Research Plan

Explainable Malware Detection Using NLP-Based Approaches

Rasoul RezvaniJalal

May 31, 2025

1 Introduction

The increasing sophistication of malware attacks has become a critical concern in modern cybersecurity. Traditional detection methods, while effective, face significant challenges in keeping pace with evolving threats. The integration of artificial intelligence and machine learning has emerged as a crucial advancement in malware detection, particularly for various file types including windows executable [3], Android applications [6], and PDF documents [4]. However, most of these techniques have limitations that will be described in next parts. Having explainability in malware detection systems can enhance detection accuracy and also reduce analysis time spent by malware analysts. In this document a mechanism will be provided demonstrating how using LLMs can involve in detection workflow.

1.1 Problem Statement

Current malware detection systems face several significant limitations:

1.1.1 Detection Limitations

- Traditional machine learning approaches often produce binary classifications
- Insufficient context for decision-making
- Time-consuming analysis processes
- Black-box decision making in binary classification methods without knowing the sample behavior

1.1.2 Information Gaps

- Lack of comprehensive threat intelligence
- Limited severity assessment capabilities
- Inadequate confidence scoring mechanisms

1.2 Research Opportunity

The emergence of Natural Language Processing (NLP) and Large Language Models presents a significant opportunity for advancing malware detection:

1.2.1 Enhanced Analysis Capabilities

- Automated generation of detailed threat descriptions
- Improved context understanding
- Enhanced decision support systems

1.2.2 Decision Support

- Accelerated analysis processes
- More informed decision-making
- Enhanced situational awareness

2 Methodology

In order to enhance the detection capability, various methodologies can be provided. For example traditional detection systems such as binary classification can be incorporated with LLMs and other factors like confidence score [5] or severity score can be involved to help decision makers, make more accurate decisions. In the provided methodology in this document which is presented in Figure 1, the preparing of new methodology containing LLMs solely is demonstrated.

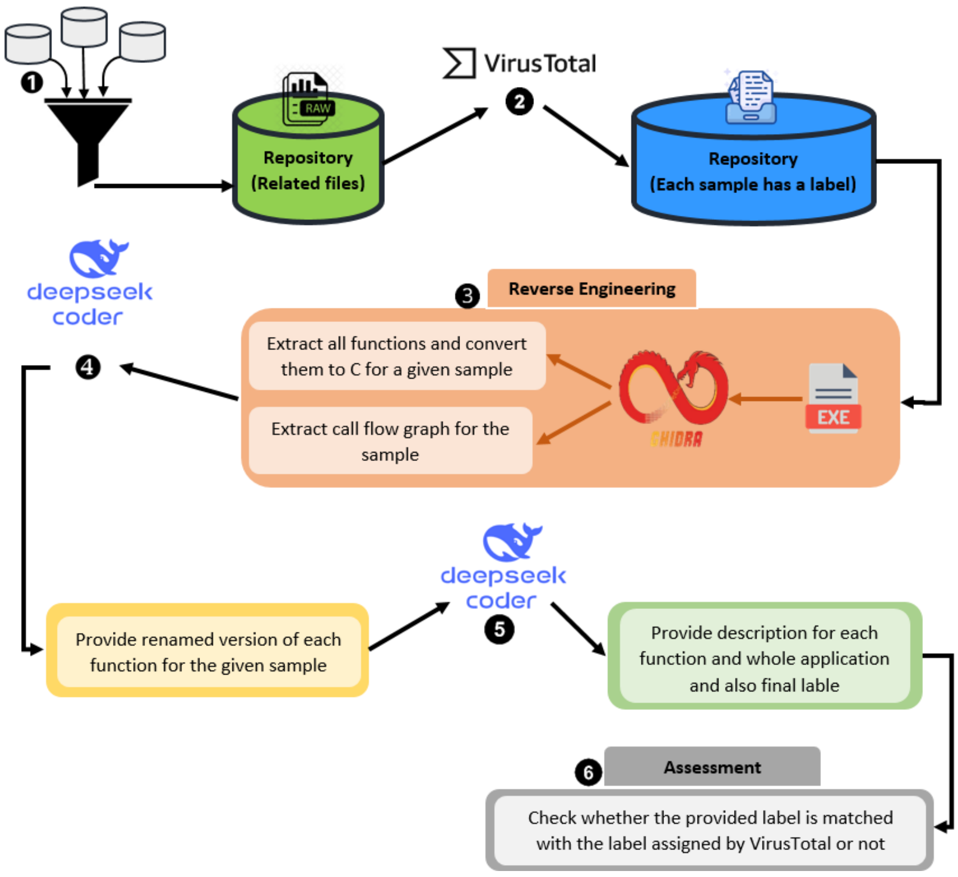
Here is some notes about each step within this structure:

1. **Data Collection**: Related files which in this document are windows executable files, must be gathered from different resources such as virus-share and github in order to create the repository.

2. **Label Assigning**: In malware analysis context, ensuring the maliciousness of samples is vital. To this end, the most reliable way is using Virus-Total API and assign proper label to each sample.

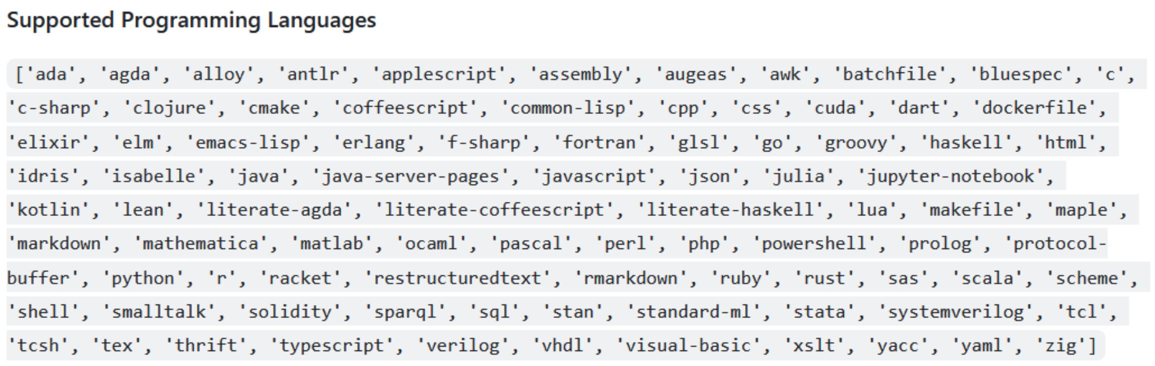
3. **Reverse Engineering**: In this step, all functions within the sample are detected and converted to C equivalent using related tools and scripts. Also the relation established between function which is known as call flow graph is important to reach a precise behavior. Furthermore, there is a limitation associated with large language modes called token restrictions, so converting the code to C can help to evade this limitation to some extent.

4. **Function Renaming**: For this stage, the call flow graph is processed and starting from leaves and ending with the root, functions are renamed to their actual name based on their behavior.



1
Repository
(Related files)
RAW
VirusTotal
2
Repository
(Each sample has a label)
deepseek coder
3
Reverse Engineering
Extract all functions and convert them to C for a given sample
Extract call flow graph for the sample
GHIDRA
EXE
4
deepseek coder
5
Provide renamed version of each function for the given sample
Provide description for each function and whole application and also final lable
6
Assessment
Check whether the provided label is matched with the label assigned by VirusTotal or not

Figure 1: The methodology of preparing model to detect malicious samples



Supported Programming Languages

```
['ada', 'agda', 'alloy', 'antlr', 'applescript', 'assembly', 'augeas', 'awk', 'batchfile', 'bluespec', 'c',
'c-sharp', 'clojure', 'cmake', 'coffeescript', 'common-lisp', 'cpp', 'css', 'cuda', 'dart', 'dockerfile',
'elixir', 'elm', 'emacs-lisp', 'erlang', 'f-sharp', 'fortran', 'glsl', 'go', 'groovy', 'haskell', 'html',
'idris', 'isabelle', 'java', 'java-server-pages', 'javascript', 'json', 'julia', 'jupyter-notebook',
'kotlin', 'lean', 'literate-agda', 'literate-coffeescript', 'literate-haskell', 'lua', 'makefile', 'maple',
'markdown', 'mathematica', 'matlab', 'ocaml', 'pascal', 'perl', 'php', 'powershell', 'prolog', 'protocol-
buffer', 'python', 'r', 'racket', 'restructuredtext', 'rmarkdown', 'ruby', 'rust', 'sas', 'scala', 'scheme',
'shell', 'smalltalk', 'solidity', 'sparql', 'sql', 'stan', 'standard-ml', 'stata', 'systemverilog', 'tcl',
'tcsh', 'tex', 'thrift', 'typescript', 'verilog', 'vhdl', 'visual-basic', 'xslt', 'yacc', 'yaml', 'zig']
```

Figure 2: The programming languages which are supported by Deep Seek Coder [1]

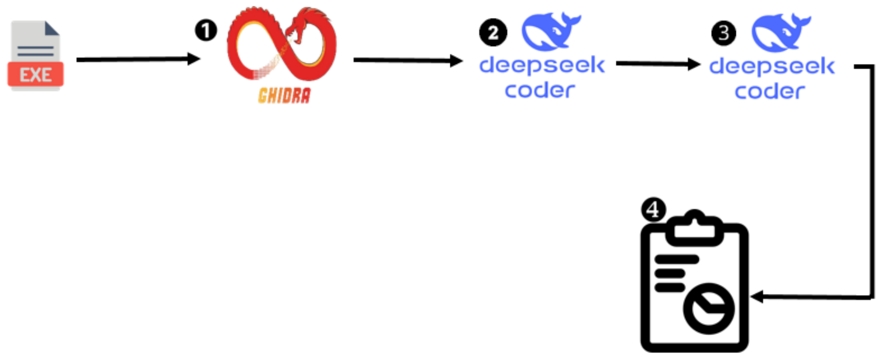


Figure 3: The detection flow of a sample

5. **Provide Behavior**: Based on renamed functions and the relations, the employed LLM provide description demonstrating their behavior. When the flow reach to root functions, the provided description and behavior illustrate whole application behavior.

6. **Assessment**: Based on the assigned labels from Virus-Total, the accuracy rate is calculated and reported.

At first glance, the Deep-Seek-Coder can be a good candidate to be employed as the above-mentioned LLM model in the methodology. This model has been trained on many programming languages including C. The list of supported languages is shown in Figure 2. However, In case the performance of the raw model is not good, leveraging techniques such as fine-tuning [2] is necessary.

After ensuring the model performance, it is possible to deploy the model to detect malicious samples in real-world scenarios. This is shown in Figure 3.

As it was explained, the sample first is processed by a debugging tool such as ghidra and then all function are renamed. Then all renamed functions are analyzed and their behavior are determined and a final label and a severity score assigned to the sample. (based on the given prompt)

3 Research Questions

1. What is the accuracy of LLM-Based malware detection?

2. What is the time complexity of this model?
3. How can we develop an AI-enhanced malware detection system that integrates natural language processing with traditional detection methods to improve accuracy and reduce false positives in real-time threat analysis?
4. What are the optimal parameters and architectural components for integrating large language models with existing malware detection systems to achieve maximum detection accuracy while maintaining computational efficiency?
5. How to reduce time complexity in this AI-enhanced malware detection system?

4 Steps

- Step 1: Literature review
- Step 2: Data collection
- Step 3: Label assigning
- Step 4: Data analysis and Debugging automation
- Step 5: Renaming process
- Step 6: Behavior extraction process
- Step 7: Assess the performance
- Step 8: Fine-Tuning models if it is needed
- Step 9: Write and submit the final report

5 Similar Research Opportunities

In the context of malware detection two different general approaches are presented which are static analysis and dynamic analysis. This document provide a mechanism to detect windows executable malwares based on static analysis using a debugger. This scenario could be used for dynamic analysis too. In this idea, different information is collected from a predesignated sandbox and the LLM make some output based on given information and the input prompt. In this approach many limitations including packed and encrypted samples which exist in static analysis are mitigated but it has various overheads including more complex resources and tool and higher time complexity. However, static and dynamic analysis complement each other and both are necessary. Furthermore, this report is providing methodology to detect specific type of file which can be fostered and improved to detect other well-known types such as android file type. Moreover, in order to enhance this methodology capability, other concepts can be integrated with this methodology such as confidence score which reduce the time wasting.

References

[1] DeepSeek. Explanations about DeepSeek Coder, https://github.com/deepseek-ai/DeepSeek-Coder, [Accessed: 2024].

[2] Simon Friederich. Fine-tuning. *The Stanford encyclopedia of philosophy*, 2017.

[3] Matthew G Gaber, Mohiuddin Ahmed, and Helge Janicke. Malware detection with artificial intelligence: A systematic literature review. *ACM Computing Surveys*, 56(6):1–33, 2024.

[4] Maryam Issakhani, Princy Victor, Ali Tekeoglu, and Arash Habibi Lashkari. Pdf malware detection based on stacking learning. In *ICISSP*, pages 562–570, 2022.

[5] Rasoul Rezvani-Jalal, Morteza Zakeri-Nasrabadi, Amin Hasan-Zarei. Enhancing Malware Detection Reliability in Non-Executable Files Using Confidence Score Prediction, [Accessed: 2024].

[6] Monika Sharma and Ajay Kaul. A review of detecting malware in android devices based on machine learning techniques. *Expert Systems*, 41(1):e13482, 2024.