequences enables efficient training of deep learning models, reducing computational overhead. From an analytical perspective, opcode representations also facilitate the discovery of structural similarities between malware samples and enhance the interpretability of the features learned by the model. To fully exploit these characteristics, we employ Long Short-Term Memory (LSTM) networks, a type of RNN designed to effectively model long-range dependencies within sequential data, an important aspect when learning from opcode-based input.

This work goes further and proposes FORENSICS, a framework that employs variable-length instruction sequences to detect malware more efficiently and accurately, focusing on a large number of malware and goodware. To this end, FORENSICS applies Natural Language Processing (NLP) techniques and LSTM Neural Networks, specializing in the predictive analysis of opcode instruction sequences. This approach not only improves the accuracy of malware detection but also provides greater effectiveness regarding operating time. The test results showed that with just 360 opcodes extracted from the malware, it is possible to achieve an accuracy rate of 99.91% in a detection time ranging from 8 to 17 milliseconds to complete the task.

The main contributions of this research include:

1. the development of an effective technique for the extraction of opcodes from PE¹;
2. the implementation of an approach to the processing of data extracted from the artifacts, aimed at eliminating redundancies;
3. the representation of opcodes using vectors of minimal possible dimensions to enhance computational efficiency;
4. the implementation of a method that enables the rapid and accurate classification of *malware* through the analysis of specific fragments of the code; and
5. the creation and availability of a new dataset² for future research in the area.

The remainder of this article is structured as follows. In Section 2, we provide the background necessary to understand key concepts. Section 3 reviews the related work, highlighting the research gap that this study investigates. In Section 4, we detail the modeling of the FORENSICS framework, followed by Section 5, where we validate FORENSICS through a comparison with existing approaches. Finally, Section 6 concludes the paper and outlines the potential directions for future research.

2 Background

In this section, we present the foundational concepts necessary for this study, including an overview of RNNs and LSTM networks, as well as key principles of Static and Dynamic Analysis in the context of malware detection. We also discuss the inherent limitations of Static Analysis, highlighting challenges that motivate the development of more robust detection approaches.

2.1 Recurrent Neural Networks

Recurrent Neural Networks (RNNs) are a class of artificial neural networks designed to process sequential data, making them particularly effective for tasks that involve ordered or temporal dependencies [Rumelhart *et al.*, 1986]. Unlike traditional feedforward neural networks, where inputs are treated as independent, RNNs have cyclic connections that

¹format Portable Executable.

²<https://github.com/CABorges72/SMARTNESS/tree/main>

allow information from previous states to influence the interpretation of subsequent states. This property makes RNNs ideal for a variety of applications, including natural language processing, speech recognition, machine translation, and time series analysis.

The functioning of RNNs is based on maintaining a hidden state that is updated at each time step. The hidden state h_t at time t is a function of the current input x_t and the previous hidden state h_{t-1} . This mechanism allows the network to have a form of “memory” of past data, accumulating useful information for the task at hand. However, one of the main challenges of traditional RNNs is the problem of vanishing gradients, which hinders the effective training of networks over long sequences. To mitigate these issues, variants such as long-short-term memory (LSTM) networks and Gated Recurrent Units (GRU) were developed. These variants introduce gating mechanisms that regulate the flow of information, allowing networks to capture long-term dependencies more effectively.

2.1.1 Basic RNN Formulas

For a basic RNN, the hidden state h_t is calculated based on the current input x_t and the previous hidden state h_{t-1} :

1. Hidden State Calculation:

$$h_t = \sigma(W_{hx}x_t + W_{hh}h_{t-1} + b_h) \quad (1)$$

where:

- h_t is the hidden state at time t
- x_t is the input at time t
- W_{hx} is the weight matrix between the input and the hidden state
- W_{hh} is the recurrent weight matrix (between hidden states)
- b_h is the bias vector
- σ is the activation function (such as \tanh or ReLU)

2. Output Calculation:

$$y_t = \phi(W_{hy}h_t + b_y) \quad (2)$$

where:

- y_t is the output at time t
- W_{hy} is the weight matrix between the hidden state and the output
- b_y is the bias vector
- ϕ is the activation function for the output (such as softmax for classification)

2.2 Long Short-Term Memory Networks

LSTM networks are an advanced class of RNNs developed to address the limitations of traditional RNNs, such as the vanishing gradient problem. Proposed by Hochreiter and Schmidhuber [1997], LSTMs are designed to model and capture long-term dependencies in data sequences. This makes them particularly suitable for tasks involving complex sequential data, such as machine translation, language modeling, speech recognition, and time series forecasting.

The functioning of LSTMs is based on a memory cell and three key gates: the input gate, the forget gate, and the output gate. The input gate controls which new information is added to the memory cell, while the forget gate decides which old information should be discarded. The output gate determines which parts of the memory cell are used to compute the output. These gates are activated by the sigmoid and tanh functions, which regulate the flow of information within the cell. This mechanism allows LSTMs to maintain and utilize relevant information over long sequences, mitigating the vanishing-gradient problem.

Recently, LSTMs have been widely adopted in various applications because of their ability to handle long-term dependencies. Furthermore, current research explores combining LSTMs with other neural network architectures, such as Convolutional Neural Networks (CNNs) and attention mechanisms, to further enhance their performance in complex tasks. Bidirectional LSTMs, which process information in both temporal directions, have also shown promising results, especially for the development of robust solutions for sequential modeling problems.

2.2.1 LSTM Equations

LSTMs use three gates (input, forget, and output) to control the flow of information through the memory cells. The equations are as follows:

1. Forget Gate:

$$f_t = \sigma(W_fx_t + U_fh_{t-1} + b_f) \quad (3)$$

where:

- f_t is the forget gate activation at time t
- W_f, U_f, b_f are the weights and bias associated with the forget gate

2. Input Gate:

$$i_t = \sigma(W_ix_t + U_ih_{t-1} + b_i) \quad (4)$$

where:

- i_t is the input gate activation at time t
- W_i, U_i, b_i are the weights and bias associated with the input gate

3. New Candidate Information:

$$\tilde{C}_t = \tanh(W_Cx_t + U_Ch_{t-1} + b_C) \quad (5)$$

where:

- \tilde{C}_t is the new candidate information for the memory cell at time t
- W_C, U_C, b_C are the weights and bias associated with the candidate information

4. Memory Cell Update:

$$C_t = f_t \odot C_{t-1} + i_t \odot \tilde{C}_t \quad (6)$$

where:

- C_t is the memory cell state at time t
- \odot denotes element-wise multiplication

5. Output Gate:

$$o_t = \sigma(W_o x_t + U_o h_{t-1} + b_o) \quad (7)$$

where:

- o_t is the output gate activation at time t
- W_o, U_o, b_o are the weights and bias associated with the output gate

6. Hidden State Calculation:

$$h_t = o_t \odot \tanh(C_t) \quad (8)$$

2.3 Static and Dynamic Analysis

Malware detection techniques are traditionally classified into two main categories: static analysis and dynamic analysis. Each of these approaches offers distinct advantages and limitations depending on the nature of the malware and the operational constraints of the detection system.

Static analysis involves inspecting the malware binary without executing it. This process includes examining file headers, strings, imported functions, control flow graphs, and, particularly, opcode sequences extracted from the disassembled code [Barzev and Borissova, 2025; Miao et al., 2024]. Static analysis is generally fast, scalable, and safe as it does not require the execution of potentially harmful code. It is particularly suitable for lightweight detection systems and is often used in antivirus engines and large-scale malware scanning pipelines. However, static analysis can be hindered by obfuscation, encryption, or packing techniques that conceal the true functionality of the code. These evasion methods are commonly used in polymorphic and metamorphic malware to bypass signature-based detection systems [Gaber et al., 2024].

In contrast, dynamic analysis executes malware in a controlled environment (for example, a sandbox³ or virtual machine) to monitor its runtime behavior. This includes observing system calls, API invocations, file and network activity, and memory usage [Panda et al., 2025; Qian and Cong, 2024]. Dynamic analysis is more resilient to code obfuscation and can detect actions that are not evident through static inspection. However, it incurs higher computational costs and may not trigger malicious behavior if malware employs environment-aware techniques, such as delaying execution, detecting virtualization, or requiring user interaction [Or-Meir et al., 2019].

Hybrid approaches that combine static and dynamic techniques aim to take advantage of the strengths of both methods, improving the overall accuracy and robustness of detection [Yoo et al., 2021]. However, in scenarios that involve limited computational resources, such as embedded systems or IoT devices, static analysis remains a viable and often necessary alternative, especially when coupled with behaviorally informed models such as the one proposed in this work.

2.4 Limitations of Static Analysis with Opcode Extraction from Malware

Although static analysis based on opcode extraction is lightweight and computationally efficient, it exhibits inherent

limitations, particularly when facing malware that adopts evasion techniques such as polymorphism and metamorphism. Polymorphic malware alters its surface representation with each execution or replication, employing encryption, packing, or instruction-level obfuscation. These transformations are specifically designed to bypass signature-based detection systems, which rely on consistent code patterns [Mauri and Damiani, 2025; Mohammed et al., 2025]. Since static analysis operates without executing the binary, it becomes susceptible to superficial changes that obscure malicious intent. Furthermore, common tools such as disassemblers and decompilers may struggle to extract meaningful representations when the code is deeply obfuscated or compressed, limiting the effectiveness of static inspection.

Metamorphic malware presents an even more complex threat. Unlike polymorphic variants, it can completely rewrite its internal structure across generations while preserving functionality, thus eliminating shared code patterns between instances [Gulmez et al., 2024; Habib et al., 2024]. Techniques such as junk code injection, reordering of instructions, and control flow manipulation are frequently used to distort static features. As a result, static analysis alone may not recognize these threats due to the absence of consistent syntactic or structural traits. These challenges underscore the need to complement static inspection with behavior-oriented models, such as the sequential opcode analysis approach proposed in this work, which aim to identify underlying patterns of malicious activity beyond superficial code transformations.

3 Related Works

Malware detection is an ever-evolving research area, exploring different approaches and techniques to effectively identify and analyze cyber threats. This section will examine the key related works, highlighting their proposals, adopted methodologies, main contributions, best-achieved metrics, malware datasets used, and presented limitations.

In Baghirov [2023], the authors propose a method for detecting malware using the frequency of opcodes extracted from malicious and benign programs. The methodology involves collecting opcodes using the IDAPro tool, removing highly correlated opcodes, and training various machine learning algorithms such as Random Forest. Their main contribution is demonstrating that classification based on opcode frequency can achieve high accuracy, with Random Forest achieving 99%. The dataset includes 2,000 malware samples and 1,000 benign samples, collected from public sources, focusing on malware affecting the Windows operating system. The study's limitations include the initial imbalance of the dataset, the potential inclusion of irrelevant opcodes, difficulties in generalizing to new malware, and the need for complete hyperparameter optimization of the algorithms. Additionally, the model's robustness and accuracy may be influenced by the size and diversity of the dataset.

Hoang et al. [2023] proposes two models for malware detection using statistics and machine learning with opcode n-grams. The methodology involves creating a list of benign opcode n-grams for a statistics-based model and applying machine learning algorithms, such as Naïve Bayes, KNN,

³<https://www.vmrays.com/glossary/malware-sandbox/>

SVM, Decision Trees, and Random Forest, to build classification models. The main contribution of the work is the improvement in malware detection accuracy and the reduction in training and detection time. The accuracy achieved by the Random Forest-based model was 96.29%, while the statistics-based model reached 92.75%. The datasets used included 5,051 benign files from Windows servers and 1,670 malicious files from the Bazaar⁴ and VirusShare⁵ collections. The limitations of the work include the longer training time for machine learning models compared to the statistics-based model, as well as potential ineffectiveness against malware using advanced obfuscation and encryption techniques.

The study presented by Carlson *et al.* [2023] proposes a static analysis method to detect malware by comparing opcode distributions. The authors create distributions by aggregating the number of operations between consecutive opcode calls and use Kullback-Leibler (KL) divergence to compare these distributions with reference samples of benign and malicious code. The main contribution of the work is the introduction of opcode distributions as a robust form of malware detection. The method achieved a 90% accuracy by combining sets of malicious opcodes and clustering of root opcodes. The dataset used includes over 200,000 benign and malicious executables from Practical Security Analytics⁶. A limitation of the work is the model's sensitivity to variations in opcode distributions between different malware samples, as well as the dependence on training data that may not represent new types of malware. Additionally, the logarithmic transformation of KL divergence values did not significantly increase accuracy in all situations.

In Muppalaneni and Patgiri [2021], the authors propose a malware detection method using machine learning techniques, specifically N-gram opcode sequences. The methodology involves extracting features from executable files, generating N-gram opcode sequences, and using machine learning algorithms such as SVM, Naïve Bayes, and Decision Trees to predict whether a file is malware or benign. The main contribution of the work is the use of N-gram opcode sequences to characterize the executable's behavior, providing high accuracy in malware detection. The dataset used includes 301 malware files and 72 benign files, totaling 531 attributes, obtained from the UCI Machine Learning Repository. The reported accuracies are 98.93% for SVM, 98.39% for Decision Trees, and 98.93% for Naïve Bayes. A mentioned limitation is the larger number of malware files compared to benign files in the dataset, which may bias the results.

In the work from Mohandas *et al.* [2021], the authors propose a malware detection method based on opcode frequency, using the conversion of executable files into assembly language to extract opcode information and transform it into frequencies stored in CSV files. These files are analyzed by machine learning algorithms, including Decision Trees, Random Forest, and Naïve Bayes, with Random Forest achieving the highest accuracy. The main contribution of the work is the ability to detect metamorphic and embedded malware without the need to execute the files, thereby avoiding environmental

infections. The Random Forest model achieved an accuracy of 97.97%. The training dataset included 150 metamorphic malware files and 100 benign files, all inactive. During the feature extraction process, approximately 953 different opcodes were considered. The main limitation of the work is the simplicity in the analysis of assembly instructions, which may not capture all the nuances of more sophisticated malware. Additionally, the approach may be vulnerable to malware that uses advanced techniques to mask its opcodes.

In Parildi *et al.* [2021], an alternative method for malware detection is proposed using sequences of assembly opcodes obtained during program execution. The methodology employs NLP and deep learning techniques to extract deep behavioral features, making the method immune to code obfuscation and effective against new malware. These features are fed into various machine learning algorithms for classification. The main contribution of the work is the development of a prediction pipeline that includes data collection, preprocessing, feature extraction, and classification, adapted to detect malware using runtime opcode sequences. Experiments conducted on a more balanced dataset, composed of 25,963 samples, demonstrated that an accuracy of up to 98% can be achieved. The dataset used includes malware samples obtained from repositories like VirusShare and benign samples collected from various sources, such as native installations of Windows 7 and Windows 10. An important limitation of the work is the need for high computational power for deep learning operations, as well as the challenge of data balancing, which can impact the false positive rate.

In the study conducted by Jha *et al.* [2020], the authors propose using an RNN for efficient malware detection. The methodology includes analyzing textual feature vectors using three NLP techniques: one-hot encoding, random vectors, and vectors trained with the Word2Vec model. The main contribution of the work is demonstrating that the Word2Vec feature vector, when used with RNN, provides the best performance and stability for malware detection. Experiments were conducted with 5,000 malware samples and 5,000 benign files, resulting in an accuracy of 97.8%. The study's limitations include high computational complexity due to the need to compute the entire word embedding vector in the vocabulary during training, which can be challenging when working with large datasets and requires significant computational resources. Additionally, the repetition and occurrence of opcodes in similar contexts can result in the same semantic meaning, affecting classification accuracy, especially in large volumes of data where high repetition can lead to conflicts in the classifier.

In Jeon and Moon [2020], the authors propose a malware detection method using a convolutional recurrent neural network (CRNN) that processes opcode sequences extracted from binary files. The methodology involves extracting opcode sequences, which are compressed by a convolutional autoencoder (OCAE) and subsequently classified by a dynamic recurrent neural network (DRNN). The main contribution of the work is the formulation of a model that uses opcode sequences to detect malware without the need to execute the code. The model achieved an accuracy of 96%, with an area under the ROC curve of 0.99 and a true positive rate (TPR) of 95%. The dataset used includes 1,000 benign files and 1,000

⁴<https://bazaar.abuse.ch/>

⁵<https://virusshare.com/>

⁶<https://practicalsecurityanalytics.com/pe-malware-machine-learning-dataset/>

malware samples, covering different categories such as ransomware, spyware, adware, trojans, and backdoors, collected from VirusShare. The created dataset was not made publicly available due to associated risks. The main limitation of the work is that the opcode extraction algorithm does not consider indirect branch instructions, which can impact accuracy in detecting malware that uses advanced obfuscation techniques. The work performs malware detection using multiple opcode sequences per sample, and there are no reported experiments using only a single opcode sequence per sample.

The study by Khan *et al.* [2019] proposes using ResNet and GoogleNet models for malware detection by transforming binary files into grayscale images through an opcode-to-image conversion technique. This approach aims to improve the accuracy and speed of detection. The methodology involves converting executable files (.exe) into assembly code, which is then converted into opcode sequences and finally into images. These images are used to train and test the convolutional neural network (CNN) models of GoogleNet and ResNet. The main contribution of the work is the application of image pattern recognition techniques for malware detection, achieving an accuracy of 74.5% with GoogleNet and 88.36% with ResNet. The used dataset includes 10,868 malware samples and 3,000 benign files, with malware distributed across nine different classes. The main limitation of the study is the longer execution time required by the ResNet model despite its higher accuracy.

For a comparative analysis between this research and related works, Table 1 was created based on criteria such as: whether it operates with variable-length instruction sequences, whether it is possible to classify malware from a small part or only the complete malware, whether a new dataset is created and made available, whether execution times are reported, and sample quantities. This research, FORENSICS, stands out by using a large, exclusive dataset composed of a wide variety of active malware and goodwill. Unlike related works, this research focuses on a restricted set of opcodes, demonstrating its high efficiency and performance. Additionally, methods were developed for extracting opcodes from samples, processing data to minimize redundancies, and representing opcodes with smaller vectors, contributing to the accuracy and effectiveness of malware detection. A new dataset was also created and made available.

4 FORENSICS: Framework for malware detection based on Instruction Sequences

FORENSICS can be presented as a new framework that uses variable-length instruction sequences to efficiently and accurately classify malware, with a particular focus on active and stealthy threats that pose significant real-world risks. By employing NLP techniques, proper data processing, and LSTMs, FORENSICS specializes in the predictive analysis of opcode sequences. This strategy not only enhances malware detection capabilities but also provides valuable insights into the behavior of these digital threats, minimizing the occurrence of unnecessary duplications in the analyzed datasets.

FORENSICS, illustrated in Figure 1, is organized into two modules: (i) Learning Mechanism and (ii) Detection Mechanism.

4.1 Problem Definition

In this subsection, the problem of malware classification in a binary context will be addressed, using an LSTM-based model for the prediction of subsequent opcodes. As presented in Figure 2, the modeling process includes the conversion of opcode sequences (i.e., “push”, “call”, “jnz”, “mov”) into vectors, where identical opcodes are mapped to equivalent vectors. The LSTM model is trained with temporal sequences of $k = 5$ consecutive opcodes (lookback) to predict the $k + 1 = 6^{\text{th}}$ opcode, which constitutes the desired output. For example, given a malware that presents an opcode sequence from op1 to op6, the input to the model consists of the first five opcodes (op1 to op5), and the sixth opcode (op6) represents the desired output. The value of k can be adjusted to represent a larger or smaller opcode window, depending on the experiment conducted.

The encoding step of opcode sequences for the intended inputs and outputs is important for the effective training of the LSTM model, as illustrated in Figure 3.

This process is replicated for both malware and goodwill, resulting in the development of two distinct LSTM models. This approach aims to improve opcode prediction accuracy, allowing for a deeper exploration of the behavior of malware and goodwill. In this way, it significantly contributes to the enhancement of detection and understanding of the threats these malware represent.

4.2 Artifact Acquisition

This subsection presents how the malware and goodwill used as initial inputs for this research were collected. Initially, a broad collection of 20,371 malware samples was obtained from the VirusShare repository⁷. These samples were categorized into seven distinct malware families, aiming to cover a wide spectrum of malicious behaviors. Subsequently, 20,371 goodwill samples were collected from the PE Malware Machine Learning Dataset repository⁸. The compatibility of the artifacts with the Microsoft Windows executable format was verified using the PEFile library⁹, a technical choice that ensures the relevance of the samples within the scope of this research.

To ensure the correct labeling of the artifacts, a Python script was developed to submit all artifacts to the VirusTotal platform¹⁰. This security analysis platform allows the evaluation of files and URLs for the detection of malware, viruses, and other cyber threats.

It should be emphasized that the sample selection process was carried out to ensure diversity, reliability, and representativeness of real-world cyber threats. The inclusion of multiple

⁷<https://virusshare.com/>

⁸<https://practicalsecurityanalytics.com/pe-malware-machine-learning-dataset/>

⁹<https://github.com/erocarrera/pefile>

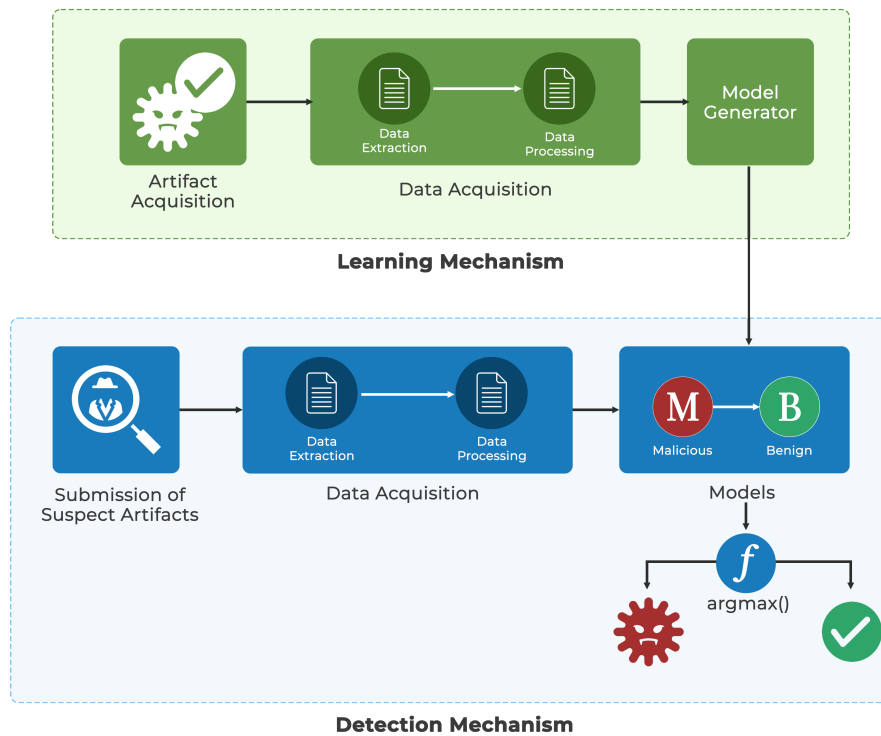
¹⁰<https://www.virustotal.com/gui/home/upload>

Table 1. Comparative Analysis of Related Works.

Work	VLIS	CPM	CND	ETR	Class	# Samples
[Baghirov, 2023]	✗	✗	✗	✗	M G	2,000 1,000
[Hoang <i>et al.</i> , 2023]	✗	✗	✗	✓	M G	1,670 5,051
[Carlson <i>et al.</i> , 2023]	✗	✗	✓	✗	M G	114,737 86,812
[Muppalaneni and Patgiri, 2021]	✗	✗	✗	✗	M G	301 72
[Mohandas <i>et al.</i> , 2021]	✗	✗	✗	✗	M G	150 100
[Parildi <i>et al.</i> , 2021]	✗	✓	✓	✗	M G	18,827 7,136
[Jha <i>et al.</i> , 2020]	✗	✓	✗	✓	M G	5,000 5,000
[Jeon and Moon, 2020]	✓	✗	✗	✗	M G	1,000 1,000
[Khan <i>et al.</i> , 2019]	✗	✗	✗	✓	M G	10,868 3,000
This	✓	✓	✓	✓	M G	20,371 20,371

¹ VLIS (Variable-Length Instruction Sequences). ² CPM (Classification from a small part of the malware). ³ CND (Creation and availability of a new dataset).

⁴ ETR (Execution time recording). ⁵ M (Malware). ⁶ G (Goodware).

**Figure 1.** Operating Scenario of FORENSICS.

actively circulating malware families, combined with the validation of all artifacts through the VirusTotal platform, which aggregates numerous antivirus engines, ensures that the analyzed samples reflect realistic, relevant, and broadly representative malware scenarios observed in current cybersecurity

contexts.



Figure 2. Example of an opcode sequence from a specific artifact.

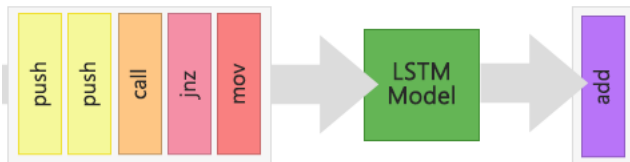


Figure 3. Each sequence of five opcodes and the output opcode are used in the LSTM training.

4.3 Data Acquisition

This subsection presents how the disassembly and extraction of opcodes were carried out from the artifacts acquired in the previous section.

4.3.1 Data Extraction

For the disassembly of the artifacts, a Python script was developed using the `pefile` and `capstone` modules¹¹ to automate this process for both the malware and goodwill bases. The program calculated the exact location of the first opcode of each artifact by using the “`AddressOfEntryPoint`” header field¹² of the PE (Portable Executable) file. This step is important for the effective extraction of data from the artifacts, and the program’s effectiveness was corroborated through comparative analyses with the outputs produced by IDA Pro¹³, a market-leading disassembler, ensuring the reliability of the extracted data.

After disassembly, the focus shifts to extracting opcode sequences from the artifacts. Unlike other approaches, this research exclusively concentrates on the opcode sequence, such as “mov”, “push”, “call”, “or”, which is extracted and stored for subsequent analyses. This methodological selection supports the analysis of malicious code behavior, relying on data extracted directly from malware and goodwill samples.

4.4 Data Processing

The first step in the data processing involved identifying and removing duplicate files. The files were organized by size, and the identified duplicates were automatically eliminated. This procedure resulted in a significant reduction in data volume, facilitating subsequent handling and analysis. The effectiveness of this step is demonstrated by the quantitative reduction of files, as presented in Table 2, column # malware (b), and Table 3, column # goodware (b). Here, the quantity of goodware is matched to the quantity of malware, i.e., 13,719, as shown in Table 3, column # goodware (c).

Next, the focus was on eliminating repeated opcode sequences among the samples, aiming to preserve only unique instances for analysis. It is worth noting that an automatic filtering process was performed, removing redundant sequences of mnemonics. This action contributed to a substantial reduction in the number of opcodes per class, as presented in Table 2, column #opcodes (b), and Table 3, column #opcodes (b).

To avoid analytical distortions caused by class imbalance, the classes were balanced. The number of opcodes in all classes was equalized to the total found in the class with the smallest volume, specifically class 3, which had 3,621,441 opcodes. This adjustment ensured a fair comparison base among the classes, as presented in Table 2, column #opcodes (c).

To further avoid analytical distortions, the number of goodware opcodes was equalized to the number of malware opcodes, specifically 25,350,087 opcodes. This adjustment also ensured a fair comparison base between the classes, as presented in Table 2, column #opcodes (c), and Table 3, column #opcodes (c).

It is worth noting that a data transformation was necessary since deep learning algorithms cannot directly process opcodes. To achieve this, the prevalent opcodes in each class analyzed were identified and cataloged. This activity is fundamental to understanding the distribution and relevance of terms within the data corpus. Using the extracted information, word clouds, Figure 4 and Figure 5, respectively, and bar charts were generated for an intuitive analysis of the frequency and importance of opcodes in each class. In addition, a specific dictionary was developed for each class (malware and goodware), where terms are defined and organized according to their relevance and frequency. This dictionary is important because it serves as the basis for subsequent data processing steps.

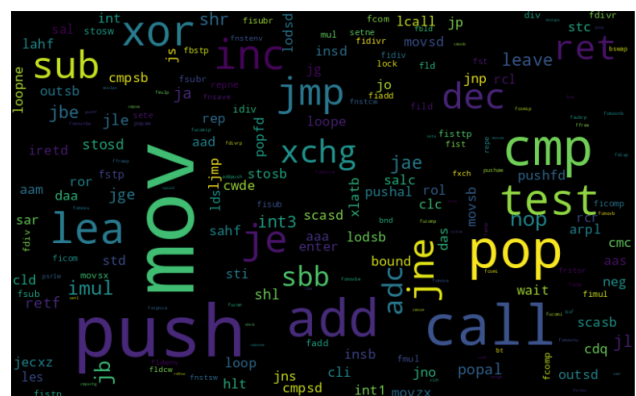


Figure 4. Opcode Cloud of Malware.

Following the creation of the dictionaries, the next phase involved transforming the opcodes into vectors used as input for the model generator. This mechanism was modeled using the One Hot Encoder 64 vectorization technique, achieving greater computational efficiency and reduced processing time. At this stage, 80,000 malware class opcodes (derived from 100 samples) and 80,000 goodware class opcodes (derived from 100 samples) were used as a training set. For the validation set, 17,000 malware class opcodes (derived from 20 samples) and

¹¹<https://github.com/capstone-engine/capstone>

¹²Address of the entry point, where the program execution begins.

¹³<https://hex-rays.com/ida-pro/>

Table 2. Distribution of the quantities of malware samples and opcodes by family.

Class	Family	# <i>malware</i>		# <i>opcodes</i>		
		(a)	(b)	(a)	(b)	(c)
1	Backdoor:Win32/Bifrose	2291	1079	42.935.835	28.186.045	3.621.441
2	Trojan:Win32/Vundo	6794	5644	123.161.189	101.995.711	3.621.441
3	BrowserModifier:Win32/Zwangi	920	468	4.910.368	3.621.441	3.621.441
4	Trojan:Win32/Koutodoor	5605	3937	45.849.096	27.896.441	3.621.441
5	Backdoor:Win32/Rbot	1170	771	35.864.389	26.273.978	3.621.441
6	Backdoor:Win32/Hupigon	1943	1174	115.136.258	79.103.051	3.621.441
7	Trojan:Win32/Startpage	1648	646	49.431.472	30.048.605	3.621.441
Total		20.371	13.719	417.288.607	297.125.272	25.350.087

Table 3. Distribution of the quantities of goodwill samples and opcodes.

# <i>goodware</i>			# <i>opcodes</i>		
(a)	(b)	(c)	(a)	(b)	(c)
20.371	17.683	13.719	135.592.306	117.733.342	25.350.087

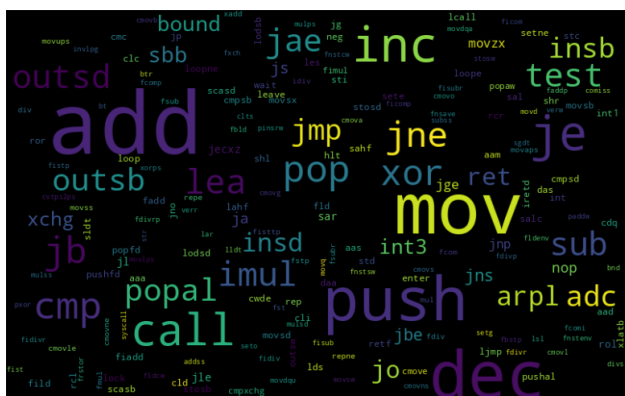


Figure 5. Opcode Cloud of Goodware.

17,000 goodwill class opcodes (derived from 20 samples) were used. The number of opcodes in the test set was adjusted according to the specific requirements of each experiment, as can be verified in Section 5.

4.5 Detection Mechanism

This section presents how the malware detection mechanism was modeled. A sequential neural network was used, equipped with four CuDNNLSTM layers, a variation of LSTM layers optimized for GPU processing. This choice was made because of the effectiveness of these layers in analyzing temporal sequences or sequential data, essential characteristics for the implementation of our proposal.

The *DetectMalware* function orchestrates the entire malware detection process. First, it extracts the instruction sequence from the input program P using the *ExtractInstructions* function. This extraction involves reading the program code and identifying the executed operations, such as “push”, “call”, “jnz”, “mov”, “add”, and “or”, among others.

After receiving the opcodes, the assembly instructions are converted into binary or one-hot vectors through the *Encode-Instructions* function. This procedure iterates over each instruction, converting it into a binary representation that can be processed by machine learning algorithms. The *Convert-ToBinary* function is responsible for this conversion, ensuring that each instruction is transformed into a unique numerical

Algorithm 1 DetectMalware

```

1: procedure DetectMalware(P)
2:   Instructions  $\leftarrow$  ExtractInstructions(P)
3:   EncodedInstructions  $\leftarrow$  EncodeInstructions(Instructions)
4:   MaliciousScore  $\leftarrow$  LSTMMaliciousModel(EncodedInstructions)
5:   BenignScore  $\leftarrow$  LSTMBenignModel(EncodedInstructions)
6:   Classification  $\leftarrow$  ArgMax(MaliciousScore, BenignScore)
7:   if Classification = MaliciousScore then
8:     return True
9:   else
10:    return False
11:  end if
12: end procedure

```

vector.

Algorithm 2 EncodeInstructions

```

1: procedure EncodeInstructions(Instructions)
2:   EncodedInstructions  $\leftarrow$  []
3:   for Instruction in Instructions do
4:     BinaryVector  $\leftarrow$  ConvertToBinary(Instruction)
5:     EncodedInstructions.Append(BinaryVector)
6:   end for
7:   return EncodedInstructions
8: end procedure

```

Once encoded, the instructions are processed by two distinct LSTM models: the *LSTM Malicious Model*, which estimates the probability that the program is malicious, and the *LSTM Benign Model*, which estimates the probability that the program is benign. Each LSTM model outputs a probability or confidence level associated with each class. The LSTM architecture is particularly suitable for this task due to its ability to process data sequences and identify complex patterns over time.

Finally, the *ArgMax* function compares the probabilities generated by the two models. This procedure determines the

final classification of the program, identifying it as malicious or benign based on the highest score obtained. If the probability calculated by the *LSTM Malicious Model* function is higher, the program is classified as malware; otherwise, it is classified as goodware. This step results in the final classification of the suspicious artifact, based on the class that showed the highest confidence or probability of association.

Example Usage

```
Program ← LoadProgram("path/to/program")
DetectionResult ← DetectMalware(Program)
Print(DetectionResult)
```

5 Performance Evaluation

To evaluate FORENSICS, experiments were conducted on hardware that includes an Intel® Core™ i7-10700 processor, an NVIDIA® GeForce® RTX™ 3060 graphics card, and 16GB of DDR4 memory. In terms of software, Linux Mint 20.1, Python 3.6.5, tensorflow-gpu 1.8.0, and Keras 2.2.0 were used. The neural network architecture was sequential and comprised four CuDNNLSTM layers. Two LSTM models were trained, each targeting a specific class of malicious artifacts, aiming to predict sequences of five opcodes. The Pandas and Numpy libraries were used to handle data, and the Scikit-Learn and TensorFlow libraries were used to implement ML and DL techniques.

To determine the minimum number of opcodes necessary to correctly detect malware, 126 experiments were conducted, equally distributed among each malware family. The number of mnemonics/opcodes tested ranged from 7 to 360 per segment, with a total of 1000 segments analyzed. These segments were evaluated within their respective malware families to determine suitability, allowing for a detailed understanding of detection efficacy by malware family. Additionally, to define the best hyperparameters for the neural network, experiments were conducted with various values, including batch sizes of 2, 5, 10, 20, 50, and 100; epochs of 20, 25, 50, and 100; lookback of 5; units of 64 and 128; loss function set as mean squared error; and the use of the Adam optimizer. Table 4 provides a description of the hyperparameters used in these experiments.

To evaluate FORENSICS, the following metrics were used: (i) Precision ($\text{Precision} = \frac{TP}{TP+FP}$) which assesses the accuracy of positive identifications; (ii) Accuracy ($\text{Accuracy} = \frac{TP+TN}{TP+TN+FP+FN}$), the percentage of correct predictions for all instances; (iii) Recall ($\text{Recall} = \frac{TP}{TP+FN}$), the proportion of relevant instances identified; (iv) F1-Score ($\text{F1-Score} = 2 \cdot \frac{\text{Precision} \cdot \text{Recall}}{\text{Precision} + \text{Recall}}$), balancing Precision and Recall; and (v) MCC ($\text{MCC} = \frac{c \times s - \sum_k (p_k \times t_k)}{\sqrt{(s^2 - \sum_k (p_k^2)) \times (s^2 - \sum_k (t_k^2))}}$), which considers all categories of predictions.

The joint analysis of the loss curves from the LSTM models trained for both malicious (Figure 7) and benign (Figure 8) classes reveals a stable and consistent learning behavior in both scenarios. In both cases, no significant divergence between the training and validation loss curves was observed,

Table 4. Set of parameters adopted for conducting the experiments in FORENSICS

Parameter	Value Range
Total experiments	126
Experiments per malware class/family	18
# of malware classes/families	7
# of segments analyzed	1000
# of mnemonics/opcodes per segment	Varies from 7 to 600
Batch size	2, 5, 10, 20, 50, 100
Epochs	20, 25 , 50, 100
Lookback	5
Units	64 e 128
Loss function	<i>mean squared error</i>
Optimizer	Adam

confirming the absence of overfitting. Both models demonstrated rapid convergence of the validation loss and maintained low and stable values until the end of the training epochs, reinforcing the networks' ability to generalize to unseen samples.

To further assess the generalization capability of the framework, the effectiveness of FORENSICS in detecting malware across different families was evaluated based on opcode sequence analysis, as shown in Figure 6. The results demonstrate high detection performance regardless of the analyzed family, confirming the model's ability to capture discriminative patterns across diverse malware variants.

Particularly, malware from the families Backdoor/Bifrose and BrowserModifier/Zwangi achieve high precision with few opcodes (approximately 30), while other families achieve high accuracy from 100 opcodes per 1000 segments. It is important to note that there are variations in the detection precision of other malware, such as Backdoor/Rbot, Backdoor/Hupigo, and Trojan/Vundo. These variations are attributed to the complexity of the opcode patterns and their similarity. It is highlighted that approximately 360 opcodes are necessary to achieve accurate detection, resulting in an average accuracy of 99.97%. This result illustrates the capability of FORENSICS to effectively recognize complex code patterns. It is worth noting that, for optimizing training time, each model was trained with only 80,000 opcodes. For the malicious model, this number of opcodes represents a small part of the opcodes from the Backdoor/Bifrose family, explaining the excellent performance of the model for this malware family.

The impact of opcode sequence length on detection accuracy was systematically evaluated. The experiments revealed that as the number of opcodes in each segment increases, the accuracy of the model also improves, particularly in families with more complex instruction patterns. While some malware families, such as Backdoor/Bifrose and BrowserModifier/Zwangi, reach high accuracy with fewer than 100 opcodes, others require longer sequences to achieve similar results. The detection accuracy stabilizes around 360 opcodes per sample, beyond which marginal improvements become negligible. This behavior highlights the LSTM model's ability to employ temporal dependencies more effectively with longer input sequences, while also suggesting that 360 op-

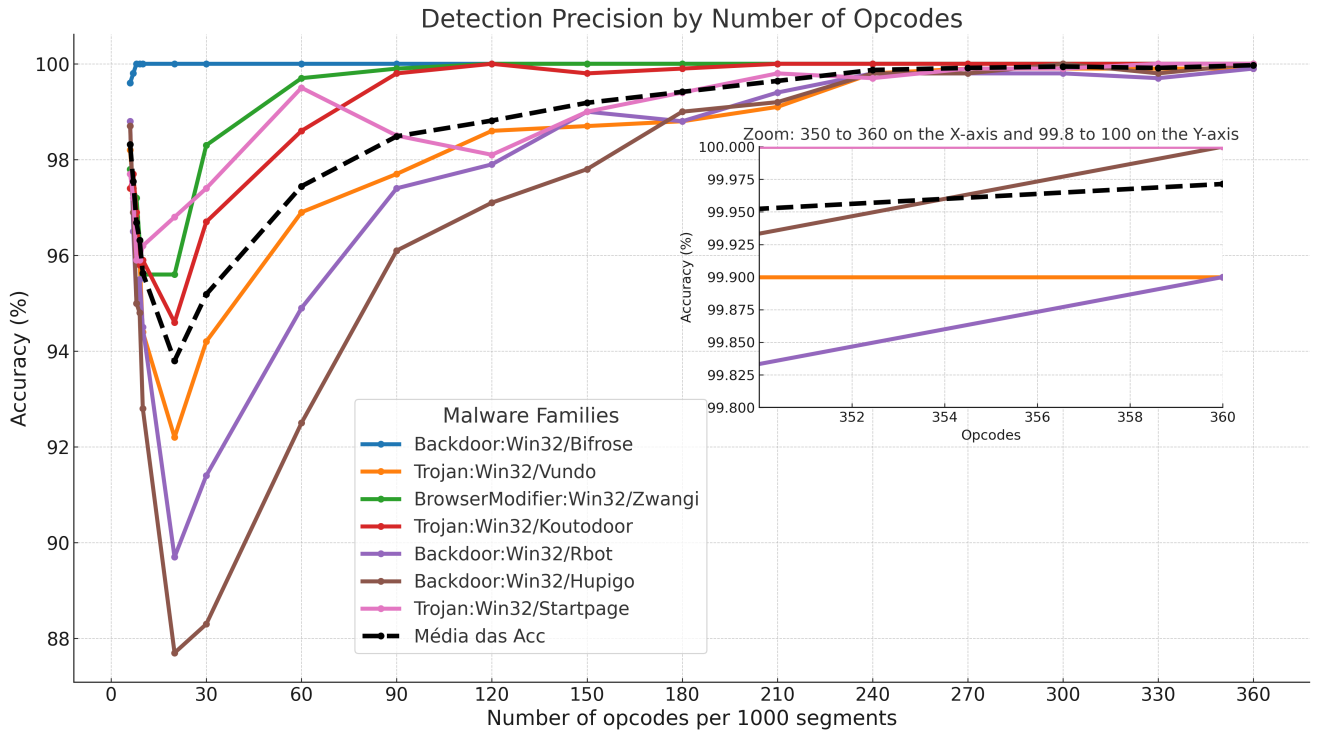


Figure 6. Impact of Accuracy on Malware Segment Detection.

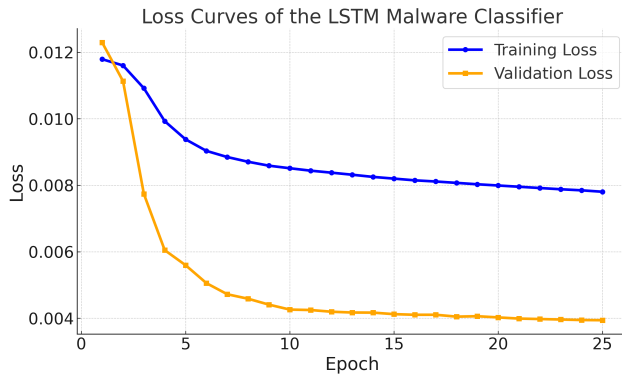


Figure 7. Loss curves obtained during the training of the LSTM model for malware classification (malicious class).

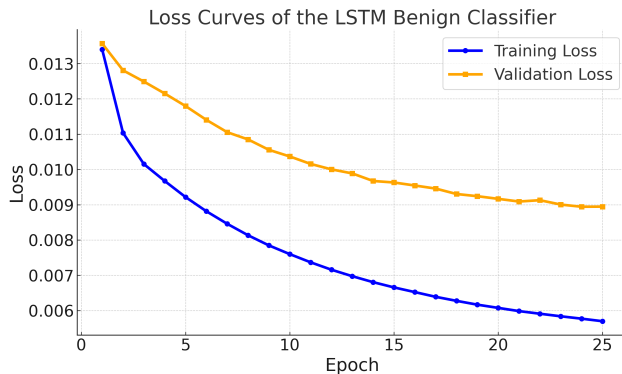


Figure 8. Loss curves obtained during the training of the LSTM model for malware classification (benign class).

codes represent a practical balance between detection performance and computational efficiency.

Next, considering only the first 360 opcodes of each malware from our entire repository, the performance of malware detection was evaluated, as shown in Figure 9. The evaluation

of FORENSICS revealed satisfactory performance across all analyzed metrics, achieving, in the best case, 99.91% F1-Score and, in the worst case, 99.8% MCC. These results indicate FORENSICS' ability to correctly detect suspicious artifacts, effectively balancing precision and sensitivity, while demonstrating its robustness and reliability in detections. The achieved precision of 99.99% also reflects the model's high capacity to minimize false positives. The overall accuracy of 99.91%, with recall values of 99.83%, and F1-Scores higher than 99.91%, presents robust overall performance, highlighting a balance between precision and recall and confirming the model's effectiveness even in the presence of potential class imbalances and a large amount of packaged malware. Table 5 presents the percentage of packed malware samples in each family of our dataset.

Table 5. Percentage of packed malware by family.

Family	Packed	Family	Packed
Family 1	39,30%	Family 5	6,36%
Family 2	33,03%	Family 6	35,95%
Family 3	2,14%	Family 7	51,39%
Family 4	66,78%	Total	41,77%

Finally, a comparison of FORENSICS with nine related works was conducted, as presented in Table 6. The results show that FORENSICS demonstrates superior performance, with 99.91% accuracy, regardless of the compared solution. Considering the lowest recorded value, 88.36%, attributed to Khan *et al.* [2019], it is noted that FORENSICS achieves a significant accuracy gain of 11.55%.

Regarding execution time, FORENSICS achieved a training time of approximately 128.34 seconds and a detection time for each artifact ranging from 8 to 17 milliseconds. This rep-

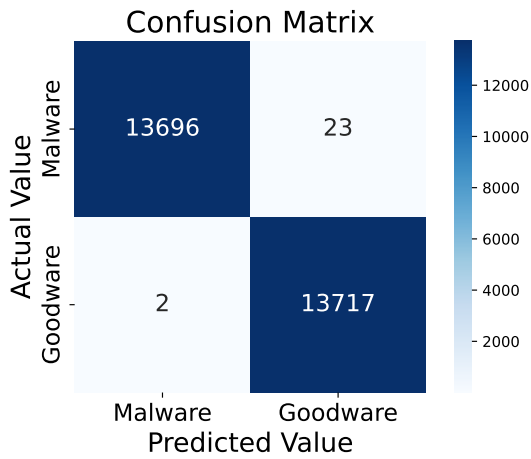


Figure 9. Confusion Matrix.

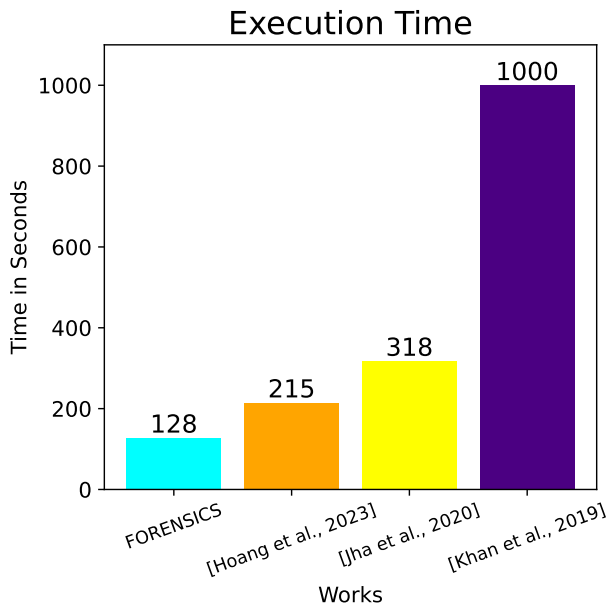


Figure 10. Performance Time

resents an improvement in execution time of approximately 40.46% compared to the previous best time of 215 seconds obtained by Hoang *et al.* [2023], as depicted in Figure 10.

Table 5 presents the percentage of packed malware samples in each family of our dataset. Packed malware often employs runtime encryption or compression to hinder static analysis and evade detection mechanisms. Notably, over 41% of all malware samples were packed, including high rates in families such as Family 4 (66.78%) and Family 7 (51.39%).

Additionally, a significant portion of the malware samples used in this study exhibited evasion techniques such as runtime code encryption and instruction obfuscation. Despite these adversarial mechanisms, FORENSICS consistently achieved high detection performance across all malware families. This resilience arises from the model's focus on learning behavioral patterns through opcode sequence analysis, rather than relying on static structural attributes or pre-defined signatures. As a result, FORENSICS demonstrates robust generalization capabilities even when facing dynamically obfuscated or anti-analysis malware.

The FORENSICS framework also yielded strong results in

Table 6. Impact of FORENSICS compared to the state of the art.

Work	Year	Accuracy (%)
[Baghirov, 2023]	2023	99
[Hoang <i>et al.</i> , 2023]	2023	96.29
[Carlson <i>et al.</i> , 2023]	2023	90
[Muppalaneni and Patgiri, 2021]	2021	98.93
[Mohandas <i>et al.</i> , 2021]	2021	97.97
[Parildi <i>et al.</i> , 2021]	2021	98
[Jha <i>et al.</i> , 2020]	2020	97.8
[Jeon and Moon, 2020]	2020	96
[Khan <i>et al.</i> , 2019]	2019	88.36
This	2025	99.91

detecting zero-day threats, achieving a 100% accuracy in tests with 238 previously unseen malware samples. These samples were selected for their high stealth level, characterized by the use of advanced evasion strategies capable of bypassing traditional security mechanisms. A representative subset of these malware, including their stealth methods and deployment scenarios, is provided in Table 7 (Appendix A). The full list of samples, including names and hashes, is available in the section “Availability of data and materials.”

Although metamorphic malware applies aggressive code transformation techniques, FORENSICS analyzes the temporal structure of opcode sequences, capturing latent behavioral patterns that often persist across variants. By focusing on specific instruction fragments, the framework is capable of detecting polymorphic threats and shows potential effectiveness even against metamorphic malware, depending on the degree of semantic preservation.

To avoid overfitting risks potentially introduced by the undersampling strategy, several methodological precautions were adopted. First, a deduplication process was applied to remove both identical binary files and redundant opcode sequences, ensuring that only unique and representative instances remained within each class, as described in subsection 4.4. The dataset was then split into separate training, validation, and test sets without overlap, thereby preserving the integrity of the evaluation. In addition, the architecture and hyperparameters of the neural network were tuned conservatively to avoid excessive model complexity. In particular, the framework was validated using a set of previously unseen zero-day malware samples, achieving an accuracy of 100%. These methodological measures collectively reinforce the robustness of the proposed model and indicate that no overfitting was observed during the experiments carried out. Altogether, the results validate the effectiveness of the FORENSICS framework in identifying a wide range of malware samples, including those employing advanced evasion strategies.

These results confirm that the classifiers were effective in capturing relevant discriminative patterns in opcode sequences, establishing themselves as promising tools for real-world deployment in automated malware detection environments. This effectiveness can be attributed to the combination of three key factors: the use of NLP techniques, the adoption of rigorous data preprocessing strategies, and the ability to detect malware using only a limited number of opcode instructions.

6 Conclusions and Future Work

Based on these results, this study presented FORENSICS, a lightweight and efficient deep learning-based framework for automatic malware detection through the analysis of variable-length instruction sequences. By employing opcode-level representations extracted from PE files and processing them with Long Short-Term Memory (LSTM) networks, the proposed framework achieved high accuracy and speed in identifying malicious behavior patterns. This approach not only optimized malware detection but also provided insights into the behavior of these cyber threats.

The framework demonstrated robust and consistent performance across seven different malware families, as evidenced by strong evaluation metrics: 99.91% accuracy, 99.99% precision, 99.91% F1-Score, and a Matthews Correlation Coefficient (MCC) of 99.80%. It also maintained a low false positive rate while ensuring high true positive detection—a critical balance for real-world cybersecurity systems. Moreover, FORENSICS proved resilient against advanced evasion techniques such as code obfuscation, packing, and zero-day malware, achieving 100% accuracy on a set of previously unseen samples.

One of the main strengths of FORENSICS is its computational efficiency. The framework requires only a limited number of opcodes per sample and relies on a compact model architecture, enabling malware detection in as little as 8 to 17 milliseconds per sample. This high detection speed, combined with a lightweight design, makes FORENSICS especially suitable for deployment on devices with constrained computational resources, such as embedded systems, IoT devices, and endpoint security solutions in edge environments.

A key outcome of this work is the development of a lightweight, fast, and accurate method for malware classification based on short opcode sequences. The framework successfully combines sequential pattern learning through LSTM networks with optimized preprocessing techniques and delivers results that support reproducibility through the release of a balanced and labeled dataset.

Future work will focus on expanding the FORENSICS framework through the evaluation of alternative recurrent neural network architectures (e.g., GRU, BiLSTM), the exploration of new opcode encoding strategies (including embedding-based and context-aware representations), and the incorporation of advanced data preprocessing methods to improve learning efficiency. These enhancements aim to further improve the adaptability, scalability, and effectiveness of FORENSICS in detecting diverse and evolving malware threats.

Declarations

Acknowledgements

The authors thank the University of Brasília (UnB), the LATITUDE Laboratory, and the research groups Projectum and Lincex for their institutional and technical support. This work was also supported by the Research Support Foundation of the Federal District (FAPDF), through the Grant Agreement No. 550/2024 — Call 06/2024 —

FAPDF Learning Program, and by the Coordination for the Improvement of Higher Education Personnel (CAPES).

Authors' Contributions

The authors equally contributed to the elaboration of this paper. All the authors read and approved the final manuscript.

Competing interests

The authors declare that they have no competing interests.

Availability of data and materials

The datasets generated and analyzed during the current study are available in SMARTNESS (*dataSet Malware fRom The wiNdows opErating SyStem*) repository, <https://github.com/CABorges72/SMARTNESS>.

References

- Abbaspour, S., Fotouhi, F., Sedaghatbaf, A., Fotouhi, H., Vahabi, M., and Linden, M. (2020). A comparative analysis of hybrid deep learning models for human activity recognition. *Sensors*, 20(19). DOI: 10.3390/s20195707.
- Abusitta, A., Li, M. Q., and Fung, B. C. (2021). Malware classification and composition analysis: A survey of recent developments. *Journal of Information Security and Applications*, 59:102828. DOI: 10.1016/j.jisa.2021.102828.
- Aggarwal, S. and Di Troia, F. (2024). Malware classification using dynamically extracted api call embeddings. *Applied Sciences*, 14(13). DOI: 10.3390/app14135731.
- Aslan, Ö. and Yilmaz, A. A. (2021). A new malware classification framework based on deep learning algorithms. *IEEE Access*, 9:87936–87951. DOI: 10.1109/ACCESS.2021.3089586.
- Aslan, O. A. and Samet, R. (2020). A comprehensive review on malware detection approaches. *IEEE Access*, 8:6249–6271. DOI: 10.1109/ACCESS.2019.2963724.
- Baghirov, E. (2023). Malware detection based on opcode frequency. *Problems of Information Technology*, 14:3–7. DOI: 10.25045/jpit.v14.i1.01.
- Barzev, I. and Borissova, D. (2025). An improved static analysis approach for malware detection by optimizing feature extraction combining different ml algorithms. In Bennour, A., Bouridane, A., Almaadeed, S., Bouaziz, B., and Edirisinghe, E., editors, *Intelligent Systems and Pattern Recognition*, pages 102–115, Cham. Springer Nature Switzerland. DOI: 10.1007/978-3-031-82153-0_8.
- Botacin, M., Alves, M. Z., Oliveira, D., and Grégio, A. (2022). Heaven: A hardware-enhanced antivirus engine to accelerate real-time, signature-based malware detection. *Expert Systems with Applications*, 201:117083. DOI: 10.1016/j.eswa.2022.117083.
- Carlson, J., Ralescu, A., Kebede, T., and Kapp, D. (2023). Experiments on recognition of malware based on static opcode occurrence distribution. In *NAECON 2023 - IEEE National Aerospace and Electronics Conference*, pages

- 98–103, Dayton, OH, USA. IEEE. DOI: 10.1109/NAE-CON58068.2023.10366040.
- Catak, F. O. and Yazici, A. F. (2019). A benchmark api call dataset for windows pe malware classification. *ArXiv*, abs/1905.01999. DOI: 10.48550/arXiv.1905.01999.
- Catal, C., Giray, G., and Tekinerdogan, B. (2022). Applications of deep learning for mobile malware detection: A systematic literature review. *Neural Comput & Applic*, 34(2):1007–1032. DOI: 10.1007/s00521-021-06597-0.
- de Andrade, C. A. B., Filho, G. P. R., Meneguette, R. I., Maranhão, J. P. A., Sant’Ana, R., Duarte, J. C., Serrano, A. L. M., and Gonçalves, V. P. (2025). Lightweight malware classification with fortunate: Precision meets computational efficiency. *Journal of Internet Services and Applications*, 16(1):87–104. DOI: 10.5753/jisa.2025.4905.
- El ghabri, N., Belmekki, E., and Bellaflkih, M. (2024). Pre-trained deep learning models for malware image based classification and detection. In *2024 Sixth International Conference on Intelligent Computing in Data Sciences (ICDS)*, pages 1–7. DOI: 10.1109/ICDS62089.2024.10756501.
- Ferdous, J., Islam, R., Mahboubi, A., and Islam, M. Z. (2023). A review of state-of-the-art malware attack trends and defense mechanisms. *IEEE Access*, 11:121118–121141. DOI: 10.1109/ACCESS.2023.3328351.
- Gaber, M. G., Ahmed, M., and Janicke, H. (2024). Malware detection with artificial intelligence: A systematic literature review. *ACM Comput. Surv.*, 56(6). DOI: 10.1145/3638552.
- Gulmez, S., Kakisim, A. G., and Sogukpinar, I. (2024). Analysis of the zero-day detection of metamorphic malware. In *2024 9th International Conference on Computer Science and Engineering (UBMK)*, pages 1–6. DOI: 10.1109/UBMK63289.2024.10773421.
- Habib, F., Shirazi, S. H., Aurangzeb, K., Khan, A., Bhushan, B., and Alhussein, M. (2024). Deep neural networks for enhanced security: Detecting metamorphic malware in iot devices. *IEEE Access*, 12:48570–48582. DOI: 10.1109/ACCESS.2024.3383831.
- Hebish, M. W. and Awni, M. (2024). Cnn-based malware family classification and evaluation. In *2024 14th International Conference on Electrical Engineering (ICEENG)*, pages 219–224. DOI: 10.1109/ICEENG58856.2024.10566448.
- Hoang, X. D., Nguyen, B. C., and Trang Ninh, T. T. (2023). Detecting malware based on statistics and machine learning using opcode n-grams. In *2023 RIVF International Conference on Computing and Communication Technologies (RIVF)*, pages 118–123, Hanoi, Vietnam. IEEE. DOI: 10.1109/RIVF60135.2023.10471824.
- Hochreiter, S. and Schmidhuber, J. (1997). Long short-term memory. *Neural Comput.*, 9(8):1735–1780. DOI: 10.1162/neco.1997.9.8.1735.
- Jannat Mim, M. M., Nela, N. A., Das, T. R., Rahman, M. S., and Ahmed Shibly, M. M. (2024). Enhancing malware detection through convolutional neural networks and explainable ai. In *2024 IEEE Region 10 Symposium (TENSYP)*, pages 1–6. DOI: 10.1109/TENSYP61132.2024.10752108.
- Jeon, S. and Moon, J. (2020). Malware-detection method with a convolutional recurrent neural network using opcode sequences. *Information Sciences*, 535:1–15. DOI: 10.1016/j.ins.2020.05.026.
- Jha, S., Prashar, D., Long, H. V., and Taniar, D. (2020). Recurrent neural network for detecting malware. *Computers & Security*, 99:102037. DOI: 10.1016/j.cose.2020.102037.
- Kale, A. S., Pandya, V., Di Troia, F., and Stamp, M. (2023). Malware classification with Word2Vec, HMM2Vec, BERT, and ELMo. *J Comput Virol Hack Tech*, 19(1):1–16. DOI: 10.1007/s11416-022-00424-3.
- Kaspersky (2023). 2023 threat intelligence report. Available at: <https://securelist.com/>.
- Khan, R. U., Zhang, X., and Kumar, R. (2019). Analysis of ResNet and GoogleNet models for malware detection. *J Comput Virol Hack Tech*, 15(1):29–37. DOI: 10.1007/s11416-018-0324-z.
- Li, C. and Zheng, J. (2021). Api call-based malware classification using recurrent neural networks. *Journal of Cyber Security and Mobility*. DOI: 10.13052/jcsm2245-1439.1036.
- Lu, R. (2019). Malware detection with lstm using opcode language.
- Mauri, L. and Damiani, E. (2025). Hardening behavioral classifiers against polymorphic malware: An ensemble approach based on minority report. *Information Sciences*, 689:121499. DOI: 10.1016/j.ins.2024.121499.
- Mehta, R., Jurečková, O., and Stamp, M. (2024). A natural language processing approach to Malware classification. *J Comput Virol Hack Tech*, 20(1):173–184. DOI: 10.1007/s11416-023-00506-w.
- Miao, C., Kou, L., Zhang, J., and Dong, G. (2024). A lightweight malware detection model based on knowledge distillation. *Mathematics*, 12(24):4009. DOI: 10.3390/math12244009.
- Mohammed, M., Abdalla, M., and Elhoseny, M. (2025). Detecting zero-day polymorphic worms using honeywall. *Journal of Cybersecurity and Information Management*, pages 34–49. DOI: 10.54216/JCIM.150104.
- Mohandas, P., Santhosh Kumar, S. K., Kulyadi, S. P., Shankar Raman, M. J., S, V. V., and Venkataswami, B. (2021). Detection of malware using machine learning based on operation code frequency. In *2021 IEEE International Conference on Industry 4.0, Artificial Intelligence, and Communications Technology (IAICT)*, pages 214–220, Bandung, Indonesia. IEEE. DOI: 10.1109/IAICT52856.2021.9532521.
- Muppalaneni, N. B. and Patgiri, R. (2021). Malware detection using machine learning approach. In Patgiri, R., Bandyopadhyay, S., and Balas, V. E., editors, *Proceedings of International Conference on Big Data, Machine Learning and Applications*, pages 219–225, Singapore. Springer Singapore. DOI: 10.1007/978-981-33-4788-5_18.
- Omar, M. (2022). *New Approach to Malware Detection Using Optimized Convolutional Neural Network*, pages 13–35. Springer International Publishing, Cham. DOI: 10.1007/978-3-031-15893-3_2.
- Or-Meir, O., Nissim, N., Elovici, Y., and Rokach, L. (2019). Dynamic malware analysis in the modern era—a state of the art survey. *ACM Comput. Surv.*, 52(5). DOI: 10.1145/3329786.
- Owoh, N., Adejoh, J., Hosseinzadeh, S., Ashawa, M., Os-

- amor, J., and Qureshi, A. (2024). Malware detection based on api call sequence analysis: A gated recurrent unit–generative adversarial network model approach. *Future Internet*, 16(10). DOI: 10.3390/fi16100369.
- Palma Salas, M. I., De Geus, P., and Botacin, M. (2023). Enhancing malware family classification in the microsoft challenge dataset via transfer learning. In *Proceedings of the 12th Latin-American Symposium on Dependable and Secure Computing*, LADC '23, page 156–163, New York, NY, USA. Association for Computing Machinery. DOI: 10.1145/3615366.3615374.
- Panda, B., Bisoyi, S. S., Panigrahy, S., and Mohanty, P. (2025). Machine learning techniques for imbalanced multiclass malware classification through adaptive feature selection. *PeerJ Computer Science*, 11:e2752. DOI: 10.7717/peerj-cs.2752.
- Parildi, E. S., Hatzinakos, D., and Lawryshyn, Y. (2021). Deep learning-aided runtime opcode-based Windows malware detection. *Neural Comput & Applic*, 33(18):11963–11983. DOI: 10.1007/s00521-021-05861-7.
- Qian, L. and Cong, L. (2024). Channel features and api frequency-based transformer model for malware identification. *Sensors*, 24(2):580. DOI: 10.3390/s24020580.
- Rumelhart, D. E., Hinton, G. E., and Williams, R. J. (1986). Learning representations by back-propagating errors. *Nature*, 323:533–536. DOI: 10.1038/323533a0.
- Shiri, F. M., Perumal, T., Mustapha, N., and Mohamed, R. (2023). A comprehensive overview and comparative analysis on deep learning models: Cnn, rnn, lstm, gru. DOI: 10.32604/jai.2024.054314.
- SonicWall (2023). 2023 cyber threat report: Shifting front lines. Available at: <https://www.sonicwall.com/news/2023-sonicwall-cyber-threat-report-casts-new-light-on-shifting-front-lines-threat-actor-behavior/>.
- Syeda, D. Z. and Asghar, M. N. (2024). Dynamic malware classification and api categorisation of windows portable executable files using machine learning. *Applied Sciences*, 14(3). DOI: 10.3390/app14031015.
- Tayyab, U.-e.-H., Khan, F. B., Durad, M. H., Khan, A., and Lee, Y. S. (2022). A survey of the recent trends in deep learning based malware detection. *Journal of Cybersecurity and Privacy*, 2(4):800–829. DOI: 10.3390/jcp2040041.
- Xiao, J. and Zhou, Z. (2020). Research progress of rnn language model. In *2020 IEEE International Conference on Artificial Intelligence and Computer Applications (ICAICA)*, pages 1285–1288, Dalian, China. IEEE. DOI: 10.1109/ICAICA50127.2020.9182390.
- Yoo, S., Kim, S., Kim, S., and Kang, B. B. (2021). Ai-hydra: Advanced hybrid approach using random forest and deep learning for malware classification. *Information Sciences*, 546:420–435. DOI: 10.1016/j.ins.2020.08.082.
- Zhang, J., Qin, Z., Yin, H., Ou, L., and Zhang, K. (2019). A feature-hybrid malware variants detection using cnn based opcode embedding and bpnn based api embedding. *Computers & Security*, 84:376–392. DOI: 10.1016/j.cose.2019.04.005.
- Zhao, J., Basole, S., and Stamp, M. (2021). Malware Classification with GMM-HMM Models. arXiv. arXiv:2103.02753 [cs, stat]. DOI: 10.48550/arXiv.2103.02753.

7 Appendices

A Tested Malware in FORENSICS

Table 7. Tested Malware with Stealth Techniques and Usage Scenarios

Malware Name	Stealth Techniques	Usage Scenario
Worm:Win32/Flame.gen!A	Advanced espionage techniques, data capture, and hidden communication.	Used in espionage campaigns in the Middle East, primarily targeting governmental and energy infrastructure.
Trojan:WinNT/Stuxnet.A	Used legitimate digital certificates to avoid detection.	Attacked Iran's nuclear program by infecting industrial control systems.
Ransom:Win32/WannaCrypt	Exploited SMB vulnerabilities and disabled security measures.	Global ransomware attack, affecting hospitals, businesses, and governments.
Backdoor:MSIL/Orcus.A!bit	Remote control by disabling security tools.	Used in espionage and remote control of systems, mainly targeting individuals and small organizations.
Trojan:Win32/CrashOverride.A!dha	Targets critical infrastructure, initially hiding its activities.	Used to disable Ukraine's power grid.
Trojan:WinNT/Duqu.A	Steals information from industrial systems using espionage techniques.	Cyber espionage in industrial systems in Europe and the Middle East.
Backdoor:Win32/Blad-abindi.B	Disables security tools and operates silently.	Remote control campaigns and theft of sensitive information.
TrojanDownloader:Win32/Nymaim.K	Installs other malware without initial detection.	Attacks on financial institutions, downloading ransomware, and banking trojans.
Trojan:Win32/Occamy.C	Hides its activities and compromises systems, avoiding detection.	Used to steal credentials and personal information from end-user systems.
Backdoor:Win32/Zegost	Steals information and allows remote access, remaining hidden.	Espionage in corporate and governmental networks.
Trojan:Win32/Havex.k	Cyber espionage in industrial systems, operating stealthily.	Targeted industrial control systems (ICS) in the energy and manufacturing sectors in Europe and the U.S.
Ransom:Win32/Petya	Encrypts hard drives and spreads across networks.	Massive attacks on companies and institutions, affecting healthcare and financial networks in Europe.
Backdoor:Win32.Cosmic-Duke.hko	Combines espionage and data theft, hiding communication with control servers.	Espionage against governmental organizations and NGOs in Europe.
Trojan:Win32/Fuerboos.E!cl	Modifies system files to avoid detection.	Theft of personal information and banking credentials.
Trojan:Win32/Ditertag.A	Obfuscates its code to avoid analysis and detection.	Attacks against financial organizations and technology companies.
Worm:Win32/Conficker.B	Disables security systems and modifies group policies.	One of the largest worm infections in history, affecting millions of computers worldwide.
TrojanDownloader:Win32/Upatre!rfn	Downloads other malware undetected, using encryption to hide its activities.	Distribution of ransomware and banking trojans in corporate networks.
Trojan:Win32/Skeeyah.A!rfn	Obfuscates its code and enables remote control.	Used in targeted attacks, stealing sensitive data and allowing remote control.
Trojan:Win32/Tiggre!rfn	Steals data and installs backdoors for remote access.	Used in cyber espionage campaigns and theft of banking credentials.
Worm:Win32/Dorkbot.A	Steals banking information and credentials, using encryption to hide its activities.	Spreads via social networks and instant messaging, aiming to steal banking credentials.