**Frontiers in Computing and Intelligent Systems**
ISSN: 2832-6024 | Vol. 6, No. 3, 2023

# Artificial Intelligence and Malicious Code Detection

**Chenhui Zhong ***

School of Computer Science, South China Normal University, Guangzhou, 510631, China
*** Corresponding author Email:** 20192132020@m.scnu.edu.cn

**Abstract:** Malicious software has become a critical cybersecurity issue due to the increasing number of threats it poses to devices and internet environments. Traditional static scanning techniques and behavior-based malware detection methods have limitations in meeting the new requirements in information security due to high false positives and false negatives. In this work, we propose a CNN convolutional neural network-based method for detecting malicious code. Practical operations were conducted using the Cuckoo sandbox system, and Python programs were utilized to preprocess analytical reports. This article presents the construction of a deep learning training model for CNN designed to identify malicious code. The model is compared with machine learning and general antivirus tools for comprehensive evaluation. Experimental verification shows that our proposed method exhibits greater advantages in comparison and achieved excellent detection results with higher feasibility. The research significance of this work is highlighted as malicious software has become a core issue in cybersecurity. The method proposed in this article provides powerful support for addressing new needs in information security. This paper emphasizes the importance of utilizing CNN-based methods in detecting malware, which can better address the limitations of traditional detection techniques. Overall, this work provides an effective solution to detect malicious code and addresses a critical cybersecurity issue.

**Keywords:** Cybersecurity; Malicious Code; Cuckoo Sandbox; CNN.

## 1. Introduction

Due to the widespread adoption of online payments and internet banking, computers have become an indispensable part of our daily lives. (Adversarial Research on Visual Detection Methods for Malicious Code[1]) Conversely, most IoT devices lack adequate security measures, rendering them vulnerable targets for cyber attackers. According to a 2017 report by McAfee, global economic losses stemming from cybercrime reached a staggering $600 billion that year, accounting for approximately 0.8% of the global GDP.(Static Detection Technique for Malicious Code Based on Feature Sequences [2-4]) In the face of the growing ubiquity of malicious software and their potential for devastation, computer users are now bearing a higher cost to ensure the stability of their systems. Consequently, the reconnaissance and defense against malicious software have evolved into central and pivotal objectives within the field of cybersecurity. Presently, the detection of malicious code primarily involves two key steps: feature extraction and detection. During the feature extraction process, our focus lies in extracting static and dynamic characteristics, employing static feature extraction methods and dynamic feature extraction methods, respectively.

## 2. Overview of Malicious Code

Maliciously crafted code refers to applications or documents that violate user intentions and infringe upon their rights, with the aim of destroying, dismantling, or stealing data and information from the targeted systems.(Malicious Code Detection Based on Squeeze-Excitation Networks [5,6]) Malicious code comes in various forms, each capable of infecting and causing varying degrees of damage to devices. However, all variants of malicious code share a common objective: compromising the security and privacy of computer systems. The purposes behind malicious programming may manifest through diverse methods, such as blocking access, data destruction, theft, resource hijacking, dissemination of false messages, propagation of malicious programming languages, and numerous other malicious actions. Moreover, in many instances, the detection of malicious code can become challenging. Malicious programs also employ a variety of strategies to disseminate their payload beyond the initial attack platform to other computer systems.

Methods of Attack by Malicious Code: (1) Attacking Malicious Websites: When accessing infected malicious websites, victims may unwittingly download m malicious code while accessing resources on the site. These malicious codes can be concealed within various file types, such as videos, images, and software applications. (2) Email-Based Delivery: Malicious code can be delivered through email attachments. If recipients open these infected email attachments, the malicious code can operate independently, leading to system compromise. Sharing such emails can potentially propagate the malicious code further across networks, posing an even greater threat. (3) Exploiting Remote System Vulnerabilities: Vulnerabilities in remote systems can provide opportunities for attackers to infiltrate systems. (4) File Sharing Software: File-sharing software can facilitate the replication of malicious code on removable media, subsequently spreading it to computer systems and networks. For instance, P2P file-sharing may unknowingly load malicious code through seemingly harmless files. In recent years, malicious software has significantly increased its targeting of mobile devices. In addition to the vulnerabilities present in internet platforms and email media, mobile phones can also be compromised through the delivery of malicious code via text messages and multimedia messages.

## 3. Setting up the Cuckoo Sandbox Environment

The primary function of the Cuckoo Sandbox environment is to dynamically analyze malicious code while providing real-time execution and monitoring of malicious files.(Research on Malicious Code Detection Technology Based on Word Embedding and API Calls [7]) Within a completely isolated, transparent, and secure analysis environment, PE files are executed, and rigorous monitoring of sample dynamics is carried out. This constitutes the fundamental steps of dynamic analysis for malicious code. Various sandboxing, virtualization, and emulation techniques are employed to create an environment that simulates the actual operational state of the files.

The Cuckoo Sandbox primarily relies on the GPLv3 open-source license and is used for initial dynamic analysis of malicious software. Its core functionalities include:

(1) Tracking the usage status of malicious code across various processes.

(2) Monitoring actions such as document removal, uploading, or creation during the execution of malicious software.

(3) Tracking the propagation activities of malicious code on the internet and recording them in PCAP format.

(4) Acquiring memory snapshots of malicious code.

The core components of the Cuckoo Sandbox consist of the Host Machine and Guest Machine, which communicate with each other through a virtual network infrastructure. The Host Machine's primary functionalities are facilitated by the Cuckoo Sandbox software, VirtualBox VM software, and various analysis components. These functions encompass launching sample analyses, behavior monitoring, and generating report files. The Guest Machine's primary responsibility is to execute malicious code and provide analysis results back to the Host Machine. The architecture of the Cuckoo Sandbox is illustrated in Figure 1.

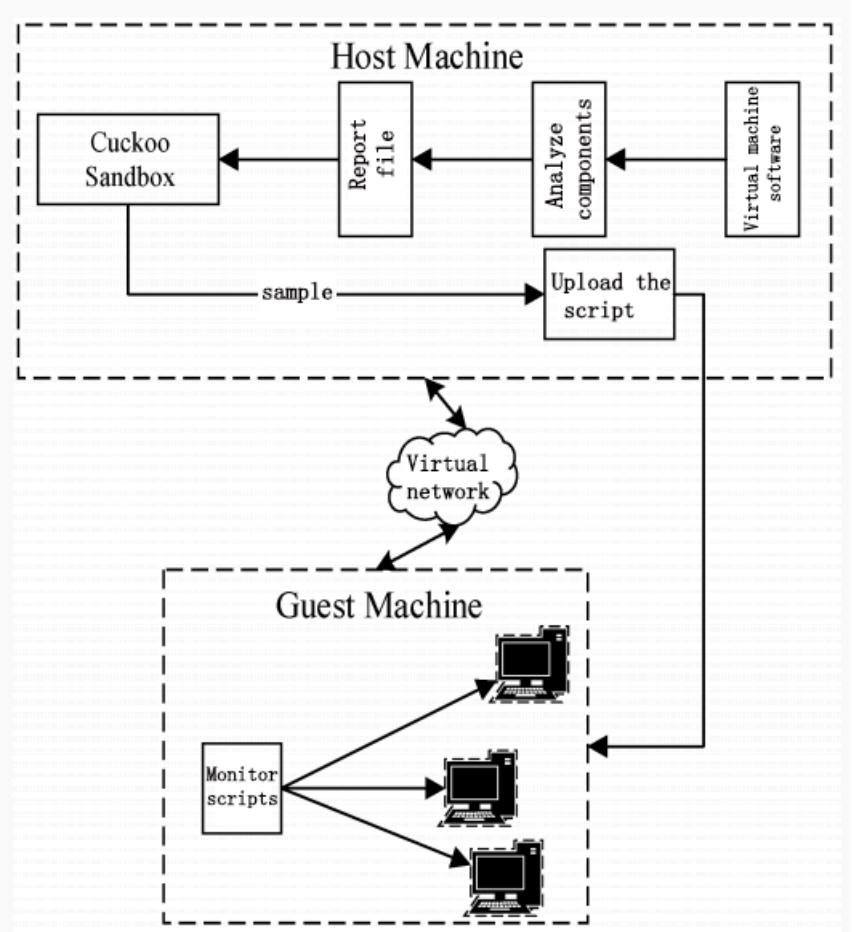


**Figure 1.** Cuckoo Sandbox Structure

## 4. Design of a Malicious Code Detection Model based on CNN

Many machines learning-based approaches have been successfully developed to address the challenges posed by a wide range of malicious software. (Malicious Code Detection Method for Network Communication Based on Texture Features [8,9]) Previous research has attempted to tackle the problem of modeling malicious software behavior, utilizing not only behavioral data but also static code attributes as sources of statistical analysis. Additionally, some studies have endeavored to merge both static and dynamic techniques. Currently, deep learning has demonstrated significant performance improvements in various domains, garnering considerable attention from researchers. However, the application of these methods in malicious software analysis remains relatively limited. Nevertheless, there are efforts to incorporate deep learning into this application domain. For instance, feedforward neural networks have been employed for parsing malicious code, while recurrent neural networks have been utilized to establish system call sequences, thus constructing language models for malicious code. In this experiment, convolutional neural networks (CNNs) are employed for the detection and analysis of malicious software. Figure CNN illustrates the foundational malicious code

detection model built upon CNN.

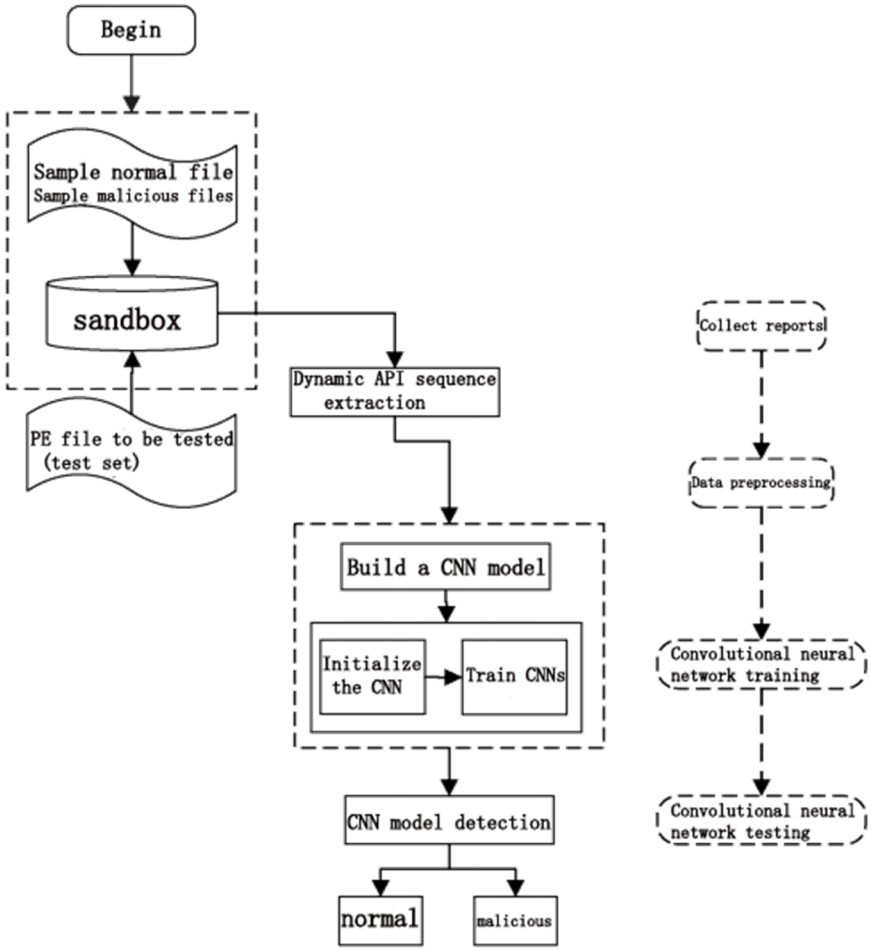


**Figure 2.** Driver-Based Malicious Code Reconnaissance Pattern

The sample files were uploaded to the sandbox virtual machine system to obtain the analysis reports required for our experiments, ultimately generating analysis reports in the ISON format. In these reports, this article primarily covers relevant information such as traced function calls, network operations, file operations, and more. All these aspects will provide robust support for the subsequent data preprocessing tasks. The data acquired far exceeds the amount of information needed for the analysis reports; therefore, preprocessing is deemed necessary before utilizing this data, as stated in this article. This article primarily focuses on the extraction of information about dynamic API functions. The reason for collecting these attributes is that the introduced functions can assist the neural network in conclusively identifying malicious software samples. For example, functions initiated by kernel32.dll, such as OpenProcess and GetCurrentProcess, can be used to execute GetProcessHeap, indicating that the malicious software has initiated and manipulated processes. Most of the Win32 API functions in the DLL documentation include GUI-related functions, such as RegisterClassEx, SetWindowText, and ShowWindow, revealing that malicious software not only possesses GUI functionality but may also mimic the design of normal application GUIs. The initiation of other programs by the malicious software is attributed to functions introduced by shel132.dll.

# 5. Experiment

## 5.1. Experimental Environment and Experimental Samples

Computer A serves as a programming environment for the execution of malicious code detection techniques based on convolutional neural networks (CNNs). It encompasses the deep learning framework TensorFlow, runs on Windows 10 x64, and is equipped with an Intel Core i7 8-core 2.4 GHz CPU, 8 GB of DDR4 RAM, and a 512 GB SSD solid-state drive.

Table 1 illustrates the configuration of the Cuckoo sandbox environment (Computer B).

**Table 1.** Sandbox Configuration Environment

| Experimental environment | Disposition |
|---|---|
| Processor | Intel (R) Core (TM) i7-8550U |
| Frequency | 1.80 GHz |
| Memory | 8GB |
| Operating system | Ubuntu 14.04 |
| Software environment | VBox5.1, Python2.7, Cuckoo2.0.5 |
| Sandbox operating system | Windows 7 |
| Sandbox software environment | WPS, QQ, IE, browser, mail, Python2.7 |

In this paper, research was conducted on a selection of the most prevalent malicious data on the Windows platform,

which was obtained from https://virusshare.com and www. malware-traffic-analysis, two publicly accessible platforms. A sample set of 2,400 Windows malicious PE files was collected. Additionally, a normal sample set was acquired from the official 360 app store, consisting of 1,000 samples. These samples included 16 different types of software based on usage, such as web browsers, text editors, office suites, and media players, among others. In total, there were 3,400 samples.

## 5.2. Comparison of CNN Results under Different Filters

Primary detection was conducted in the convolutional layers using four main filters: 3, 4, 5, and 6, as described in this article. Figure 3 illustrates the comparison between CNN classification results using One-Hot encoding and CNN classification results using the Skip-gram model, in terms of accuracy.

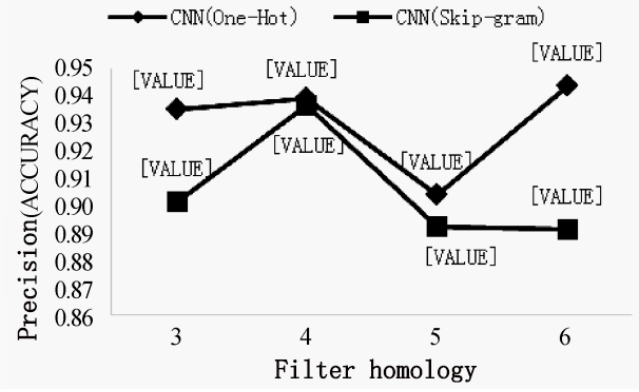


**Figure 3.** Comparison of Accuracy Under Different Filters

After adjusting the filter heights, it was found that the CNN model using One-Hot encoding exhibited superior performance in terms of accuracy, whereas the CNN model using Skip-gram encoding showed relatively poorer results, as discussed in this article. This strongly indicates that One-Hot encoding is better suited for vectorizing sparse API features. Consequently, in subsequent experiments, One-Hot encoding was predominantly employed to create word vectors and it was compared with conventional machine learning algorithms and antivirus software, as detailed in this article.

## 5.3. Comparison with Common Machine Learning Algorithms

In Experiment 2, three rounds of cross-validation were conducted to predict new data outcomes, as reported in this article. Throughout all phases of the experiments, the dataset was randomly partitioned into three equally sized portions and trained on two of them while testing on the remaining one, as detailed in this article. This entire process was repeated three times, with only one independent portion reserved for testing in each iteration. By calculating the averages of these three tests, a reliable evaluation method for assessing the performance of the proposed convolutional neural network model across the entire dataset was ultimately established, as outlined in this article. Additionally, the performance of the convolutional neural network (One-Hot) model was compared with MLP, NaiveBayes, SVM, and CNN, as detailed in this article. The final model's performance was assessed using three quantitative metrics: Accuracy, Recall, and F1-score. Please refer to Table 2 for experimental data in this article.

**Table 2.** Comparison of Classifier's Average Performance

| Algorithm | Accuracy | Recall | F1-score |
|---|---|---|---|
| MLP | 0.92 | 0.91 | 0.91 |
| Naive Bayes | 0.84 | 0.75 | 0.76 |
| SVM | 0.90 | 0.89 | 0.88 |
| CNN | 0.94 | 0.92 | 0.93 |

Based on the data in Table 2, it can be observed that the CNN model outperforms several common machine learning algorithms in terms of overall Accuracy, Recall, and F1-score, as indicated in this article. This implies that CNN exhibits a significant advantage over these other machine learning algorithms in these aspects.

## 5.4. Comparison with Common Antivirus Software

Experiment 3 involved obtaining an additional 100 asynchronous samples of PE files, as detailed in this article. Each submitted sample in VirusTotal was scanned by antivirus software such as ClamAV and TotalDefense ZoneAlarm Malwarebytes. Here, R = n/t represents the number of malicious samples detected by VirusTotal, where t denotes the total number of uploaded malicious samples. These samples were examined using CNN (One-Hot), and their results were compared with those obtained using antivirus tools like ClamAV, TotalDefense ZoneAlarm Malwarebytes, and others. Detailed data can be found in Table 3.

**Table 3.** Comparison of Detection Rates for Unknown Malicious Software

| Antivirus | R |
|---|---|
| Clam AV | 64% |
| Total Defense | 54% |
| Zone Alarm | 82% |
| Malware bytes | 43% |
| CNN | 91% |

# 6. Conclusion

This article draws upon practical experience in the field of natural language processing to construct a convolutional neural network model and applies it to the detection of malicious code. Experimental validation demonstrates that integrating deep learning algorithms into the domain of malicious code detection represents one of the most significant innovations to date, yielding detection results surpassing those of traditional machine learning algorithms. One limitation of this article is the relatively small size of the experimental dataset, which may lead to incomplete model training and, subsequently, impact the final detection results. Furthermore, a major advantage of deep learning lies in its ability to automatically learn features. In the future, consideration will be given to leveraging deep learning for feature automation and subsequently applying the derived features to machine learning classifiers to achieve superior experimental results, as discussed in this article.

# References

[1] Huang, K. (2023) Adversarial Research on Visual Detection Methods for Malicious Code. Thesis of Guizhou Normal University, 63.

[2] Wei, L., Shi, C., Xu, F., et al. (2022) Static Detection Technique for Malicious Code Based on Feature Sequences. Cyber Security and Data Governance, 41(10): 56-64.

[3] Long, M., Kang, H. (2023) Malicious Code Detection Method for Industrial Internet Based on Code Visualization. Computer Integrated Manufacturing System, 14.

[4] Liu, Y., Li, J., Ou, Z., et al. (2022) Adversarial Training-Driven Enhancement Method for Malicious Code Detection. Journal of Communications. 43(09): 169-180.

[5] Shen, G. (2023) Malicious Code Detection Based on Squeeze-Excitation Networks. Thesis of Minnan Normal University, 09:63.

[6] Jiang, R. (2023) Research on Malicious Code Detection Based on Neural Networks. Thesis of Southwest University of Science and Technology, 54.

[7] Xin, W. (2023) Research on Malicious Code Detection Technology Based on Word Embedding and API Calls. Thesis of North University of China, 81.

[8] Cai, D. (2022) Malicious Code Detection Method for Network Communication Based on Texture Features. Digital Communication World, 06: 79-81.

[9] Yang, Y. (2023) Research on Malicious Code Detection Based on Feature Fusion and Machine Learning. Thesis of Hebei University of Architecture, 02:61.