LangChain을 활용한 악성코드 자동 분석 프로세스 설계

도예진[1], 김형식[2]
[1]성균관대학교 전자전기컴퓨터공학과 대학원생
[2]성균관대학교 소프트웨어학과 교수

dyj001213@g.skku.edu, hyoung@skku.edu

Designing an Automated Malware Analysis Process Using LangChain

Ye-Jin Do[1], Hyoung-Shick Kim[2]
[1]Dept. of Electrical and Computer Engineering, Sungkyunkwan University
[2]Dept. of Computer Science and Engineering, Sungkyunkwan University

요 약

최근 악성코드의 복잡성과 다양성 증가로 인한 전통적인 수동 분석의 한계를 극복하고자 본 논문에서는 생성형 AI 기술을 활용한 자동화된 악성코드 분석 프레임워크를 제안한다. 본 연구는 LangChain 프레임워크를 사용하여 악성코드 행위를 분석하고 이해할 수 있는 자동화된 리포트 생성 방식을 개발하였다. 해당 프로세스는 LangChain agent 에 의해 자동화되어 분석가의 부담을 줄이고, 실수 가능성을 감소시키며, 빠르고 일관된 분석 결과를 제공한다. 이러한 자동화된 접근 방식은 악성코드 분석 과정의 효율성을 증가시키며 신속하고 정확한 대응을 가능하게 한다.

1. 서론

최근 악성코드 공격의 증가와 그 복잡성이 커지면서 악성코드 분석의 중요성이 더욱 강조되고 있다. 악성코드가 급속도로 진화하고 그 종류가 다양해지면서, 시간이 많이 걸리고, 자원을 많이 소모하며 높은 전문 지식을 요구하는 전통적인 수동 분석 방법만으로는 모든 위협을 효과적으로 대응하기 어려워졌다. 반면 자동화된 프로세스는 빠르고 일관된 분석을 가능하게 하며 분석가의 작업 부담을 덜어줄 수 있다[1]. 생성형 AI 기술의 발전으로 인해 LLM(Large Language Model)의 활용 및 도입이 확대되고 있는 상황에서 악성코드 분석 프로세스에 LLM의 적용은 분석의 정확도와 효율성을 높이는데 중요한 역할을 할 수 있다[2]. 본 논문에서는 악성코드 분석과 연관된 자연어 처리를 통해 복잡한 악성코드 행위를 분석하고 이해할 수 있는 LangChain 프레임워크를 이용한 LLM 애플리케이션 아키텍처를 활용하여 효율적인 악성코드 분석 리포트 생성을 위한 자동화 프레임워크를 제안한다.

본 논문의 순서는 다음과 같다. 2장에서는 제안하는 악성코드 분석 자동화 프레임워크와 동작 방법에 대해 설명하고, 3장에서는 생성된 리포트 결과에 대해서, 4장에서는 결론 및 향후 연구방향을 제시한다.

2. 악성코드 분석 자동화 프레임워크 설계

이 절에서는 본 연구에서 사용되는 악성코드 데이터셋과 제안하는 자동화 프레임워크의 설계와 동작 방식에 대해 설명한다.

본 논문에서 사용한 악성코드 데이터셋인 theZoo는 악성코드 연구 및 교육을 목적으로 하는 악성코드 아카이브로, 바이러스, 웜, 트로이 목마, 랜섬웨어 등 다양한 유형의 악성코드 샘플들을 포함하고 있다[3]. 본 연구에서 제안하는 악성코드 분석 자동화 프레임워크 동작 과정은 다음과 같다. 압축된 악성코드 ZIP 파일을 압축 해제하여 암호화된 악성코드 샘플들을 제공된 암호를 사용하여 해제한 후, 악성코드 분석 툴인 Virustotal과 Ghidra API를 활용하여 분석을 진행하고 나온 결과를 기반으로 악성코드 분석 리포트를 작성한다[4,5]. 이렇게 LangChain은 여러 작업을 자동으로 수행하고, 각 단계에서 발생한 결과를 활용하여 최종적으로는 악성코드 분석 리포트를 생성한다. 또한 이 프로세스는 LangChain 프

레임워크의 핵심 구성 요소 중 하나인 LangChain agent를 통해 자동화되어, 수동적인 개입없이 일련의 단계들이 순차적으로 진행되고 외부 시스템과의 통합을 지원하여 효율적으로 유연한 자동화 작업이 가능하도록 설계되었다. 이 자동화는 분석 과정의 효율성을 높이고, 인간 분석가의 실수 가능성을 줄이며, 빠르고 일관된 결과를 제공한다. 제안하는 프레임워크의 전체적인 동작과정은 그림 1과 같다.

(그림 1) 전체적인 흐름도

3. 결과

이 절에서는 본 논문에서 제안한 자동화 프레임워크를 통해 얻은 결과를 기반으로 한 악성코드 분석 리포트의 구체적인 내용과 형식에 대해 설명한다.

LLM을 활용하여 리포트를 생성할 때 활용한 프롬프트는 그림 2와 같다. 분석 리포트를 두 개의 주요 섹션으로 나누어 작성하도록 지시하며, 각 섹션에는 악성코드의 기본정보와 간단한 설명, 악의적인 기능에 대한 동작에 대한 세부 정보를 제공한다. 분석된 악성코드 함수의 동작 방식과 악성 행위에 대한 분석을 통해 작성된 리포트의 구체적인 내용과 형식을 그림 3과 그림 4를 통해 예시로 보여준다. 각 함수의 특성과 악성 행위에 대한 자세한 설명을 포함하고 있으며, 이를 통해 악성코드의 행위와 구체적인 내용을 다룬다. 이 리포트는 자동화된 분석 툴을 사용하여 얻은 결과를 바탕으로 하여, 각 함수의 특성과 악성 행위를 자세히 설명한다.

```
"""
        Analyze the following data extracted from malware file {filename}:
        {full_text}

        Create a comprehensive malware analysis report based on the text above.
        The report should be divided into two main sections:
        1. Malware Information: Summarize the name, type, behavior, SHA-256, and description of the malware.
        2. Behavior Information: Provide details about the behavior of the malicious functions of the malware along with the corresponding code excerpts.
        """
"""
```

(그림 2) Prompt 예시

```
Suspicious function: CreateFileA
Potential malicious behaviors detected: CreateFile
Decompiled code:

HANDLE __stdcall
CreateFileA(LPCSTR lpFileName,DWORD dwDesiredAccess,DWORD
    dwShareMode,
            LPSECURITY_ATTRIBUTES lpSecurityAttributes,DWORD
    dwCreationDisposition,
            DWORD dwFlagsAndAttributes,HANDLE hTemplateFile)

{
  HANDLE pvVar1;

                    /* WARNING: Could not recover jumptable at 0
    x00401168. Too many branches */
                    /* WARNING: Treating indirect jump as call */
  pvVar1 = CreateFileA(lpFileName,dwDesiredAccess,dwShareMode,
    lpSecurityAttributes,
                         dwCreationDisposition,dwFlagsAndAttributes,
    hTemplateFile);
  return pvVar1;
}
```

(그림 3) 악성코드 예시

As a malware analysis expert, I have analyzed the provided code snippet and identified several suspicious behaviors that could indicate potential malicious activity. Here are my findings:

1. Unknown Function Call: The function "CreateFileA" is called with an unknown number of arguments, which is unusual for a standard Windows API function. This raises suspicion and suggests that the function may be used for malicious purposes.
2. Indirect Jump Table: The decompiled code contains an indirect jump table at address 0x00401168, which cannot be fully recovered. This indicates that there might be additional code or data in the program that is not visible through static analysis.
3. Unused Variables: Two variables, "pvVar1" and "hTemplateFile", are declared but not used anywhere in the code. It's possible that these variables are used elsewhere in the program, but their presence here suggests that the code may be trying to hide something.
4. Lack of Error Handling: The function does not handle errors properly. For example, it does not check if the file already exists before attempting to create it, which could lead to unexpected behavior if the file already exists.
5. Inconsistent Argument Passing: The function takes multiple arguments of different types, including a string (lpFileName), a DWORD (dwDesiredAccess), a DWORD (dwShareMode), a SECURITY_ATTRIBUTES structure (lpSecurityAttributes), and a DWORD (dwCreationDisposition). While this is consistent with the standard Windows API function signature, it's worth noting that some of these arguments are not typically passed in the same way. For example, the SECURITY_ATTRIBUTES structure is usually passed as a pointer to a struct, rather than as a separate argument.
6. Unusual Memory Access: The function accesses memory at address 0x00401168 using an indirect jump table. This could indicate that the function is trying to execute malicious code or data.
7. Missing Return Statement: The function does not have a return statement, which means that its behavior is undefined. This could indicate that the function is intended to be infinite or that it will crash the program when executed.

In conclusion, based on the above observations, it appears that the code snippet exhibits suspicious behavior

(그림 4) 생성된 리포트 예시

4. 결론 및 향후 연구

본 논문에서는 LangChain을 활용하여 자동화된 악성코드 분석 프레임워크를 제안한다. 생성되는 분석 리포트는 악성코드의 식별과 이해를 돕는 중요한 도구로써 리포트에서 제공하는 자료와 상세한 설명은 분석가 및 보안 전문가가 악성코드의 위험을 신속하게 평가하고 효과적으로 대응할 수 있도록 지원한다. 이러한 체계적인 분석 접근 방식은 악성코드 분석 과정에서의 인적 오류를 줄이고, 처리 속도를 향상시키며 보다 정확하고 일관된 결과를 도출할 수 있도록 도와줄 뿐만 아니라, 악성코드 대응 전략의 개선에 기여할 수 있다.

본 연구의 프레임워크와 분석 방법에서 더욱 확장하여 향후 연구에서는 더 다양한 유형의 악성코드 샘플들을 기반으로 악성코드 패턴을 더 정확하게 예측하고, 새로운 악성코드 변형을 식별하도록 훈련시킬 계획이다. 또한, 생성된 리포트에서 YARA 규칙을 추출하고 이를 활용하여 악성코드 탐지 시스템의 성능을 평가하는 연구를 진행할 계획이며, 이를 통해 YARA 규칙의 효과성과 정확성을 검증하고, 자동화된 악성코드 분석 도구의 전반적인 성능을 향상시킬 수 있음을 기대한다. 이와 함께, 생성된 리포트의 품질과 유효성을 평가하기 위해 기존에 존재하는 분석 리포트와 비교 분석을 실시할 계획이다. 이 비

교 분석을 통해 자동화된 방식으로 생성된 리포트가 기존의 리포트와 어떻게 차별화되며, 어떤 장점을 제공하는지 명확히 할 수 있으며, 필요한 개선점을 도출해낼 수 있을 것이다. 이러한 접근은 자동화 프로세스의 정확성과 신뢰성을 강화하는데 기여할 것이다.

Acknowledgements

이 논문은 2024년도 정부(과학기술정보통신부)의 재원으로 정보통신기획평가원의 지원 (RS-2023-00229400, 안전한 메타버스 환경을 위한 사용자 인증 및 프라이버시 보호 기술 개발)과 2024년도 정부(과학기술정보통신부)의 재원으로 정보통신기획평가원의 지원을 받아 수행된 연구임 (No. 2022-0-01199, 융합보안핵심인재양성)(No.2023-2523, 휴대폰 단말에서의 보이스피싱 탐지, 예방 기술 개발)

참고문헌

[1] Yiheng Liu, Tianle Han, Siyuan Ma, Jiayue Zhang, Yuanyuan Yang, Jiaming Tian, Hao He, Antong Li, Mengshen He, Zhengliang Liu, Zihao Wu, Lin Zhao, Dajiang Zhu, Xiang Li, Ning Qiang, Dingang Shen, Tianming Liu, Bao Ge, “Summary of ChatGPT-Related research and perspective towards the future of large language models”, Meta-Radiology, Volume 1, Issue 2, 2023

[2] Nusrat Zahan, Philipp Burckhardt, Mikola Lysenko, Feross Aboukhadijeh, Laurie Williams, “Shifting the Lens: Detecting Malware in npm Ecosystem with Large Language Models”, arXiv preprint arXiv:2403.12196 (2024).

[3] theZoo, https://github.com/ytisf/theZoo , Last accessed : April 24, 2024

[4] VirusTotal API, https://www.virustotal.com/

[5] Ghidra API, https://ghidra-sre.org/