



BRNO UNIVERSITY OF TECHNOLOGY

VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ

FACULTY OF INFORMATION TECHNOLOGY

FAKULTA INFORMAČNÍCH TECHNOLOGIÍ

DEPARTMENT OF INFORMATION SYSTEMS

ÚSTAV INFORMAČNÍCH SYSTÉMŮ

ANALYSIS OF MALWARE BEHAVIOR USING LARGE LANGUAGE MODELS

ANALÝZA CHOVÁNÍ MALWARU POMOCÍ VELKÝCH JAZYKOVÝCH MODELŮ

MASTER'S THESIS

DIPLOMOVÁ PRÁCE

AUTHOR

AUTOR PRÁCE

Bc. VLASTIMIL RÁDSETOULAL

SUPERVISOR

VEDOUCÍ PRÁCE

Mgr. Ing. PAVEL OČENÁŠEK, Ph.D.

BRNO 2024

Abstract

This thesis investigates the use of large language models (LLMs) enhanced with Retrieval-Augmented Generation (RAG) techniques to analyze malware behaviors effectively. Starting with an overview of malware analysis methods, both static and dynamic, the study delves into the use of the MITRE ATT&CK framework to understand and categorize malware strategies.

The core of the research focuses on the architecture and implementation of a malware behavior analysis tool that integrates RAG with LLMs. This tool aims to aid security professionals leveraging generative AI's capabilities to interpret complex malware behaviors.

Additionally, the research includes a practical deployment of the Security Information and Events Management (SIEM) system, using the Wazuh platform to detect simulated adversarial behaviors. The deployment and testing are done in a controlled virtual environment, highlighting the potential of LLMs in enhancing cyber security measures.

The thesis concludes with recommendations for future enhancements and the potential expansion of generative AI applications in cyber security.

Abstrakt

Táto práca skúma využitie veľkých jazykových modelov (LLMs) vylepšených technikou Retrieval-Augmented Generation (RAG) pre efektívnu analýzu správania malvéru. Začína prehľadom metód analýzy malvéru, ako statických tak dynamických. Štúdia sa zameriava na využitie rámca MITRE ATT&CK na pochopenie správania malvéru veľkým jazykovým modelom.

Jadro výskumu sa zameriava na architektúru a implementáciu nástroja na analýzu správania malvéru, ktorý implementuje RAG s využitím LLMs. Tento nástroj má za cieľ pomôcť profesionálom v oblasti bezpečnosti využívať možnosti generatívnej AI na interpretáciu komplexného správania malvéru.

Okrem toho, výskum zahŕňa praktické nasadenie systému pre správu bezpečnostných informácií a udalostí (SIEM), pričom využíva platformu Wazuh na detekciu simulovaných útokov. Nasadenie a testovanie prebiehajú v kontrolovanom virtuálnom prostredí. Práca poukazuje na potenciál LLM modelov pri zlepšovaní opatrení v kybernetickej bezpečnosti.

Práca končí diskusiou o možných vylepšeniach implementovaného nástroja.

Keywords

large, language, models, malware, behavior, analysis, detection

Klíčová slova

velké, jazykové, modely, malware, analýza, chování, detekce

Reference

RÁDSETOULAL, Vlastimil. *Analysis of Malware Behavior using Large Language Models*. Brno, 2024. Master's thesis. Brno University of Technology, Faculty of Information Technology. Supervisor Mgr. Ing. Pavel Očenášek, Ph.D.

Analysis of Malware Behavior using Large Language Models

Declaration

I hereby declare that this Master's thesis was prepared as an original work by the author under the supervision of Mgr. Ing. Pavel Očenášek, Ph.D. The supplementary information was provided by Ing. Petr Chmelář, Ph.D. I have listed all the literary sources, publications and other sources, which were used during the preparation of this thesis.

.....
Vlastimil Rádsetoulal
May 17, 2024

Acknowledgements

I would like to express my sincere appreciation for the help of my supervisors Mgr. Ing. Pavel Očenášek, Ph.D., for the initial consultations and supportive environment, and Ing. Peter Chmelář, Ph.D. for his always optimistic attitude during online consultations and all the information and professional help provided.

Contents

1	Introduction	5
2	Malware Analysis	6
2.1	Static Malware Analysis	6
2.2	Dynamic Malware Analysis	7
2.3	MITRE ATT&CK Framework	7
3	Host Based Intrusion Detection Systems	9
3.1	Intrusion Detection and Prevention	9
3.2	HIDS and NIDS Comparison	9
3.2.1	Network-Based Intrusion Detection Systems	10
3.2.2	Host-Based Intrusion Detection	11
3.2.3	File Integrity Check	11
3.2.4	Registry Monitoring	11
3.2.5	Rootkit Detection	11
3.2.6	Data Outputs of HIDS	12
3.3	OSSEC and Wazuh Comparison	12
3.4	Wazuh Host-Based Intrusion Detection System	12
3.4.1	Architecture and Components of Wazuh	12
3.4.2	Wazuh for Malware Detection	13
3.4.3	Integration Methods and Options for Wazuh	13
4	Large Language Models	14
4.1	Large Language Models and Generative Pre-trained Transformers	14
4.2	Large Language Models for Malware Behavior Analysis	15
4.2.1	Generative AI in Cyber Security - General Overview	16
4.2.2	Malware Behavior Analysis with LLMs	16
5	Architecture of the Malware Behavior Analysis Tool.	19
5.1	The Requirements for Malware Analysis Tool using Large Language Models	19
5.2	High Level Overview of The Architecture	20
5.3	Retrieval-Augmented Generation (RAG)	21
5.3.1	Retrieval-Augmented Generation Overview	21
5.4	The Final Design of Adversarial Behavior Analysis Tool using RAG with LLM	23
5.4.1	Chosen Technologies	23
5.4.2	Implementation Design of RAG with LLM for Adversarial Behavior Analysis	25

6	SIEM Deployment and Attack Simulation	27
6.1	Wazuh Deployment and Configuration in Virtual Environment	27
6.1.1	Wazuh Open Virtual Appliance	27
6.1.2	Deploying Wazuh Agent on Windows 10 Endpoint	28
6.1.3	Wazuh Server and Agent Configuration	29
6.2	Simulation of Adversarial Behavior	31
6.2.1	Manual Testing Using Kali Virtual Machine	32
6.2.2	Testing with Atomic Red Team Framework	32
6.2.3	Exploiting the CVE-2023-23397	34
7	Implementation and Experiments	38
7.1	Implementing Retrieval-Augmented Generation with LLM	38
7.1.1	Dataset Preparation and Vectorstores Database Creation	39
7.1.2	Alerts Collection	39
7.1.3	Alerts Processing	41
7.1.4	Alerts Analysis	42
7.2	Experiments	42
7.2.1	Experiments using OllamaEmbeddings	42
7.2.2	Experiments using OpenAIEmbeddings	44
7.2.3	Enhanced Information and CVE-2023-23397	50
7.3	Possible Improvements to the Implementation	51
8	Conclusion	52
	Bibliography	53

List of Figures

4.1	The schematic structure presenting the position of Generative AI as a subset of Machine Learning (ML) and its overlap with different technologies such as Deep Learning (DL), Natural Language Processing (NLP) and Computer Vision (CV).	15
5.1	High level architecture diagram. The left part of the diagram with blue color consists of Wazuh SIEM solution and its components. Wazuh Agent sends logging data to Wazuh Server, which relays this data to Indexer for creating indices. Once indexed, the data is forwarded by server to a data collection point. On the right section with yellow color is Analysis Tool. The data is processed by Data Processor and passed to Analysis Engine including the model to be queried. Output is then provided to user.	21
5.2	The figure shows the data ingestion model for Retrieval-Augmented Generation.	22
5.3	The figure shows the flow of the Retrieval-Augmented Generation technique used with large language model (LLM) (retrieved from [10]).	23
5.4	The figure shows the performance of Mistral 7B compared to Llama models. The benchmarks performed are MMLU: Massive Multitask Language Understanding, Knowledge, Reasoning, Comprehension, AGI Eval: Artificial General Intelligence Evaluation, Math, BBH: Beyond the Benchmark Hurdle, Code. (retrieved from [9]).	24
5.5	The figure shows the design of the architecture and the interaction between the components. The alerts saving and data enrichment is omitted for simplicity.	26
6.1	The figure shows Windows operating system versions throughout January 2018 to July 2023. Purple line: Windows 10, light blue line: Windows 11, yellow line: Windows 7. The rest is Win8, Win8.1 and WinXP. Adopted from [24].	28
6.2	The Constant Database (CDB) list of common ports displayed on Wazuh dashboard. The first 10 ports are displayed.	30
6.3	The figure shows an increase of Wazuh alerts during one of the testing sessions. First half of the timeline represents no malicious activity and the spike of the alerts begins at the time of the test beginning. Some of the rule levels were omitted for the clarity.	34
6.4	The figure shows an OutlookSpy plugin window with commands to set the parameters required to exploit the vulnerability. The appointment item can be seen in the background.	36

6.5	The figure shows the terminal window of Kali attacker machine after successful exploit of CVE-2023-23397. It contains user information and leaked NTLM hashes.	37
-----	--	----

Chapter 1

Introduction

Information technologies are becoming increasingly interwoven with our everyday work duties as well as free time. The crime is part of our society since time out of mind and electronic environment is not an exception. Therefore, it is as important to secure our digital world as it is to have a safe lock on our home door.

An undisputed part of computer criminal's arsenal is malicious software, which is ever evolving. This creates a challenging task for improving defense mechanisms to prevent any possible financial or emotional damage and data loss.

A widely used category of defense mechanisms are Intrusion Detection Systems. Specifically Host-Based Intrusion Detection Systems are of the interest in this thesis. These systems provide comprehensive way of analyzing the impacts of malware on the endpoint devices. However, in constantly changing environment of cyber security, it is a necessity to possess an even more efficient ways of combating malware.

This thesis aims to explore the possibilities of utilizing a current state of the art technology in never ending battle with malware. This technology is Large Language Models, which has exploded in popularity in the recent years. Large Language Models have a seemingly unlimited range of possible applications in numerous areas. A successful employment of such technology requires thorough review of the current solutions and analysis of their feasible improvements or additions.

One of the ways to engage Large Language Models into Host-Based Intrusion Detection Systems could be to generate human readable insights about malware activities on the host. These insights would be of tremendous help to overwhelmed information security analysts trying to "decipher" system logs. With such insights the analysts would be able to effectively prioritize their focus on important events, alerts and detections. However, such insights are not the only feature that large language models could offer. It is possible that these models would be a great addition to decision making at incident response. If a sufficient reliability of such implementation is proven, it could be used for automated intrusion reaction.

The thesis will cover analysis of requirements for such tool or integration, its architecture and implementation details. A summary of the results and conclusion will be conducted at the end of the thesis.

Chapter 2

Malware Analysis

This chapter covers an overview of malware analysis fundamentals and reviews existing approaches to malware detection and analysis. Furthermore it describes methodologies of behavioral analysis.

Malware analysis is a vital part of computer security activities as it serves a crucial role in developing effective defense mechanisms. From business perspective, when a security incident occurs, cyber security analysts and incident response team members need to understand the nature of the threat and how it affects system resources and business continuity. It helps them more efficiently identify and measure damages.

Malware often targets sensitive data, such as personal information, financial records, or intellectual property. Effective malware analysis helps to identify, whether such data is targeted by specific malware or not.

Malware analysis can help trace the threat actor behind an attack by distinguishing similarities in malware's methodologies and techniques.

In the context of this paper the most important purpose of malware analysis is understanding the behavior of malware. It is essential for developing intrusion detection systems, that are capable of real time malware detection.

2.1 Static Malware Analysis

Static analysis is a malware analysis method without executing the analyzed program. It involves review of either source code or binary code of the analyzed software. Source code of a software is usually not available, therefore it requires reverse engineering techniques including disassembly. Part of static analysis is extraction of metadata from the program. The extracted metadata such as version numbers or embedded information, could help to gain insights into the software's origin or purpose.

The binary of the analyzed program can contain useful indicators of compromise (IOC) in form of strings. The strings found in program's binary could include:

- IP addresses to connect to.
- Domains or URL addresses to download additional malicious software.
- Scripts in different programming language to be dropped and executed on the machine.
- Other strings found in known malware.

However, static analysis alone is nowadays insufficient. Common static analysis techniques can be easily deceived by various techniques such as code packing, anti debugging features, instruction virtualization, control-flow obfuscation and others. One of the advanced techniques is anti-disassembly technique. It uses specially crafted code or data in order to let disassembly tools to produce validly appearing, however incorrect program listing [20].

2.2 Dynamic Malware Analysis

Dynamic malware analysis is a methodology performed during the runtime of the analyzed program. The execution of the program is done in controlled environment, such as a sandbox or virtual machine, that is restricted from connection to outside world.

There are two approaches to dynamic analysis:

- Capturing the system state image before malware execution to explore how it changes after the malware is executed.
- Executing the malware and observing its actions and behavior directly during the runtime.

The first approach only captures the end result of malware execution. It does not capture all the dynamic changes, in granular enough level, which occur during the execution of the analyzed malware. For this approach there are different system snapshot taking tools. These tools are capable of recording changes to file system such as the files created, downloaded or deleted before its termination. Some of the tools can create more complex picture of the hosting system, such as directory structure, file properties, registries, and other system files.

The second approach, where the program and its associated processes are observed during the runtime, provides more in-depth behavior properties. In this methodology, the analyst monitors interaction with the file system, the registry, and the network using various monitoring tools. These interactions with the file system, registries, and the network can be monitored using tools such as Process Monitor, Process Explorer, TCPView, and Wireshark. Process Explorer and Process Monitor are used to monitor file system and registry changes. TCPView and Wireshark network activity monitoring tools. The result of such monitoring is the behavioral analysis of the whole system, rather than the behavior of a single malicious program. The ordinary, expected background activities, which are not related to the malware process, have to be ruled out of the results of the behavioral analysis [32].

2.3 MITRE ATT&CK Framework

The MITRE ATT&CK framework is a comprehensive knowledge base that consists of detailed tactics, techniques, and procedures (TTPs) used by cyber attackers based on real-world observations. It stands for Adversarial Tactics, Techniques, and Common Knowledge. This framework helps cybersecurity professionals understand how attackers operate and provides a common language to describe these activities [6].

This framework is undoubtedly an important tool for implementing security solutions to better and more efficiently defend against various threats. As the objective of this work is to design and implement a solution that is capable of describing malware behavior using large

language models, MITRE framework will be very helpful in providing context information to the large language model. The following is a brief overview of its core components:

- **Tactics** are the high-level goals that attackers aim to achieve during a cyber attack. It explains the „why“ of the attackers action and aims to describe, what an attacker is trying to accomplish. Examples include initial access to a network, execution of malicious code, credential access [6].
- **Techniques** are the specific methods used by attackers to achieve their goals. The techniques explain a way, in which attackers try to accomplish their tactical goals. These include reconnaissance, Initial Access, Persistence, Privilege Escalation and more. Techniques then consist of several sub-techniques [6].
- **Sub-techniques** are more specific lower level descriptions of adversarial behavior. These include terms such as Startup Items, Windows Service, Bypass User Account Control or Credential Stuffing [6].
- **Procedures** refer to the specific implementations or real-world methods employed by adversaries for executing techniques or sub-techniques [6].

The notation of MITRE framework techniques and sub-techniques is quite simple. The technique code name begins with letter 'T' followed by four digit number, for example T1557 which would refer to „Adversary-in-the-Middle,“ followed by dot separating sub-technique's numeric identifier T1557.002 which refers to „ARP Cache Poisoning [6].“

The MITRE Corporation also provides a list of software that is used during the attacks, that is used in this work alongside the techniques to provide a contextual information to the model.

Chapter 3

Host Based Intrusion Detection Systems

This chapter explores host-based intrusion detection systems (HIDS). Initially, the intrusion detection and prevention concept is explained, followed by a comparison of host-based intrusion detection systems with network-based intrusion detection systems (NIDS). The most widely used open-source implementations of HIDS are introduced and evaluated. Furthermore, an examination of the data retrieved from these systems is conducted to select relevant and usable information for analysis with large language models.

3.1 Intrusion Detection and Prevention

An intrusion occurs when an adversary attempts to gain entry into information system or to disrupt its normal operations. The motivation of such action is usually to do harm. Such attacks could be self-propagating, as in the case of viruses and distributed denial-of-service attacks, nonetheless they are still initiated with the purpose to harm targeted organization.

The series of activities including information security policies, installing network firewalls and intrusion detection systems, and effective and measurable employee training is called intrusion prevention.

Intrusion detection is set of procedures and systems capable of identifying system intrusion. A successful intrusion detection should lead to intrusion reaction, which defines actions necessary to be taken by company or individuals when an intrusion occurs. These actions aim to soften the impact of an intrusion, to enumerate exposed data and to return the system to its usual operational standard.

Intrusion correction activities, strive to find the source of the attack, tactics and techniques used to perform it, in order to implement countermeasure to prevent such attacks from happening in the future [31].

3.2 HIDS and NIDS Comparison

This section contains overview of network-based intrusion detection systems in comparison with host-based intrusion detection systems. It is important to establish the crucial differences and advantages of HIDS over NIDS, for better comprehension of the chosen architecture in chapter 5.

3.2.1 Network-Based Intrusion Detection Systems

The Network-Based Intrusion Detection System (NIDS) is deployed to monitor network traffic using a network interface card (NIC) directly connected into network. For an attack to be detected a signature or pattern is used to match specific events. If the traffic seen on the network matches one or more pre-defined IDS signatures, an alert is generated. The details of an alert are usually stored in form of logs. It is possible to tie actions with such alerts, which could include notifying a person responsible for the system or even adding a firewall rule to block the future traffic [8].

Although NIDS is useful, there are certain limitations to its usage:

- The communications between two endpoints can be encrypted, which disables the readability of the packets.
- Packets sent by adversaries could be encrypted and signatures built to detect them could be unsuccessful in catching them.
- Various evasion techniques exist to bypass NIDS.

Plenty of evasion techniques exist to bypass, or sidestep NIDS solutions. Most network intrusion detection systems today have mechanisms of mitigating such techniques, for example by reassembling the traffic contents in their memory. This method can potentially exhaust system resources, in case of improper tuning. Some of the common NIDS evasion techniques are:

- **String matching** weaknesses, which are result of poorly constructed signatures. This applies to NIDS, that are signature-based. This weakness occurs, when the adversary is able to get to know, either publicly available signature, or gets hands on customized signature. In case the signature is incorrectly addressing the information indicating the attack, the attacker can hide the attack with adjustments to not trigger the signature [8].
- **Session splicing** utilizes the session splicing mechanism of data by TCP. The data stream is not reassembled until it reaches its final destination. Adversary can split the attack data into multiple packets that contain payload in pieces. This splicing could cause signature not to trigger as it does not match the pattern. Some network intrusion detection systems reassemble the TCP stream in order to detect such evasion. However, this technique increases the processing load of the respective NIDS [8].
- **Fragmentation attacks** are more advanced than session splicing attacks. These include fragmentation overlap attacks and fragmentation time-out attacks. The first works on instructing the host to reassemble the packets and to overwrite or overlap some of the received packets at specific offsets. Fragmentation time-out attacks exploit the NIDS fragmentation timers flushing the reassembly caches after time-out.
- **Denial of Service** technique simply overloads the NIDS in order for it to crash [8].

It is apparent that NIDS have its limitations that need to be addresses at the final destination of the inspected data. This introduces the need for host-based intrusion detection systems.

3.2.2 Host-Based Intrusion Detection

A host-based intrusion detection system is installed directly on the endpoint host, whether it is a server or workstation. It detects events and generates alerts similar to an NIDS. The main difference is that HIDS can inspect full communication stream. The reassembled data received by operating system are accessible by HIDS installed on the system. Due to this insight, the previously discussed NIDS evasion techniques do not work for HIDS. Additionally, HIDS is capable of system level checks such as log analysis, registry monitoring, file integrity checking, rootkit detection, and active response [8].

It is important to review HIDS capabilities to comprehend the ways of utilizing HIDS as a source of the data for analysis of malware behavior. The aspect of active response is not necessary for the scope of the needs of this paper.

3.2.3 File Integrity Check

It is possible to generate a unique digital fingerprint out of every file on an operating system. This fingerprint is computed as cryptographic hash. It is generated based on the content and the name of the file using various mathematical operations. One of the capabilities of HIDS is to check the integrity of the files stored on the file system based on the hash comparison [8]. This provides the ability to detect alteration or whole replacement of files by the analyzed malware. It is an important aspect to monitor as the file modification is an often effect of malware activities. The highest volume of file modifications could be attributed to malware category ransomware. These large quantities of file replacement could indicate that the file system is being compromised and that the files are being replaced by their encrypted versions.

3.2.4 Registry Monitoring

Another important operating system's component, that needs to be closely monitored is the system registry. It is a directory listing of all hardware and software settings, operating system configurations. Additionally, on Microsoft Windows, these include users settings, users, groups and other preferences.

It is common for malware to make changes to the system registry keys. An HIDS monitor for such changes as they might be an indication of establishing a persistence within a system. The registry key changes are also a subject for analysis to reveal true capabilities of newly installed software. Signatures of an HIDS should be able to distinguish patterns of suspicious registry changes leading to successful intrusion detection. A malware can often change registries of existing installations of legitimate programs in order to exploit them for its needs or to replace them with malicious copy. The registry also provide a view at how the system kernel interacts with hardware and software [8].

3.2.5 Rootkit Detection

Rootkits modify the internal functionality of the operating system in order to take over them. The modifications include hiding files, processes, network connections, ports, services and other resources, making them invisible by other running programs. This makes discovering malicious activity by security software or personnel difficult [20].

There are several types of rootkits:

- **A firmware** rootkit is installed with firmware. This type of rootkit is difficult to detect as for its detection, one has to inspect their compiled installation package prior to installing it on endpoints or network devices.
- **A virtualized** rootkit is installed to intercept system calls between the system hardware and its operating system. It is loaded at boot time and is eligible of altering the interactions with operating system in the favor of the rootkit.
- **A kernel level** rootkit is usually deployed through loadable kernel modules or hardware drivers. These rootkits are difficult to detect as they advertise themselves to the operating system as ordinary software components.
- **A library level** rootkit utilizes system calls to hide any information about the adversary from the system.
- **An application** level rootkit being one of the most common types of rootkits. It replaces a known, legitimate application binary with its malicious copy. This copy then serves slightly different purpose than intended [8].

3.2.6 Data Outputs of HIDS

In this part there will be provided a description of any other data that can HIDS provide as an output. In combination with the previously described, monitored data, it will provide a possibility for training data for large language models.

3.3 OSSEC and Wazuh Comparison

In this section there is a brief comparison between two popular HIDS open source solutions. After the evaluation of the advantages and disadvantages of each other, for the purposes of this work, there will be a brief explanatory of the chosen option.

3.4 Wazuh Host-Based Intrusion Detection System

The chosen Host-Based Intrusion Detection system, used in this work, is Wazuh, which is an Extended Detection and Response (XDR) platform. It is an open source platform providing XDR and Security Information and Event Management (SIEM) features [26].

The features relevant to this work include log data analysis, intrusion and malware detection and file integrity monitoring.

3.4.1 Architecture and Components of Wazuh

The Wazuh architecture is based on three main components, the Wazuh server, the Wazuh indexer, and the Wazuh dashboard. The Wazuh agent is then deployed on the monitored endpoints and forwards security data to a central server [26].

Wazuh supports data from agent-less devices such as firewalls, routers, switches, and access points. The log data from such sources can be submitted via Syslog, SSH, or using their API. The central server's role is to decode and analyze the information received from raw log sources, and to forward the information to the indexer for indexing and storage purposes. The Wazuh indexer is then comprised of cluster of nodes communicating with

each other. However, for the purpose of this work, only a single-node cluster would be required as small Wazuh deployments do not require processing of large amount of data [25].

3.4.2 Wazuh for Malware Detection

Wazuh does not rely solely on signature based approach. It has threat detection rules that enable behavior-based malware detection. Wazuh aims to monitor and analyze anomalous behavior within the system environment exhibited by malware. As an effect it allows for detection of previously unknown threats. Wazuh comes with out-of-the-box rule sets. However, these rules can be customized and new rules can be created. According to Wazuh documentation, Wazuh decodes and organizes logs from monitored endpoints into fields. These can be used to create custom rules [27].

The predefined rule set is useful for detecting known malware, when there are indicators of compromise (IOCs) that connect related malware activities. Wazuh provides a rule description in form of easily digestible text, in its Security Events module. Many of the rules are tightly connected to specific behavior of specific malware and are usually created after the malware has been discovered. The conceptual idea of a malware analysis agent using large language model, could allow for directing security analysts in such discoveries and rule creation.

3.4.3 Integration Methods and Options for Wazuh

An important part of building the malware analysis agent is to efficiently retrieve the security data that Wazuh agent collects from monitored endpoints. Wazuh server analyzes this data and passes them to Wazuh indexer for indexation. The Wazuh indexer integration forwards analyzed security data, using Logstash as a data forwarder to forward the alerts to other security platforms [29]. In the case of this thesis the receiving end would be at first training dataset collector, that would require a Logstash installation on dedicated server or building an indexer. Once the sufficient model is trained the same forwarding could be used for real time malware analysis. This could lead to increased resources consumption and would be an insufficient way for the purposes of this paper's objective of implementation.

Another option for Wazuh integration is an integration of Wazuh Server, which reads out Wazuh's alerts file, forwarding it using data forwarders. The Wazuh server integration requires that a data forwarder has to be installed on the same system as the Wazuh server. Choosing this option, the forwarder executes periodic queries to the Wazuh indexer. The forwarder queries are performed in blocks with the set time range [29]. The delay of receiving the data that could be caused by the periodic querying is not a concern for the purposes of the data collection for future training. However, once the malware analysis agent is to be built either integrated, or running on its own, such delays should be optimized.

Based on the recommendations for integration option and data forwarder selection in [29], the most promising option is Elastic Logstash. It is a free, open server-side data processing pipeline that is capable of ingesting data from multiple sources. Logstash transforms the data and then sends it to the next destination. Logstash scheduling frequency is in intervals of up to one second.

Chapter 4

Large Language Models

Large Language Models (LLMs) are likely one of the most adopted technological advancements witnessed by modern population during the past decades. It has exploded in popularity in recent years due to success of ChatGPT a Generative Pre-trained Transformers (GPT) developed by research organization OpenAI. It has been adopted by many professionals to aid their daily tasks that require human-like content. One of the reasons of this success could be the fact that its usage is simple in nature. Just provide the model with a task or question and it is able to generate response in almost any subject. It is also capable of producing a programming language outputs and can be useful for debugging of the code. There is also a multitude of students using this product to help them fulfill their assignments, which is rightfully concerning to their pedagogues.

On the other hand, such a powerful tool can be leveraged to implement various industry specific tools tailored for individual problems. Cyber security is not an exception. The following sections will discuss Generative Pre-trained Transformers and Large Language Models, including existing solutions in the battle against adversaries and malware. Following is the section introducing challenges of using these models and the section describing their usage in this work.

4.1 Large Language Models and Generative Pre-trained Transformers

This section briefly describes what the terms Generative Artificial Intelligence (Generative AI), Large Language Models (LLM) and Generative Pre-trained Transformers (GPT) represent.

Generative Artificial Intelligence is a subset of Artificial Intelligence and Machine Learning, which combines Deep Learning Strategies, Natural Language Processing and even Computer Vision. It shifts its focus on data production rather than conventional tasks of regression or classification. Generative AI is capable of generating data resembling real data produced by humans by utilizing multiple machine learning techniques [19]. The figure 4.1 presents a schematic structure of generative AI and its technologies (adopted from [19]).

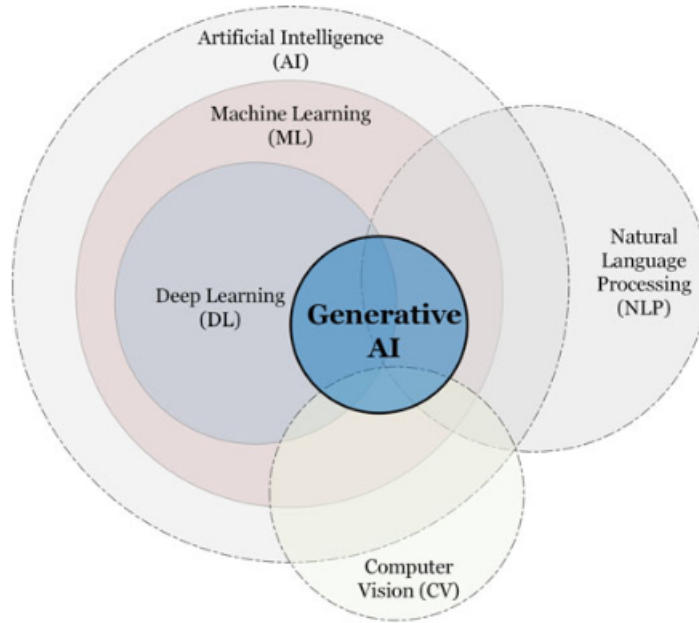


Figure 4.1: The schematic structure presenting the position of Generative AI as a subset of Machine Learning (ML) and its overlap with different technologies such as Deep Learning (DL), Natural Language Processing (NLP) and Computer Vision (CV).

Large Language Models are part of Generative AI, focused on natural language processing (NLP), that uses large scale language models for tasks including text generation, summarizing, completion, translation and more. It is trained on large datasets with large number of parameters. This translates to millions or even billions of parameters with the GPT-3 having 175 billions of parameters [19].

While the above are more broader terms, the **Generative Pre-Trained Transformers** is a family of Generative AI models, designed for natural language understanding and generating tasks, capable of learning language patterns, semantics, syntax and to concept large amount of general knowledge [19].

Especially the capability of these models to comprehend large amount of knowledge base of real world and to process text inputs, including code, make them promising candidate for use in Cyber security, namely in the malware analysis and adversarial behavior analysis. These models can potentially be used to generate descriptions of adversarial behavior in organization's environment and on end-user's endpoints, by taking security logs, endpoint detection details and more. Then by analyzing them with the aid of given context describing the situation preceding security alerts. Such models could also provide remediation steps for each detection or security alert, tailored to acknowledge organization's procedures and policies.

4.2 Large Language Models for Malware Behavior Analysis

This section discusses the options to leverage large language models in cyber security and in attempts to describe adversarial behavior and malware analysis.

4.2.1 Generative AI in Cyber Security - General Overview

Staying ahead of malicious actors and emerging threats poses an ongoing challenge. Traditional cyber security methods, that consist of static rules and indicators, struggle to keep pace with the dynamic nature of cyber attacks. However, the emergence of generative AI presents a appealing solution. By utilizing the power of generative AI, organizations can more effectively manage risks, mitigate the impact of cyber threats, and increase the security of their digital assets [19]. There are multiple ways of achieving this:

- **Threat Simulation and Prediction:** Generative AI models can simulate realistic attack patterns, aiding in the training and testing of defense systems. Additionally, these models can predict vulnerabilities and potential attack vectors, by analyzing large datasets and discovering trends, enabling proactive defense strategies. Thanks to data augmentation capabilities of this models, security researchers can get access to diverse and realistic datasets for training machine learning models [19]. The simulation of newly generated realistic attacks, could potentially lessen the number of zero-day attacks as these could be explored sooner. However, the same way the LLMs can be used by security researchers, they can be used by adversaries.
- **Enhancing Anomaly Detection** with using Generative AI models for developing models of usual system behavior, with additional data generated from learned patterns. This approach enhances anomaly detection efficiency, as these models excel at pattern recognition [19].
- **Reducing False Positives:** Even with endless tuning of traditional cyber security solutions, there is always large number of false positive alerts. These alerts shift the focus of security analysts from alerts that are true positive and have to be dealt with. By increasing the accuracy of threat detection algorithms, generative AI can minimize false positives, allowing security professionals to focus on genuine threats.
- **Automated Incident Response:** The generative AI models can greatly impact the mitigation time of incident response by assessing the scope and severity of security issues. The models can provide recommendations for mitigation or even automate some of the procedures [19].
- **Red Team Activities:** The generative AI model can be useful in creation of new automated tests for red teams to test the smaller organizations relying on traditional tools. These tests could be tailored specifically for the currently tested environment and relieve penetration testers from long preparations, which precede the testing. One of the domains of testing, that could generative AI models prove useful, would be phishing attacks. The fine tuned versions of models could learn to mimic employee's colleague's writing style, even chatting with the victim on prolonged basis, using social engineering techniques.

This was a summary of some of the potential use cases of Generative AI in cyber security. We will discuss how could the malware or adversary behavior analysis be aided by this models in the following sections.

4.2.2 Malware Behavior Analysis with LLMs

There are several options to take into consideration when using large language models for malware behavior analysis.

One of the options could be to create a sandbox environment for dynamically executing malware sample in secured environment. With the help of traditional malware analysis tools, the pre-trained large language model could be given the processed output of these tools to summarize the activity in a comprehensive description. Such tool could be used to aid malware analyst to better understand the capabilities of analyzed malware. During the malware analysis, it often happens that malware analysts face highly obfuscated portions of scripts. Sometimes it is hard to pinpoint every clue exhibited by the analyzed software and connect the dots. The ability of Generative AI to generate human like written text and recognize patterns would set it for being able to describe the goals of the adversary that written the malicious software. The malware analysis is highly time consuming process and requires deep technical knowledge. This process could be made more efficient with the aid of large language model.

The second approach would be to use generative AI for analysis of alerts and detections in real time. Security analysts are often overwhelmed by the large amount of alerts triggered by traditional tools. They have to usually manually review each of them, while not being able to distinguish between true positive alerts and false positive alerts at first glance. By the time that an analyst reaches the conclusion, there could already be several new alerts triggered. With the help of generative AI describing the spotted behavior in digestible form, the analysts would be better able to prioritize, which alerts to take on first.

Such tool would most likely have to be integrated into one of the existing solutions, to not introduce yet another source to monitor. The best suit would be SIEM system, which already possesses access to various security logs, application logs, firewall logs and logs from endpoint detection systems. All this information could be aggregated and fed to the analysis tool. The tool could also be given some additional information that would further enhance its descriptive capabilities:

- Organization knowledge base, which would help the model to put the detection into context. This could include assets information to understand the environment and points of interest. The incident response procedures would help to provide the analyst a proper steps for escalation.
- Common Vulnerabilities and Exposures (CVE) list for the model to be able to recognize whether the triggering application is known to be vulnerable and is potentially being exploit. This is indeed a job of threat intelligence team, however with the amount of vulnerabilities disclosed each year it is hard to keep up. This information could be fed to the model automatically from a trusted source.
- MITRE framework techniques could be used to help the model to understand various ways of how attackers operate and compare it with the behavior within the system.
- Malware databases information could be introduced to the model. There is plenty of available information that could be fetched about existing malware samples. Model could compare this information with the detection details and possibly query for indicators of compromise provided in such database. This would be an effective way for threat hunting. The model could either provide an analyst with the steps to take in order effectively search an environment for possible artifacts left by the malware or even perform the steps itself.

The complexity of such a tool would require a significant amount of work and fine-tuning, but it would be of tremendous value in enhancing the efficiency and effectiveness of cyber security operations.

Chapter 5

Architecture of the Malware Behavior Analysis Tool.

This chapter consists of requirements analysis, the high level architecture of the system and the architecture of the analysis module for malware behavior analysis tool using large language model. It describes the technologies required, and chosen for its implementation.

5.1 The Requirements for Malware Analysis Tool using Large Language Models

This section consists of the analysis of the requirements for the tool capable of analysis of the malware behavior. The points are not solely focused on the requirements for the final product, rather than an ideal scenario. It is also important to take into consideration the resources required for training or fine tuning a sufficient and precise model. The requirements analysis for a tool capable of classification of malware behavior using Large Language Models involves several key aspects:

- **Performance and Scalability:** The system should be able to efficiently process large amounts of data, including real-time classification of malware behavior. In the case of usage within environment of security operations center monitoring, the implemented system should be able to address the malware activities that generate a large amount of system events and changes. The one aspect that needs to be taken into consideration is whether the system is to be agent based, installed on each endpoint, consuming system resources of workstations, or whether it is meant to be centralized with dedicated computing power.
- **Classification Accuracy:** The system should achieve high levels of accuracy in classifying malware behavior. This includes the ability to distinguish between different types of malware while minimizing false positives and false negatives. The objective of such implementation is to ease the manual evaluation of malware incidents or malware analysis and should be accurate enough in order to not create yet another demanding task for security analysts or malware analysts.
- **Flexibility and Adaptability:** The system should have the ability to adapt to new malware variants, previously unseen in the “wild” and to respond rapidly to

new emerging threats. As the system aims to improve the current state of the malware behavior analysis, it should be capable of recognizing and describing patterns in malware behavior and to output useful information in textual form. Flexibility also involves working with various types of input data. This might be a demanding task as malware activities affect various system variables, leading into possible training data diversity in the case of fine tuning of the model.

- **Integration with HIDS and Other Security Tools:** The system should be able to be integrated into HIDS or other security tools, such as security and information event management systems for their enrichment. This creates a demand for the system to have an interface capable of effective communication with other tools.
- **Processing of Textual Data:** Given the use of Large Language Models, the system should effectively process textual data related to malware behavior. This creates a requirement for the system to effectively transform the input data and parse it to the Large Language Models in comprehensive form. This might include analyzing the descriptions provided by HIDS serving as a data source, raw logs and other data.
- **Selection of Suitable Pre-Trained Model** This thesis is a single person project with limited resources. It is impossible for an individual to collect tremendous data required to train Large Language Model from scratch. This creates a need for proper pre-trained model selection and testing of possible variants in order to find the most suitable option. It is a crucial step in the design of such system.
- **User Interface:** In terms of user interface, the question is whether the final system architecture will be as a standalone unit or if it will serve as an integration module. In the second case, the user interface would be covered by the system that the module would be integrated into. However, one important aspect of usability of the software by users is the capability of such implementation to generate understandable reports on malware detection and classification.

5.2 High Level Overview of The Architecture

The high level overview of the architecture of designed tool can be seen at figure 5.1.

The malware behavior analysis tool periodically queries Wazuh for data during the runtime of analyzed program or during constant host monitoring. Wazuh forwards the data using Data Forwarder and the tool receives it via its Data Collector. For this purposes the technology used will be Logstash as it is the best suiting data forwarder for Wazuh integration.

Data processor of the malware behavior analysis tool pre-processes the data received by data forwarder in order to pass it to Behavioral Analysis Engine. The behavioral analysis engine uses pre-trained large language model to analyze the important features from the received data and describes the behavior of the program that has either triggered a Wazuh alert or is being manually analyzed. It also produces textual report about the characteristics of malware and outputs it to user interface. It is possible to integrate the tool into Wazuh and use the output of the tool as a data enrichment on Wazuh's dashboard.

At user interface, user can manage the results of the analysis, by viewing them, rating them or managing countermeasures. The countermeasure will then be reflected on Wazuh Servers in form of new prevention signatures. The feedback of the user in form of False

Positive or True Positive verdict is then sent to data processor for the record. These data can be then used for re-training purposes. It is worth mentioning that such verdicts would be accessible to more experienced system users.

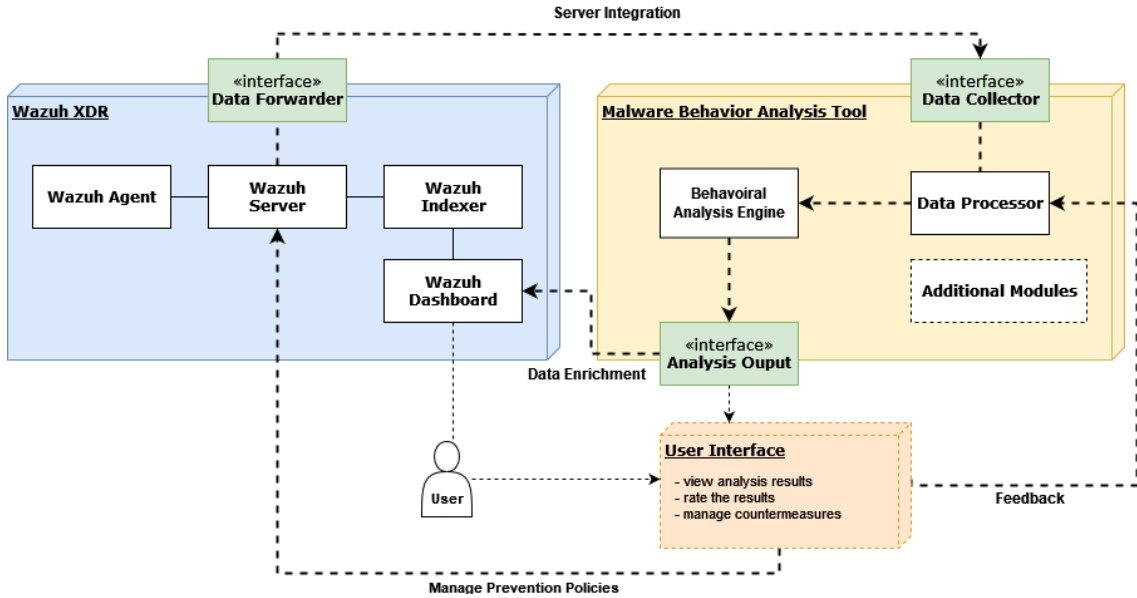


Figure 5.1: High level architecture diagram. The left part of the diagram with blue color consists of Wazuh SIEM solution and its components. Wazuh Agent sends logging data to Wazuh Server, which relays this data to Indexer for creating indices. Once indexed, the data is forwarded by server to a data collection point. On the right section with yellow color is Analysis Tool. The data is processed by Data Processor and passed to Analysis Engine including the model to be queried. Output is then provided to user.

5.3 Retrieval-Augmented Generation (RAG)

This section contains an overview of what the Retrieval-Augmented Generation (RAG) stands for and its benefits in using large language model in specific topic. Following this is the architecture of the tool implemented in this paper.

5.3.1 Retrieval-Augmented Generation Overview

Retrieval-Augmented Generation (RAG) is a technique used for enhancing the accuracy and reliability of large language models in specific topic, with the use of additional external sources to retrieve text documents and use them as additional context for generating the model output[11].

Large language models are proving to be excellent at generating text that appears to be written by human. However, when used in tasks that require specific knowledge, which includes cyber security tasks, they tend to hallucinate and provide unreliable answers. These answers can be taken with a grain of salt, or the technique such as RAG can be used to aid the models in completing knowledge demanding tasks.

Another motivation to use RAG is for the organizations that would like to complete tasks using internal knowledge, but the confidentiality of this information does not allow

for the usage of external LLMs. This is especially the case in cyber security related tasks when feeding an external LLM could greatly affect privacy of the organization by leaking secret information. One of the options is to run such model locally, however the resources required to run some of the largest generative AI models are also an aspect to consider. This leaves smaller organizations which would like to implement a topic specific solution with the internal information as an input, with a necessity to opt for smaller, optimized models. As these models are usually trained with significantly lesser amount of parameters, than their bigger peers the inaccuracy of the results in topic specific tasks raises. The main goal of the Retrieval-Augmented Generation technique is to fill these gaps by providing models with context from chosen information sources that is tailored for each query.

On a high level, the creation of Retrieval-Augmented Generation tool consists of two parts. Preparing the context retrieval system and a query processor feeding the context based on the query and the query itself. The ingestion of the data is broken into following steps (based on [10]):

- **Loading data sources to text** which involves pre-processing the data to be usable by embedding models, choosing the data sources, validating its relevancy for the specific task.
- **Splitting the text to chunks** is a process of slicing the text data into smaller units in order for the model to be able to concept it into its context window.
- **Text embedding** is the process of creating a numerical embedding for each chunk, which helps to select the most relevant text chunks for a given question by identifying the most similar chunks in the embedding space.
- **Loading the embeddings to vectorstore** for efficiency and speed.

The figure 5.2 represents the data ingestion for Retrieval-Augmented Generation (adopted from [10]).

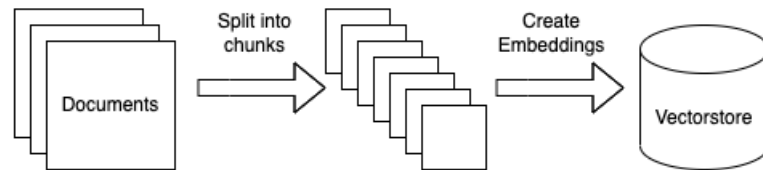


Figure 5.2: The figure shows the data ingestion model for Retrieval-Augmented Generation.

The next part is to query the model. This can be performed using a chat history, in case that the solution works as a chat bot remembering the previous answers. In cases like security alerts analysis, this step may be omitted. If this option is omitted the query prompt should be optimized for the model to produce understandable and structured answers that do not require additional queries. The more advanced architecture could even consist of multiple models chained together. One of the models could purely analyze the security alerts and the second model could be used to generate incident response actions based on the answer of the model analyzing the alerts. The second part of querying the model is to search for relevant documents in vectorstore database. This is done using the same

embeddings as the ones used to create the database. Once the question, the chat history and the relevant documents are combined into single query, it can be passed to model to generate the response. This architecture is based on the [10] and follows the same principles as the original paper [11], experimenting with the idea of Retrieval-Augmented Generation technique. The figure 5.3 shows the flow overview of the Retrieval-Augmented Generation technique described above from [10].

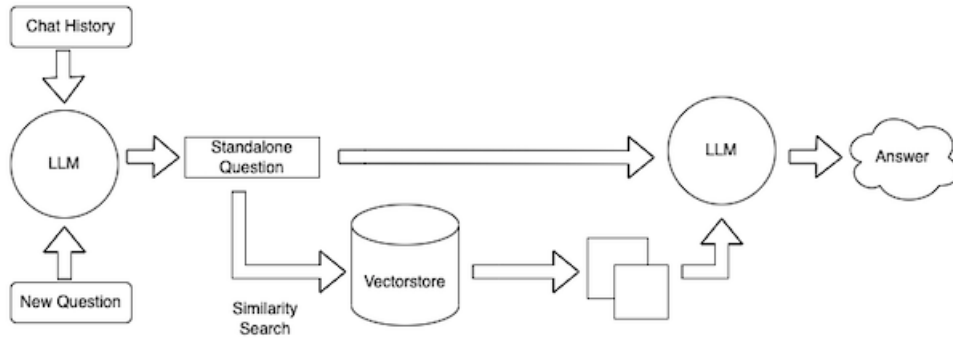


Figure 5.3: The figure shows the flow of the Retrieval-Augmented Generation technique used with large language model (LLM) (retrieved from [10]).

5.4 The Final Design of Adversarial Behavior Analysis Tool using RAG with LLM

This section contains the design details of the tool that has been implemented as a practical output of this work. It contains brief overview of the chosen tools, technology and the final architecture of the tool.

5.4.1 Chosen Technologies

The chosen technologies for designing the final tool for adversarial behavior analysis depend on multiple factors. The aim of this work was to use open-source technologies to maximize the cost efficiency of such tool, which could be easily adopted by smaller organizations, without the requirement of purchasing expensive tools. However, to improve the resulting outcome, there was a need to opt for some technologies with limited free usage or at some small financial cost. The following is the list of the technologies used in the design and final implementation of the adversarial behavior analysis tool:

- **Wazuh SIEM** is an open-source solution for the security information and events management system and is representing a traditional tool that feeds the security alerting data to the adversarial behavior analysis tool.
- **LangChain** is an open-source framework for developing tools using large language models. This work uses its Python library.
- **Mistral 7B** is an open-source large language model used to generate the descriptions of the adversarial behavior.

- **Ollama** is an open-source solution allowing to run large language models locally and to interact with them. It supports Mistral 7B and is supported by LangChain.
- **ChromaDB** is an open-source embedding database used to store embeddings in vectorstores. The LangChain Python library contains interface for operations with ChromaDB.
- **OpenAI API** is API used to create embeddings in this work.
- **VirusTotal API** is used for context enrichment with details about malware samples.
- **MITRE Framework** is used as a knowledge base source.

The selection of the right large language model for the goals of this work depend on two main factors. The performance of the model and its resources consumption. With larger amount of parameters the hardware requirements for running the model increase. The practical part of this work was performed on a single host machine, including the security information and event management system (SIEM). This created a requirement for the model that is the most efficient for its size. Three models taken into consideration were Llama 2 7B, Llama 2 13B and Mistral 7B.

The Mistral AI has conducted a series of benchmarks comparing the model performance to Llama models showcasing that Mistral 7B is outperforming the Llama models, including the ones with higher parameter count. The figure 5.4 shows the benchmark results compared to Llama 2 7b, Llama 2 13B and Llama 1 34B [9].

The parameters important for this work are especially reasoning, comprehension, beyond the benchmark hurdle and code. For the tool designed in this work is important to use model that is able to reason about the provided context and detection details, to comprehend the task and context data provided. The code portion is an important attribute to ensure, that the model will not struggle with scripts potentially introduced to its query. Beyond the benchmark hurdle represents the models adaptability to beyond standard tasks.

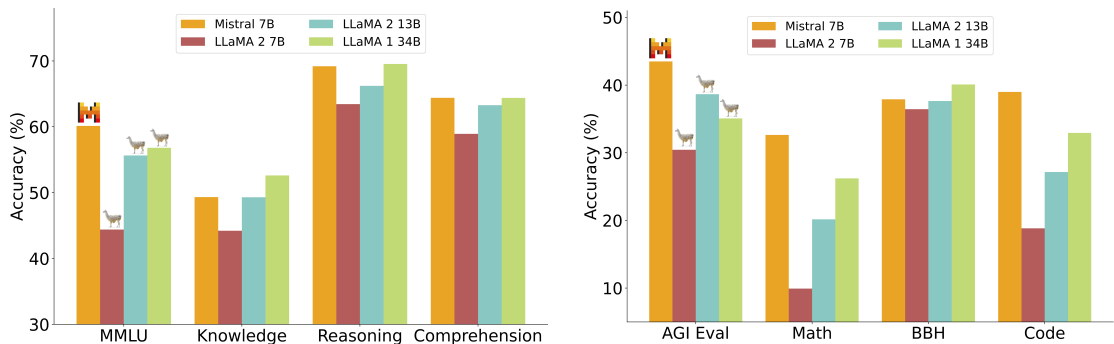


Figure 5.4: The figure shows the performance of Mistral 7B compared to Llama models. The benchmarks performed are MMLU: Massive Multitask Language Understanding, Knowledge, Reasoning, Comprehension, AGI Eval: Artificial General Intelligence Evaluation, Math, BBH: Beyond the Benchmark Hurdle, Code. (retrieved from [9]).

The solution for running the large language models chosen is **Ollama**. Another option was **LMStudio**, which supports larger amount of models and possibility to directly import

them from HuggingFace. However, during the testing of both Ollama was deemed more pleasant to work with. Ollama supports Mistral 7B.

The OpenAI API is used to create OpenAIEmbeddings and is the only paid tool in this work. An alternative considered and tested was OllamaEmbeddings, however it produced significantly worse results and the search function was not able to retrieve the context information from vectorstores properly.

The last technology to be discussed is VirusTotal API. VirusTotal is an online scanner used to analyze files submitted by users, using a large amount of security tools [12]. In this paper it is used as an enrichment source to retrieve additional information that can be either passed as a context to the model directly, or used to search for the information in vectorstores database. It comes with a limitation, as the free tier allows only 4 API requests per minute. This is sufficient for the testing purposes in this work, but would either needed to be replaced by similar, free tool, or upgraded to paid tier.

5.4.2 Implementation Design of RAG with LLM for Adversarial Behavior Analysis

The design used to implement the adversarial behavior analysis tool is briefly explained in this part.

The figure 5.5 shows the interaction model between main components, which are the following:

- **Alerts Collector**
- **Alerts Processor**
- **Analysis Engine**
- **Vectorstores Database**

Alerts Collector is responsible for periodically collecting the alerts generated by Wazuh SIEM and to store it in file system. The alerts collected are in a JSON file format. The alerts are stored in single file, which contains all the alerts collected during the current time interval. The default alert interval is set to five minutes. This can be changed depending on the performance of the system and the required mean time to respond. The Alerts Collector is running on Wazuh Server. It is querying Wazuh Indexer API to fulfill this task.

Alerts Processor processes the alerts collected by Alerts Collector. It defines the parameters required to analyze alerts based on the triggering rule. The Alerts Processor parses the required data, employs data enrichment and generates query parameters for Analysis Engine. The part of the query parameters is whether custom prompt template should be used or not. If the query for the currently processes alert requires custom template, Alerts Processor passes this information to Analysis Engine including the required template. Another parameter that is controlled by Alerts Processor is, the information how should the context be searched by the Analysis Engine in Vectorstores Database. This depends on the triggering rule and reflects whether the context search query

should be split into multiple parts or as a single query. **Alerts Processor** launches **Analysis Engine** and waits for its response. Once the response is received, **Alerts Processor** saves the result to the file system. Each individual alert is stored separately named after its `id` in Wazuh.

Analysis Engine employs the large language model to create descriptions of adversarial behavior identified from Wazuh alerts. Based on the parameters received from **Alerts Processor** it searches the **Vectorstores Database** for additional context that has to be provided to large language model alongside the query data encapsulated in prompt. The prompt is defined by prompt template. **Analysis Engine** has a default prompt template set, but it is able to receive prompt templates from **Alerts Processor**. It queries the model for response and appends the sources of context data to the response. It returns the response to **Alerts Processor**.

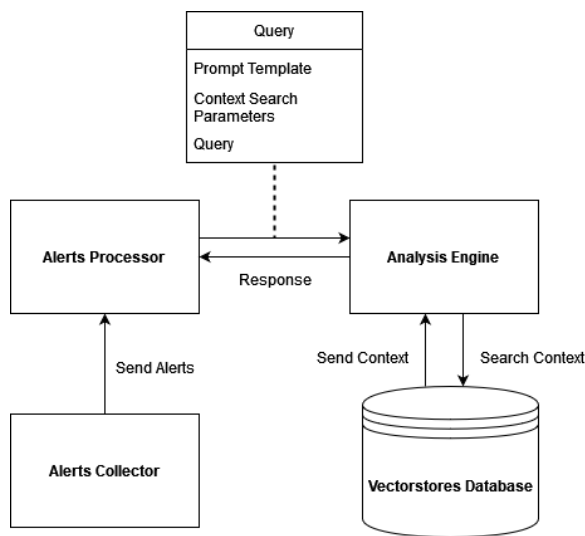


Figure 5.5: The figure shows the design of the architecture and the interaction between the components. The alerts saving and data enrichment is omitted for simplicity.

Chapter 6

SIEM Deployment and Attack Simulation

This chapter begins the practical component of this paper. It describes the preparation of testing environment, which includes deploying Wazuh Security Information and Event Management (SIEM) system in a virtual environment. Following this is the simulation of adversarial behavior in order to determine the alerting capabilities of the deployed system and to generate security alerts.

6.1 Wazuh Deployment and Configuration in Virtual Environment

This section contains the steps taken to deploy and configure Wazuh SIEM within virtual environment in VirtualBox.

6.1.1 Wazuh Open Virtual Appliance

The nature of this work requires a setup within virtual environment due to high risk of infecting the host with malware. The ideal option to install Wazuh SIEM is Wazuh Open Virtual Appliance (OVA). It is pre-built virtual machine image, which can be directly imported to VirtualBox [30].

The system runs on Amazon Linux 2 64-bit and contains all the necessary components for the complete Wazuh server:

- Wazuh Manager
- Wazuh Indexer
- Wazuh Dashboard

The requirements that must be met before the Wazuh Virtual Machine can be imported into a host operating system include that the host operating system must be a 64-bit system and hardware virtualization must be enabled in the host's firmware. The host operating system used for the virtual environment is Windows 10 Professional edition 64-bit. The table 6.1 shows the hardware specifications of the virtual machine used for the purposes of this work. The specifications align with those outlined in the Wazuh documentation [30].

Component	CPU (cores)	RAM (GB)	Storage (GB)
Wazuh v4.7.4 OVA	4	8	50

Table 6.1: The default hardware specifications for Wazuh v4.7.4 OVA retrieved from [30]

The last step before the first launch was to configure the graphic controller of the machine to `VMSVGA` as advised in [30].

6.1.2 Deploying Wazuh Agent on Windows 10 Endpoint

The operating system chosen for the endpoint that will act as a victim host is Windows 10 64-bit Professional edition. The reason behind this choice is that according to [24] this is the most used version of Windows operating system at the time with more than 69 %. The figure 6.1 (adopted from [24]) shows the Windows versions used by visitors of websites with StatCounter from January 2018 to July 2023.

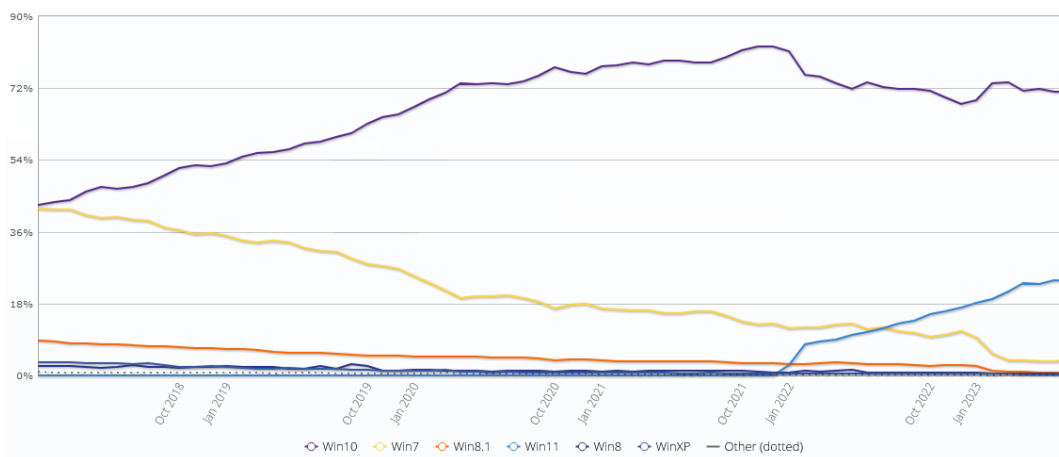


Figure 6.1: The figure shows Windows operating system versions throughout January 2018 to July 2023. Purple line: Windows 10, light blue line: Windows 11, yellow line: Windows 7. The rest is Win8, Win8.1 and WinXP. Adopted from [24].

The table 6.2 shows hardware specifications used for Windows 10 endpoint. The specifications were adjusted during the testing for optimal performance.

Component	CPU (cores)	RAM (GB)	Storage (GB)
Windows 10 64-bit	2	6	50

Table 6.2: The hardware specifications of Windows 10 machine used as victim endpoint.

After successful operating system installation some common applications were installed using Ninite. This is done to mimic a real world host machine in case some malware samples with Virtualization/Sandbox Evasion capabilities such as user activity based checks are tested [7].

The Wazuh Agent installation is quite straightforward. Wazuh provides Microsoft Windows Installer (MSI) file and the installation may be performed using both command line

interface and graphical user interface. The Manager IP address is an installation parameter that has to be configured during the installation. Once the agent is installed it with provided Manager IP it will be able to automatically request the key to encrypt communication between the agent and the manager and import it [28].

6.1.3 Wazuh Server and Agent Configuration

To be capable of the basic monitoring an additional configuration of both Wazuh server and Wazuh agent is necessary. The following was configured on Wazuh agent:

- **Windows Defender Logs** shows information about the malware detection, scans passed, and actions taken.
- **System Monitor (Sysmon)** is a Windows service and drive persistent on the system across reboots, used to monitor system activity by observing processes being created and terminated, network connections, and modifications to files [17].
- **Windows Power-Shell activity monitoring.**
- **Windows logs** including security logs, application logs and system events logs.

An attempt to integrate and configure Sigma rules was made by integrating an open-source tool Chainsaw with Wazuh. The integration is done by installing Chainsaw on endpoint agent and configuring Wazuh wodle command to run a PowerShell script. This script retrieves Sigma rules from official Sigma repository and runs Chainsaw executable using this rules [22].

The Chainsaw was able to scan the host using Sigma rules when ran manually, however the only Wazuh alert generated by automated execution was about failed Sigma rules update. There could be several reasons causing this issue, including insufficient privileges or incorrect Wazuh configuration. The manual scans using Chainsaw should suffice in case there is a need for detections based on Sigma rules.

The next to be configured is Wazuh Manager. Wazuh comes with out-of-the-box defined rules, however to be able to increase detection capabilities it is important to add some additional rules. One subset of rules that can be added are open-source rules for the Sysmon events. There is couple of rules that address Sysmon events to be imported to Wazuh on GitHub. The rules used to complement Wazuh rules used in this work are from [21]. The rules can be imported to Wazuh via Wazuh Management Dashboard under

Wazuh -> Management -> Rules -> Manage Rules Files -> Custom Rules.

The rule files are stored in XML format. Once a new rule set is introduced to Wazuh the Wazuh Manager has to be restarted with command line command shown in listing 6.1.

```
systemctl restart wazuh-manager
```

Listing 6.1: Restarting wazuh-manager

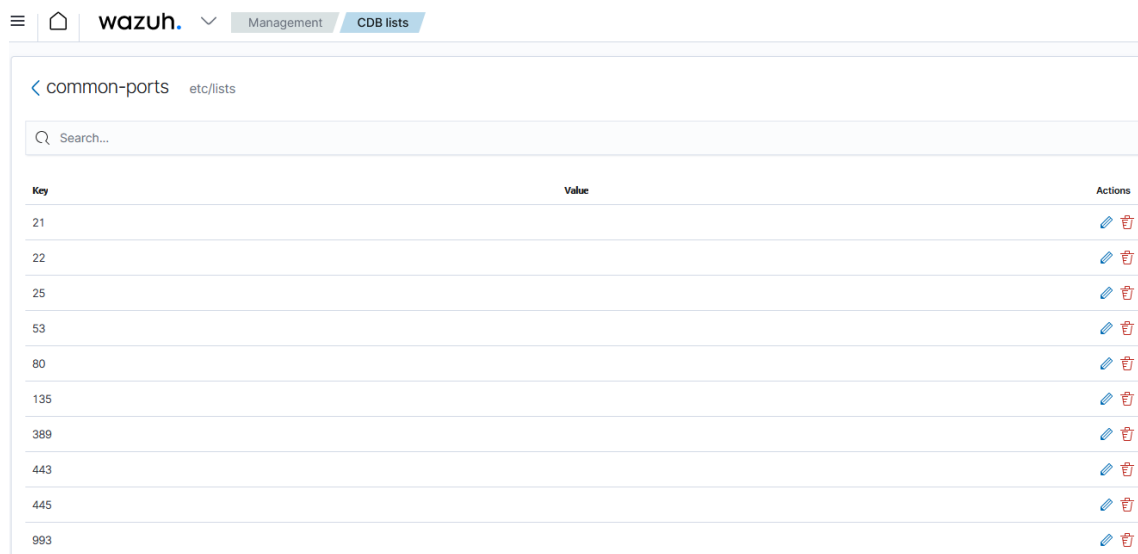
The two remaining rules were configure on Wazuh Manager. A network connection to uncommon port and Server Message Block (SMB) connection to external IP address. This can be done using Constant Database (CDB) list in Wazuh. To be able to detect a network connection to uncommon port, it is needed to specify what are the uncommon ports. This list can then be adjusted according the needs of an organization. The same applies for SMB

connection to external IP. The internal IP addresses can be whitelisted for SMB connections and any other address is external. The creation of a CDB list in Wazuh can be done via Wazuh dashboard or by saving the list directly on the server. The list has to be saved at `/var/ossec/etc/lists/` and read-write permissions have to be set by `chmod 660`. Once the list is created it has to be added to configuration file `/var/ossec/etc/ossec.conf`. For the Wazuh to be able to pull the list from lists folder a line from the listing 6.2 has to be added among other list definitions.

```
<list>etc/lists/common-ports</list>
```

Listing 6.2: Add list to Wazuh

To apply the changes to the configuration file a command from 6.1 has to be executed again. Once everything has been done the list will appear in CDB list section on Wazuh dashboard 6.2.























Key	Value	Actions
21		 
22		 
25		 
53		 
80		 
135		 
389		 
443		 
445		 
993		 

Figure 6.2: The Constant Database (CDB) list of common ports displayed on Wazuh dashboard. The first 10 ports are displayed.

The next step is to create a Wazuh rule for the alerts to generate on events when a network connection is made to different ports than the ones listed in the newly created list. The rule is shown in the listing 6.3. When uploading rules via Wazuh dashboard the Wazuh Manager is automatically restarted. The Wazuh SIEM will now be able to detect outgoing connections to uncommon ports and possibly detect malicious connections.

```

<rule id="102600" level="12">
  <if_group>sysmon_event3</if_group>
  <list field="win.eventdata.destinationPort" lookup="not_address_match_key
    ">etc/lists/common-ports</list>
  <description>$(win.eventdata.image) Made a Network Connection to Uncommon
    Port</description>
  <group>sysmon_event3,</group>
</rule>

```

Listing 6.3: A Wazuh rule to trigger an alert of level 12 when a network connection is made to a port that does not match any of the ports in common ports list.

A similar rule can be created for detecting SMB connections to external IP address. The principle is the same. A list with internal IP addresses and a rule to trigger an alert need to be created. The rule created in this work can be seen in listing 6.4.

```

<rule id="102601" level="14">
  <if_group>sysmon_event3</if_group>
  <list field="win.eventdata.destinationIp" lookup="not_address_match_key">
    etc/lists/internal-ips</list>
  <field name="win.eventdata.destinationPort">445</field>
  <description>$(win.eventdata.image) Made an SMB Connection to External IP
    </description>
  <group>sysmon_event3,sysmon_smb_external,</group>
</rule>

```

Listing 6.4: A Wazuh rule to trigger an alert of level 14 when an SMB connection to an external IP on port 445 is detected.

The configuration is set and the SIEM should be able to trigger enough alerts for this project. The following section will describe the simulation of adversarial behavior performed on the victim machine in order to generate alerts.

6.2 Simulation of Adversarial Behavior

This section explains the steps taken to generate security alerts in order to be able to test the implemented adversarial behavior analysis tool and to further improve it based on the inputs it is eligible of receiving.

The simulation of adversarial behavior performed in an attempt to generate security alerts, that could be used in this work, can be split into three testing types:

- Manual Testing - Kali Virtual Machine
- Automated Testing - Atomic Red Team Framework
- Specific Vulnerability Exploit (CVE-2023-23397)

As an attacking machine for this project was chosen a Kali machine deployed in virtual environment. Its specifications are listed in the table 6.3.

Component	CPU (cores)	RAM (GB)	Storage (GB)
Kali Rolling (2024.1) x64	2	2	20

Table 6.3: The hardware specifications of Kali virtual machine used as an attacker endpoint.

The following subsections comment on the steps taken in each testing type.

6.2.1 Manual Testing Using Kali Virtual Machine

The initial idea was to create some Windows payloads in Metasploit framework using `msfvenom`, deliver it to the machine and let the virus connect to the attacker machine. The first attempt was made before fully configuring the Wazuh Manager and Wazuh agent and the alerts did not trigger. This led to a realization a much more configuration would need to be done in order to detect malicious behavior. After the first failure to generate alerts with activity performed from Kali virtual machine towards victim machine hosted on the Windows endpoint, the Microsoft Defender Logs were enabled. However, this was insufficient as if the Microsoft Defender was kept active, hunting for threads, it would not allow for any payload from Kali virtual machine to be executed.

On the other hand, when the Microsoft Defender was turned off, the reverse TCP connection was successfully made. At this phase of the testing, the alert triggered by connections to uncommon ports has not been configured yet. The turned off defender did not produce any alerts, thus there was a need to catch this behavior with different alert. This is why the alert from 6.2 in listing 6.3 was created.

This approach has proven to be time consuming and nonproductive. It might be useful for slowly setting up Wazuh alerts and creating robust system capable of detecting various types of alerts, but there was a need for faster way to create various types of alerts. Also as the design of the implemented adversarial behavior analysis tool counts with MITRE Techniques to be used as a knowledge base, this approach lacked the ability to swiftly cover various types of techniques.

6.2.2 Testing with Atomic Red Team Framework

This subsection describes an approach chosen after manual testing in Kali virtual machine has been deemed unworthy for the purposes of this work.

For the more complex testing an open-source framework Atomic Red Team from Red Canary was used. It is a library of tests that can be used by security teams to simulate adversarial activity in their environment. The atomic in the name of the project relates to the character of the size of the tests. The tests are small and atomic. The atomic tests are mapped to MITRE Framework [3]. This makes it an ideal candidate for the purposes of this work.

A big benefit of this framework is that a specific operating system can be chosen, filtering out the tests that are not needed for this specific use case. The tests can be run either one by one or all at once, which is not recommended. This is due to the fact that there is a certain delay of the tests executions and the cleaning of the environment needs to be performed to be able to continue without an enormous mess made in logs.

To make the testing even smoother Atomic Red Team provides a PowerShell module called `Invoke-Atomic`. The module was installed on the victim endpoint. To ensure the

undisturbed run of the tests, the folder to download and store Atomic tests and payload has been whitelisted by exclusion in Microsoft Defender.

The testing with the framework was first done automatically to try to generate as many alerts as possible and then as needed for the testing of specific scenarios. To conduct the automatic, yet controlled test a PowerShell script was created. The script is shown in the listing 6.5 (modified version of the original retrieved from [4]).

```
$techniques = gci C:\AtomicRedTeam\atomics\* -Recurse -Include T*.yaml |
    Get-AtomicTechnique

foreach ($technique in $techniques) {
    foreach ($atomic in $technique.atomic_tests) {
        if ($atomic.supported_platforms.contains("windows") -and ($atomic.
            executor -ne "manual")) {
            Invoke-AtomicTest $technique.attack_technique -TestGuids $atomic
                .auto_generated_guid -GetPrereqs
            Invoke-AtomicTest $technique.attack_technique -TestGuids $atomic
                .auto_generated_guid -ExecutionLogPath 'C:\AtomicLog\mylog.
                csv'
            Start-Sleep 3600
            Invoke-AtomicTest $technique.attack_technique -TestGuids $atomic
                .auto_generated_guid -Cleanup
        }
    }
}
```

Listing 6.5: Script for automatically launching Atomic Red Team tests on Windows operating system. It is set to sleep after each test for one hour. It stores the logging data to mylog.csv. The script has been retrieved from [4]

There were several sessions for launching the atomic tests due to few complications. Some of the dependencies were unable to be installed automatically and some of the tests require user interaction. The figure 6.3 shows the increase of Wazuh alerts during the beginning of one of the testing sessions. The relevant rule levels were selected as low level alerts are noisy at almost any time.

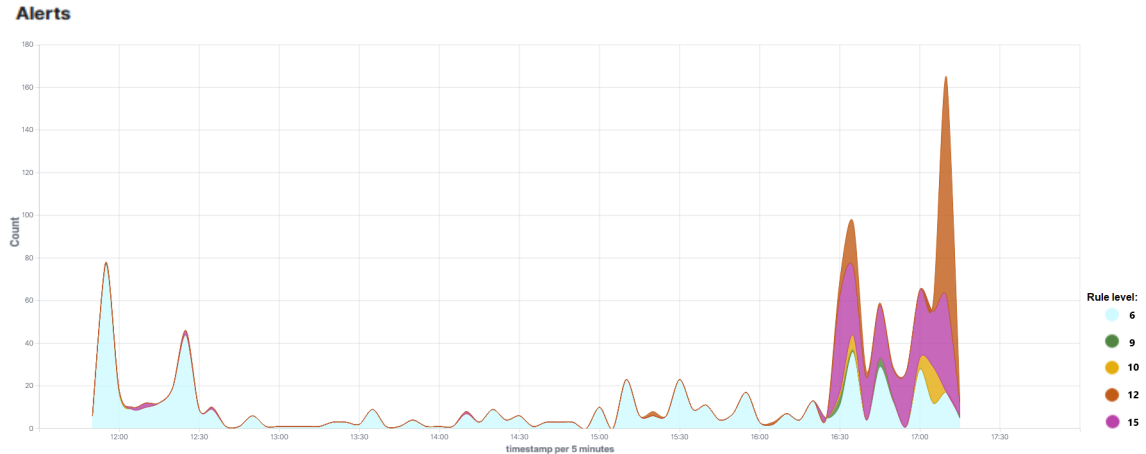


Figure 6.3: The figure shows an increase of Wazuh alerts during one of the testing sessions. First half of the timeline represents no malicious activity and the spike of the alerts begins at the time of the test beginning. Some of the rule levels were omitted for the clarity.

6.2.3 Exploiting the CVE-2023-23397

The next goal in simulation of adversarial behavior was to try to find a vulnerability that is not easily detectable when exploited. The vulnerability CVE-2023-23397 was provided by technical supervisor of this work Ing. Petr Chmelář, Phd. as a suggestion for the review and it was chosen to be exploited on the victim endpoint.

On March 14th, 2023, Microsoft has released the details about Exchange Online Protection (EoP) vulnerability in Microsoft Outlook for Windows, identified as CVE-2023-23397. This vulnerability, discovered by Microsoft Threat Intelligence, involved the potential theft of NTLM credentials without user interaction, triggered by a specially crafted message. It affected all supported versions of Microsoft Outlook for Windows [14].

In order to successfully exploit this vulnerability the version of Outlook in the test environment has to be downgraded to any version prior to the update addressing this vulnerability. Microsoft support portal provides guide to downgrade Office package with Office Deployment Tool [13]. To downgrade to the vulnerable version, we have reviewed the [18] and found the version prior to the release of the vulnerability. The full version number chosen is 16.0.15928.20198. For the Office Deployment Tool to choose this version a configuration file in XML format needs to be provided as an execution parameter. We have written the configuration file shown in the listing 6.6. The configuration file specifies the Office version and contains an instruction to disable further updates.

```

<Configuration>
  <Add OfficeClientEdition="64" Channel="Current" Version
    ="16.0.15928.20198">
    <Product ID="0365ProPlusRetail">
      <Language ID="en-us" />
    </Product>
  </Add>

  <Updates Enabled="FALSE" Channel="Current" />
-</Configuration>

```

Listing 6.6: The configuration file used to specify the version of Office package to revert to. It contains an instruction to disable further updates.

When the vulnerable version of the application is on the system, the specially crafted Outlook item, specifically the appointment for the calendar, needs to be created. This can be done in two different ways:

- Using development kit for Outlook compatible, malicious message creation.
- Using OutlookSpy plugin to change the parameters of the message created in Outlook.

The first approach includes implementing the specially crafted appointment message using `MsgKit` from [23]. This was demonstrated by Dominic Chell in [5]. This approach has been attempted during the exploitation attempts of CVE-2023-23397, however unsuccessfully. The message was not able to be built due to the fact that for it to be built a modification has to be made to the `MsgKit` library on a local copy. This is no longer possible as the kit has been since updated.

The second approach is to use OutlookSpy plugin for Outlook, to modify the appointment created directly in Outlook. The principle and the modifications to the message needed for successful exploit remain the same. Dominic Chell found out, that the Microsoft script released for the scanning for malicious Outlook messages, is specifically looking for the `PidLidReminderFileParameter` property of the mail [5].

The `PidLidReminderFileParameter` is used to specify the filename of the sound that a client application should play when the reminder reaches the deadline. [15]. This parameter accepts an Universal Naming Convention (UNC) path and can trigger NTLM authentication when alert's reminder is executed. Another property explored in Microsoft's cleaning script is `PidLidReminderOverride` [5]. The property `PidLidReminderOverride` determines if the client should use the specified properties to decide whether to play a sound and what file to play when a reminder is triggered [16]. When set, this setting forces the alert to be played, thus leaving the victim defenseless.

Before moving on to the crafting of the malicious appointment, the attacking machine needs to be set to listen for authentication attempts. We have set our Kali machine to listen for such attempts using `Responder`. We do not need to continue further with relaying the retrieved hash for authentication, or cracking it with brute-force. Simply outputting it to console will be enough. The `Responder` is launched using command from the listing 6.7.

```
kali@kali: responder -I eth0
```

Listing 6.7: Command to launch responder to listen for authentication attempts. `-I` parameter specifies the networking interface.

The attacker machine is ready and listening. The next step is to create a malicious appointment that will have `PidLidReminderFileParameter` set to the network share location of the attacker machine.

We have installed the OutlookSpy plugin and created a regular appointment first. Once the appointment is in the calendar, it can be open as an object in separate window by double-clicking it. On the menu bar for the new window, the OutlookSpy button launches a contextual window of this plugin allowing for reviewing and setting otherwise hidden properties of this object. We clicked on the Script tab and written the commands to change the properties needed to perform this exploit. This is presented on the figure 6.4.

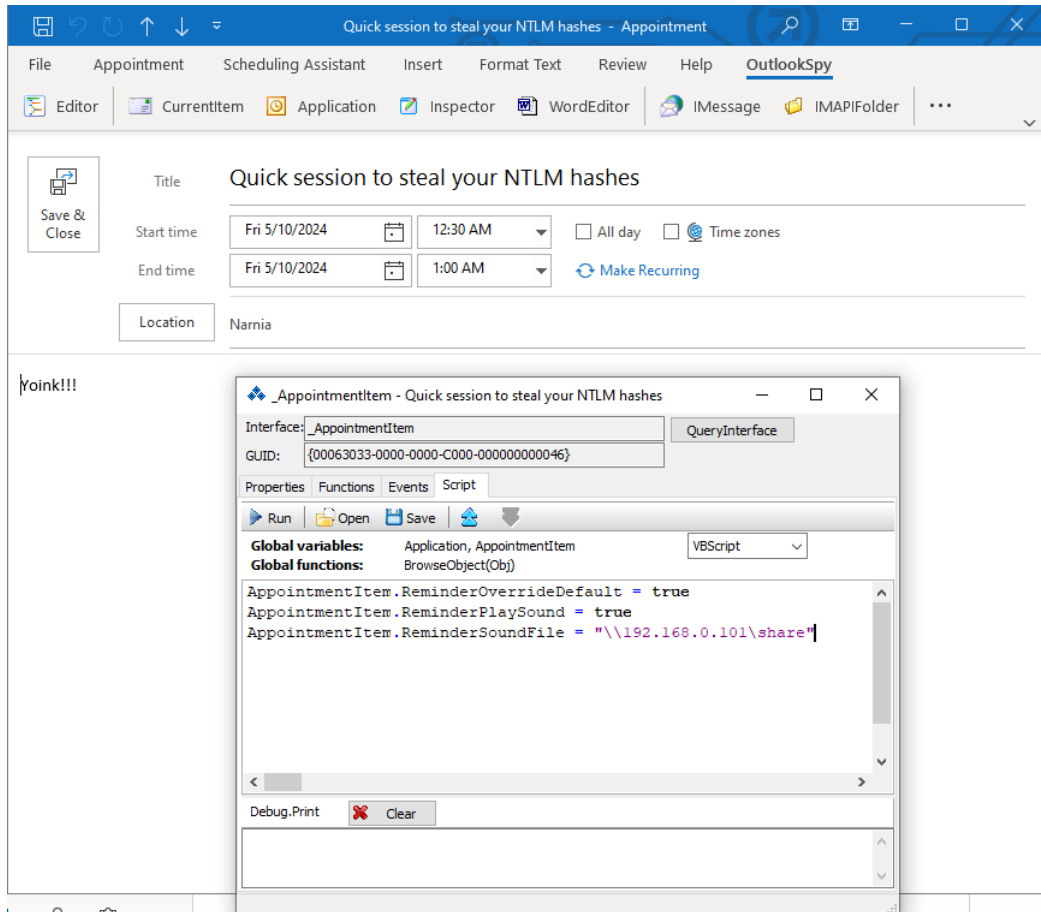


Figure 6.4: The figure shows an OutlookSpy plugin window with commands to set the parameters required to exploit the vulnerability. The appointment item can be seen in the background.

The following is to set the reminder for the malicious appointment to 0 minutes, in order for it to instantly trigger. This can be done via regular Outlook user interface.

The figure 6.5 reveals that the exploit has been successful and the terminal of Kali machine is full of leaked NTLM hashes.

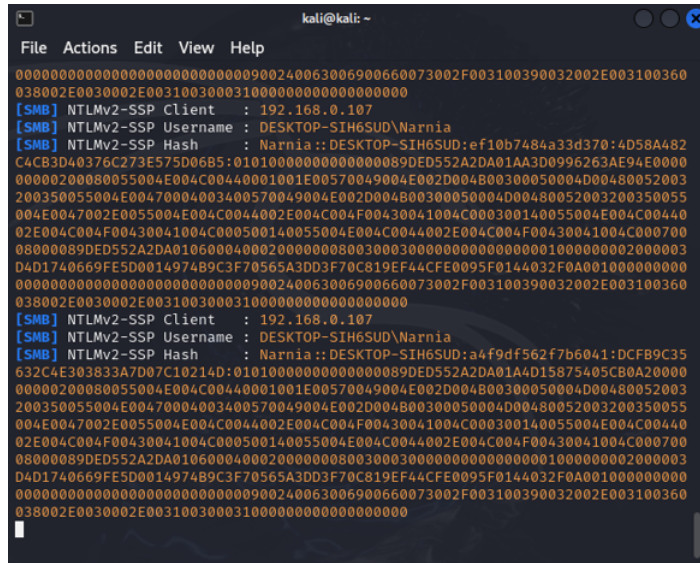


Figure 6.5: The figure shows the terminal window of Kali attacker machine after successful exploit of CVE-2023-23397. It contains user information and leaked NTLM hashes.

This vulnerability is highly dangerous as it is relatively easy to exploit and impossible to avoid if the victim's machine is vulnerable to it. The appointment could be simply forwarded to victim in a phishing mail.

When first exploiting this vulnerability in our environment, there was not a sign of an alert that could indicate that the Windows endpoint has been attacked. In such cases, it is up to diligent threat intelligence team to be constantly on the watch for newly disclosed vulnerabilities. Based on this findings the Wazuh rule from 6.4 has been implemented.

Chapter 7

Implementation and Experiments

This chapter covers the implementation of the alert analysis tool using Retrieval-Augmented Generation architecture with a large language model. The chapter concludes with experiments conducted on the implemented solution, to demonstrate the solution's capabilities, will be described at the end of this chapter. These experiments revealed some issues, leading to modifications in the implementation.

7.1 Implementing Retrieval-Augmented Generation with LLM

This section presents the implementation details of the tool for adversarial behavior analysis using large language model with Retrieval-Augmented Generation technique.

The tool has been implemented in Python programming language, with the version installed on the development host being Python 3.9.12. An open-source platform Ollama was used for hosting the large language model. The large language model used in this project is Mistral 7b. The embedding function used for the final version is OpenAIEmbeddings with the embedding model text-embedding-3-small. The embedding model was the only paid product used in this project and its usage costed 0.01\$. The database to store the embeddings in vectorstores is ChromaDB. As a wrapper over all of the technologies was used LangChain Python library.

The logical structure of the implementation follows the design presented in the section 5.4. The project consists of scripts in the main folder and three additional folders: data, chromadb and wazuh. The wazuh folder is a shared folder, shared with virtual machine hosting the Wazuh server. It contains the script that is used as a data collector to retrieve alerts from Wazuh Indexer. It is also a destination to store alerts retrieved for analysis in a file alerts.json.

The chromadb folder contains pre-built database, built during the implementation. This is mainly to save the cost related to using OpenAI embeddings. An additional version that uses OllamaEmbeddings function is in folder chroma, but is not part of the final solution. However, it can be used in case of missing OpenAI API Key.

The data folder contains three comma separated values (CSV) lists. The two files contain the MITRE techniques list and MITRE software list. These lists have been retrieved from [6] and further processed. The third list called mitre_dataset.csv is a resulting data set created by combining the two lists and cleaned of an unnecessary bloat, that would

worsen the accuracy of responses. The script for the preparation of the data is included in the main folder of the project.

The main folder contains the scripts representing the rest of the components described in the section 5.4. The following subsections detail the creation of the database and how do the components operate.

7.1.1 Dataset Preparation and Vectorstores Database Creation

This subsection consists of steps taken to create the database that can be used for searching for the relevant pieces of information from MITRE Techniques list and MITRE Software list.

The character of the data used for this project is a table consisting of name and the description of either MITRE technique or software from MITRE Software list. The descriptions are of variable length. While techniques descriptions are extensive, the software descriptions are much shorter. The descriptions contain URLs and citations across the whole description. This information would not be useful for the context creation, thus was removed using the script `prepare_dataset.py`. The script cleans the descriptions of citations and URLs and merges the two list into one uniform dataset.

The embedding of the data follows in the `create_mitre_chromadb.py`. Before the embedding, the data is split to smaller chunks. The size of chunks is a parameter that can be experimented with. The final solution uses chunks of 800 characters long texts. The overlap between the chunks is 80 characters. The embedding of the chunks of text is done using `OpenAIEmbeddings` with the model `text-embedding-3-small`. To be able to use this embedding model an `OpenAI API key` has to be provided as a parameter of the embedding function. The `API key` is retrieved from the environment variable called `OPENAI_API_KEY`, where it was stored using `setx` command. In case of missing key, an alternative embedding function `OllamaEmbeddings` may be used. This is an open-source embedding function and does not require `API key` for its functioning. It is important to note, that this function has been used in an initial implementation cycle. This function did not provide the required results and the solution was unusable.

After the text is embedded it is stored in chroma database, which is directly created using the chunks. The database is automatically persistent and the method `persist()` is obsolete.

7.1.2 Alerts Collection

The alerts collection is done on the Wazuh server by querying the Wazuh Indexer. This was not necessary in the given setup for this work as the Wazuh server from this project is a single cluster unit. However, having the option to use the querying mechanism from the outside of the Wazuh server leaves the door open for different setups and the tool can be used as a standalone unit.

The script used to query Wazuh Indexer and retrieve alerts is `wazuh_query.py`. It uses `Wazuh Indexer API`, which is served on `Port 9200` by default. The script combines `term` matching with `range` matching in a single `bool` query. Wazuh uses `OpenSearch`, thus the query language is `Query domain-specific language (DSL)` [2]. The `term` to used for retrieving the alerts is the `ID` of the rule. This means that only the alerts triggered by the rules that are supported should be queried. The time range for alert retrieval is based on

alert parameter `timestamp`. The time zone was set to `UTC+2`, as this is the timezone of both Wazuh server and Wazuh endpoint. The query retrieves alerts from the past 5 minutes. The query is shown in the listing 7.1.

```
query_body = {
  "query": {
    "bool": {
      "must": [
        {"term": { "rule.id": rule_ids } },
        {"range": {
          "timestamp": {
            "time_zone": "+02:00",
            "gte": "now-5m"
          }
        }}
      ]
    }
  },
  "sort": [
    { "timestamp": { "order": "desc" } }
  ],
}
```

Listing 7.1: The query used by Wazuh server to retrieve alerts using Wazuh Indexer API.

To make the Wazuh server automatically run the script in the required time intervals, a `Wazuh Wodle Command` is used. The command has to be put into configuration file of `Wazuh Manager` stored at `/var/ossec/etc/ossec.conf`. The command used in this project, set to be running in 5 minute intervals, is in the listing 7.2.

```
<wodle name="command">
  <disabled>no</disabled>
  <tag>LLM</tag>
  <command>python /media/sf_wazuh/wazuh_query.py alerts</command>
  <interval>5m</interval>
  <run_on_start>yes</run_on_start>
  <timeout>0</timeout>
</wodle>
```

Listing 7.2: The Wazuh Wodle Command used to run alerts collection script in 5 minute intervals.

As an alternative to have the alerts querying mechanism on the Wazuh server would be to query the alerts directly from the hosting machine of the analysis tool. We have faced issues to be able to communicate with Wazuh Indexer from outside of the Wazuh server. There is lack of a documentation for this use case. After saving the alerts in `JSON` format, they are processed by `alerts processor`.

7.1.3 Alerts Processing

The alerts processing segment is performed by script called `alerts_processor.py`. It is retrieving the the alerts stored by alerts collection module and processing them based on the triggering rule. During the testing phase of this project it was explored that due to the usage of more compact model, it is necessary for certain alerts to have their own custom `prompt template`, `query` and even context retrieval methodology. This is implemented in `alerts_processor`.

The `prompt template` is a template for query that is completed with context information and query. An example prompt template for the alert `Unsigned Image loaded by C:\Windows\System32\rundll32.exe` is in the listing 7.3.

```
"""
    You are senior security analyst, describing received alert to your
        junior colleague. The alert: {question}. Here is a description of
        observed MITRE Techniques:

    {context}

    ---

    Explain the alert and describe the loaded image's capabilities based on
        the MITRE Techniques.
"""
```

Listing 7.3: An example of prompt template. This particular template is used for the alert triggered by unsigned image being loaded by `rundll32.exe`.

The `alerts processor` is further responsible for:

- What information is used to search the context information in the database. This is based on the character of an alert. If the alert contains a file SHA256 hash, that can be analyzed using the VirusTotal, the information from VirusTotal is preferred. This way the language model can directly explain the behavior that the file has exhibited in different sandboxes.
- How is the context search performed. This is done as some of the alerts were getting better description results by the model when the context search was split into multiple queries. For some of the alerts a single query worked better.
- Saving the alerts to the file system.

The context for some of the alerts, related to files, is enriched using a VirusTotal integration. It is done by querying the `VirusTotal` API. This is implemented in separate module, that is called by the `alerts processor` when needed. In the current version of the tool, the information queried from `VirusTotal` API is MITRE Techniques exhibited by the file sample in various sandboxes. This allows the `alerts processor` to set the query to search context directly to these techniques. When combined with the right prompt template, the model is able to provide meaningful and beneficial behavior description, that

could be used by security analysts. The API key for VirusTotal has to be stored as an environment variable `VT_API_KEY`.

The response received from alerts analysis module is saved as a text file in `Analyzed_Alerts` folder. The resulting text files are named using the Wazuh alert's unique IDs.

7.1.4 Alerts Analysis

The alerts analysis is done by large language model `Mistral 7b`, which is provided the context information and query, encapsulated in prompt template.

The mechanism to query the model is implemented in `mistral_query.py`. It searches the chroma database for the context, based on the instructions received by `alerts_processor`. For searching the database, the same embedding function that was used to create the embeddings, has to be used.

The module invokes the model and formats its response appending the sources used for the context search.

7.2 Experiments

This section presents some of the experiments taken during the implementation and tuning of the final solution. The selected test logs are stored in the `test.log` file in the main folder of the tool. The outputs of the experiments in this section contain information and paraphrased information from MITRE Framework [6].

7.2.1 Experiments using OllamaEmbeddings

The first few experiments were conducted using the database created with `OllamaEmbeddings` method with `nomic-embed-text` model. The query using the `OllamaEmbeddings` with an unoptimized prompt template to test the context retrieval is shown in the listing 7.4. The portions of the context was condensed as it consisted of redundant information and literal representation of the descriptions from the MITRE Framework [6]. The context for the model contains redundant entries where the same information about `Boot or Logon Autostart Execution` is repeated multiple times with identical description parts. This redundancy could impact the model's efficiency. Instead, it could introduce a bias or overemphasis on the repeated parts of the technique description.

After the initial experiment using `OllamaEmbeddings` the several tests introducing changes to prompt template were conducted. The prompt template did not provide significantly different results.

Another query parameter introduced to the context search query and the model was the alert description. The first attempt was to propose a fictional alert description, to see how it affects the response. This was rather a human ramble about what happened, than structured short description of an alert. This had an effect on the accuracy of the context search, as well as model's response. The model was providing better responses to shorter, structured queries.

The listing 7.5 presents an attempt to query the model with a longer query and adjusted prompt template. Both the query and the response are condensed for better readability, leaving only the significant parts.

The query might be already too long, containing all the information from the detection. The context search returned is unrelated to the alert details and yet again consists of redundant information. This was thought to be due to long queries passed as the search parameter for the context search. However, when re-creating the database using `OpenAIEmbeddings` the context retrieved from the database was not redundant and consisted of relevant information that matched the alert details. A positive aspect worth of highlighting is that the model was able to reason about the detection details and in some part of the response was not biased by the given context.

```
Human:
Answer the question based only on the following context:

(3x) Boot or Logon Autostart Execution
description: The description has been hidden.

(2x) Boot or Logon Autostart Execution: Port Monitors
description: The description has been hidden.

Answer the question based on the above context: How can
adversaries utilize Boot or Logon Autostart Execution?
-----
Response: Adversaries can utilize Boot or Logon Autostart
Execution by setting a port monitor through the 'AddMonitor'
API call. This allows them to load and run an adversary-
supplied DLL during system boot, which can be located in 'C
:\Windows\System32'. The print spooler service, 'spoolsv.exe
', under SYSTEM level permissions will load and run the DLL
on boot. This method is used by adversaries for persistence
or privilege escalation purposes.
```

Listing 7.4: The first query to the implemented tool, aimed at testing the context search functionality. The prompt has been condensed. (The response from the model contains paraphrased content from [6].)

Human:

You are senior security analyst, describing an alert to your junior colleague. Explain the situation based on the following context and question details:

(3x) Hijack Execution Flow: Path Interception by Search Order Hijacking

(2x) Hijack Execution Flow: Path Interception by Unquoted Path

Answer this question based on the above context: We have received an alert with the following information:
technique_id=T1016,technique_name=System Network Configuration Discovery, Commandline: nslookup -querytype=ALL -timeout=10 _ldap._tcp.dc._msdcs.WORKGROUP. Suspicious Windows cmd shell execution Can you explain it?

Response: Based on the context provided, the alert indicates that there was a suspicious execution of the "cmd.exe" command with the argument "nslookup querytype=ALL-timeout=10 _ldap._tcp..dc..WORKGROUP" in a Windows system.

However, this context does not explicitly mention any path interception or search order hijacking incidents.

If you suspect path interception or search order hijacking based on additional evidence, make sure to investigate further by examining the directories and paths involved in the executions of "cmd.exe" and "nslookup".

Listing 7.5: The query and the response of the model using `OllamaEmbeddings`. This query introduces alert details. The context search returned redundant information. The model was able to reason about the alert details and irrelevancy of the context provided. (The response from the model may contain paraphrased content from [6].)

7.2.2 Experiments using `OpenAIEmbeddings`

After the re-creation of the embeddings vectorstore database using `OpenAIEmbeddings` the context search began to retrieve results more relevant to the query. The software list, retrieved from [6], was added to the database, alongside with the change of the embedding function. This change could benefit the relevancy of the context information. If some executable is a part of the alert details, model should be given a short description of its usage.

The very first test using the new additions to the context database has proven its benefits. The embeddings are also more accurate, thus retrieving more relevant information.

In the listing 7.6 are noticeably improved results of the context search. This aided the model to respond to the query with higher accuracy. The model was able to explain every tool mentioned in the alert details. The subject to improve in the future might be the suggested incident response. This could be adjusted with better prompt, demanding a bullet points with remediation actions.

```
Human:
You are senior security analyst, describing an alert to your
  junior colleague. Explain the situation based on the
  following context and question details:

['Nltest', 'XSL Script Processing', 'XSL Script Processing', '
  cmd', 'netsh']

Answer the question based on the above context: We have received
  an alert with following information: technique_id=T1059,
  technique_name=Command-Line Interface Commandline:"cmd.exe
  \" /c nltest.exe /dclist:%%userdnsdomain%% Can you explain
  it?
-----
Response: This alert indicates that there has been an observed
  use of the "Command Line Interface" (CLI) technique,
  specifically with the command 'cmd.exe' followed by 'nltest.
  exe'. The 'nltest.exe' is a Windows utility used to list
  domain controllers and enumerate domain trusts.

The command 'nltest.exe /dclist:%%userdnsdomain%%' is used here,
  where '%%userdnsdomain%%' represents the current user's
  domain. This command is being executed in a batch process
  using the Windows command-line interpreter, 'cmd.exe'.

The CLI technique is often used by adversaries to execute
  commands or tools directly on target systems, as it provides
  them with an easy way to interact with the system and
  perform various tasks. In this case, the use of 'nltest.exe'
  could potentially be for reconnaissance purposes, as it
  allows an attacker to discover information about the domain
  controllers and domain trusts within a network.

*** Generic recommendation for investigation. ***
```

Listing 7.6: The query and the response of the model using OpenAIEmbeddings. The context search returned relevant information that helped the model accurately reason about the alert details. (The response from the model may contain paraphrased content from [6].)

The following were tests on multiple types of alerts, to see how does the response change for the different alert type. Some of the alerts did not receive as good descriptions. This led to decision to create separate methods for specific types of alerts. These methods will customize the input that is provided to the model and the context search query. Some of the alerts require only the triggering `command line` and some will benefit from the `parent image` parameter. Using `LangChain` it is possible to chain multiple models together. This approach could be used to reserve one model to prepare the query and the second model to answer it using context like in this project.

The next step was to test an alert where there is `SHA256 hash` of the file available, to be able to use VirusTotal integration. It seemed like a great idea, but it revealed flaws in the `prompt template`. Using the previous prompt template, the model was unable to distinguish between what has been detected in the testing environment and what is reported from sandboxes in VirusTotal.

The information chosen from VirusTotal was MITRE Techniques exhibited by the sample file in supported sandboxes. Each technique was pulled only once. The techniques returned were used to search for the context information. The listing [7.7](#) shows the context provided to the model and a specific type of alert. The prompt has remained almost the same, except for the sentence: „`The above context contains MITRE techniques that the loaded image has demonstrated in Sandbox.`“ This sentence apparently was not enough and the model made a false assumption that the behavior described in the MITRE Techniques was detected in our environment. The prompt had to be adjusted using multiple iterations to come with pleasant result.

```

-- Prompt --
[Device Driver Discovery; Software Discovery; Virtualization/
  Sandbox Evasion: System Checks; File and Directory Discovery
  ~; System Information Discovery]

Explain the alert based on the above context and detection
  details: [Alert Details] The above context contains MITRE
  techniques that the loaded image has demonstrated in Sandbox
  .
  -----
Response:
This behavior itself is not necessarily malicious as rundll32.
  exe is a legitimate Windows utility used for running DLL
  files, but the fact that an unsigned image was being loaded
  raises concerns. The context provided in our alert mentions
  techniques related to Adversary Behavior and Discovery.

1. Device Driver Discovery: explanation + correlation to
  alert
2. Software Discovery: explanation + correlation to alert
3. Virtualization/Sandbox Evasion: System Checks:
  explanation + correlation to alert
4. File and Directory Discovery: explanation + correlation
  to alert
5. System Information Discovery: explanation + correlation
  to alert

Given this context, it is essential to investigate further. We
  should gather more data about the unsigned image in question
  , including its origin, digital signature (if available),
  and any associated network traffic or system activity.
  Additionally, we should assess the impact of the unsigned
  image on the system and determine if any remediation actions
  are necessary.

```

Listing 7.7: The query and the response of the model using `OpenAIEmbeddings`. The context was searched based on the MITRE Techniques exhibited by the malware sample from the detection. The model was not able to distinguish between our environment and sandbox. (The response from the model may contain paraphrased content from [6].)

The following was done to improve the query:

- **Prompt template has been restructured.** The alert details were moved to the first few rows of the query.

- **Higher emphasis on the details of usage of VirusTotal.** It was made clear, that the techniques described in the context are taken from VirusTotal analysis.
- **The task was better specified.** The task „Explain the alert and describe the loaded image’s capabilities based on the MITRE Techniques" better represents what is the goal of this query.

This resulted in an accurate response from the model, that can be seen in the listing [7.8](#). The model was able to describe the situation within our environment, the VirusTotal report, distinguish between them and to reason about file capabilities.

You are senior security analyst, describing received alert to your junior colleague. The alert: "Unsigned Image C:\\Users\\Narnia\\Downloads\\T1036\\2.dll loaded by rundll32.exe". We queried VirusTotal for details about this loaded image. VirusTotal reports several MITRE Techniques used by this image. Here is the description of observed MITRE Techniques:

[Context Information]

Explain the alert and describe the loaded image's capabilities based on the MITRE Techniques.

Response: The alert indicates that an unsigned image file named "2.dll" located at "C:\\Users\\Narnia\\Downloads\\T1036" was loaded by the legitimate system process "rundll32.exe". This information, combined with the MITRE Techniques observed in VirusTotal reports, suggests that the loaded image may be used for malicious activities.

The first technique mentioned is "Application Layer Protocol: File Transfer Protocols", which implies that this image could potentially be involved in data transfer or command and control communication between an attacker's server and the compromised system.

Another technique listed is "File and Directory Discovery". This suggests that the loaded DLL might be used to enumerate files and directories on the infected system, possibly for further reconnaissance or exploitation purposes.

Additionally, "Content Injection" is mentioned. This technique could mean that the loaded image might be capable of modifying existing files or injecting malicious code into them.

The technique "Software Discovery: Security Software Discovery" indicates that this image may attempt to detect and gather information about security software installed on the system, potentially to bypass or disable it for further attacks.

The description also mentions that Rundll32.exe, which is used to load the DLL in question, can be employed to execute Control Panel Items (.cpl files) and scripts like JavaScript. This raises concerns about potential PowerShell script execution or other similar malicious activities.

Listing 7.8: The query and the response of the model using OpenAIEmbeddings. The prompt has been restructured. The alert details are more straightforward. An emphasis was given on usage of VirusTotal. The model described the alert and the possible capabilities of the triggering file. (The response from the model may contain paraphrased content from [6].)

7.2.3 Enhanced Information and CVE-2023-23397

This subsection experiments with the idea of what could be possible, if the RAG LLM tool received some additional information. With the enriched information, bigger embeddings database and better security alerting, the prompt could look something like in 7.9.

```
You are senior security analyst, describing received alert to
your junior colleague. The alert: Windows system attempts to
connect to an SMB resource on external IP;ParentImage=
Outlook.exe. Here is a description of observed MITRE
Techniques:

Sources: ['Remote Services: SMB/Windows Admin Shares', 'Brute
Force: Password Spraying', 'Brute Force: Password Guessing']
CVE-2023-23397: [Information from CVE List - Known Exploited
Vulnerabilities []]

Explain the alert based on the MITRE Techniques.
-----
Response: The alert you received indicates that a Windows system
attempted to connect to an SMB resource using Outlook.exe
as the ParentImage. This behavior is significant because it
suggests that an adversary may be attempting to exploit a
known vulnerability, specifically CVE-2023-23397, in
Microsoft Office Outlook for privilege escalation purposes.
The adversary might be using this vulnerability to perform a
NTLM Relay attack against another service, allowing them to
authenticate as the user.
Furthermore, the connection to an SMB resource over an unsecured
or externally accessible channel raises concerns about
other MITRE techniques that could be employed, such as:
[List of techniques]
These techniques can enable adversaries to remotely access a
networked system, transfer files, run transferred binaries
through remote execution, and abuse single sign-on
applications or cloud-based services with federated
authentication protocols.
Additionally, consider reviewing the system configuration
settings to ensure that all necessary security measures are
in place, such as securing SMB ports (TCP 139, TCP 445),
enforcing strong password policies, and applying software
updates to address known vulnerabilities like CVE
-2023-23397.
```

Listing 7.9: The query and the response of the model using OpenAIEmbeddings. Enhanced context database and alerting, result in better descriptions from the model. (The response from the model may contain paraphrased content from [6].)

The results are surprisingly good. This response was achieved only by adding an information from known exploited vulnerabilities list from [1] and being able to formulate the query in a comprehensive way for the model to respond to. This opens a discussion for the possible improvements to the existing implementation in the section 7.3.

7.3 Possible Improvements to the Implementation

This section briefly introduces the options, how could the solution presented in this work, be further improved.

There is multiple aspects that the tool implemented as a practical part of this work could be improved. It is not a user friendly ready to be shipped application. However, it lays strong basis for the further improvements.

Some of the improvements could be:

- **Chaining multiple models together.** By chaining multiple LLM models together we could divide the demanding task of malware and adversarial behavior analysis into smaller pieces, that can be solved by models the most suitable for each task.
- **Using larger model.** This work demonstrated what can be achieved at home environment with limited resources and a small, compact model, compared to its LLM peers. If we could use larger models, while maintaining the privacy of the information fed to its prompts, we could achieve astonishing results.
- **Enhancing the knowledge base database.** By enhancing the database to retrieve context from and enabling it to be periodically updated, the responses from the model could increase in accuracy. The experiment in the subsection 7.2.3 has proven, that even small additions to the knowledge base improve the results significantly.
- **Fine-tuning the used models.** Either fine-tuning the models by ourselves or using fine-tuned models for the specific task could in combination with context driven queries present interesting results.

The technical aspects of the tool, that could be improved are, creating an integration to feed the response data to user interface and create a larger rule set to be able to describe more alerts. This section concludes the practical part of this work.

Chapter 8

Conclusion

This paper explored the possibilities of using a more compact large language model in a cost efficient way in a demanding task such as malware behavior analysis.

The theoretical background was reviewed in chapters 2, 3 and 4, with the emphasis on Host-Based-Intrusion-Detection (HIDS) systems, Malware Analysis and Large Language Models (LLM).

The paper then built on this foundation to design an architecture of adversarial behavior analysis tool using large language model with the technique of Retrieval-Augmented Generation (RAG) in chapter 5

The important part of this paper is a chapter 6 about deployment of Security Information and Events Management (SIEM) system, for ensuring the alerting capabilities. With the aid of penetration testing technique, this system has generated the very much needed data in the form of alerts that could be analyzed by the implemented tool. It also provided a way to automate the process of alerts collection.

The implementation part of this work from the chapter 7 has detailed the tool that is the result of practical part of this project. An inherent component of this chapter are the experiments conducted on the progressing versions of the implemented tool.

The end of this thesis discusses the possible ways to improve the implemented tool and looks forward to undisputed potential of Generative AI in specialized tasks.

Bibliography

- [1] AGENCY, C. . I. S. *Known Exploited Vulnerabilities Catalog* [online]. 2024 [cit. 2024-5-1]. Available at: <https://www.cisa.gov/known-exploited-vulnerabilities-catalog>.
- [2] AMAZON WEB SERVICES, I. *Query DSL* [online]. 2024 [cit. 2024-5-1]. Available at: <https://opensearch.org/docs/latest/query-dsl/>.
- [3] CANARY, R. *Atomic Red Team* [online]. 2024 [cit. 2024-5-1]. Available at: <https://atomicredteam.io/atomic-red-team/>.
- [4] CANARY, R. *Execute Atomic Tests (Local)* [online]. 2024 [cit. 2024-5-1]. Available at: [https://github.com/redcanaryco/invoke-atomicredteam/wiki/Execute-Atomic-Tests-\(Local\)](https://github.com/redcanaryco/invoke-atomicredteam/wiki/Execute-Atomic-Tests-(Local)).
- [5] CHELL, D. *Exploiting CVE-2023-23397: Microsoft Outlook Elevation of Privilege Vulnerability* [online]. 2024 [cit. 2024-5-1]. Available at: <https://www.mdsec.co.uk/2023/03/exploiting-cve-2023-23397-microsoft-outlook-elevation-of-privilege-vulnerability/>.
- [6] CORPORATION, T. M. *MITRE ATT&CK* [online]. 2024 [cit. 2024-5-1]. Available at: <https://attack.mitre.org/>.
- [7] CORPORATION, T. M. *Virtualization/Sandbox Evasion* [online]. 2024 [cit. 2024-5-1]. Available at: <https://attack.mitre.org/techniques/T1497/>.
- [8] DANIEL CID, R. B. *OSSEC Host-Based Intrusion Detection Guide*. 1st ed. Syngress Publishing, Inc., 2008. ISBN 9781597492409.
- [9] JIANG, A. Q., SABLAYROLLES, A., MENSCH, A., BAMFORD, C., CHAPLOT, D. S. et al. *Mistral 7B*. 2023.
- [10] LANGCHAIN, I. *Tutorial: ChatGPT Over Your Data* [online]. 2024 [cit. 2024-5-1]. Available at: <https://blog.langchain.dev/tutorial-chatgpt-over-your-data/>.
- [11] LEWIS, P., PEREZ, E., PIKTUS, A., PETRONI, F., KARPUKHIN, V. et al. Retrieval-Augmented Generation for Knowledge-Intensive NLP Tasks. *ArXiv.org*. Ithaca: Cornell University Library, arXiv.org. 2021. ISSN 2331-8422.
- [12] LIEBERGEN, K. van, CABALLERO, J., KOTZIAS, P. and GATES, C. A Deep Dive into the VirusTotal File Feed. In: *Detection of Intrusions and Malware, and Vulnerability Assessment*. Cham: Springer Nature Switzerland, 2023, vol. 13959, p. 155–176. Lecture Notes in Computer Science. ISBN 9783031355035.

- [13] MICROSOFT CORPORATION, I. *How to revert to an earlier version of Office* [online]. 2024 [cit. 2024-5-1]. Available at: <https://support.microsoft.com/en-gb/topic/how-to-revert-to-an-earlier-version-of-office-2bd5c457-a917-d57e-35a1-f709e3dda841>.
- [14] MICROSOFT CORPORATION, I. *Microsoft Mitigates Outlook Elevation of Privilege Vulnerability* [online]. 2024 [cit. 2024-5-1]. Available at: <https://msrc.microsoft.com/blog/2023/03/microsoft-mitigates-outlook-elevation-of-privilege-vulnerability/>.
- [15] MICROSOFT CORPORATION, I. *PidLidReminderFileParameter Canonical Property* [online]. 2024 [cit. 2024-5-1]. Available at: <https://learn.microsoft.com/en-us/office/client-developer/outlook/mapi/pidlidreminderfileparameter-canonical-property>.
- [16] MICROSOFT CORPORATION, I. *PidLidReminderOverride Canonical Property* [online]. 2024 [cit. 2024-5-1]. Available at: <https://learn.microsoft.com/en-us/office/client-developer/outlook/mapi/pidlidreminderoverride-canonical-property>.
- [17] MICROSOFT CORPORATION, I. *Sysmon v15.14* [online]. 2024 [cit. 2024-5-1]. Available at: <https://learn.microsoft.com/en-us/sysinternals/downloads/sysmon>.
- [18] MICROSOFT CORPORATION, I. *Update history for Microsoft 365 Apps (listed by date)* [online]. 2024 [cit. 2024-5-1]. Available at: <https://learn.microsoft.com/en-us/officeupdates/update-history-microsoft365-apps-by-date>.
- [19] SARKER, I. H. *AI-Driven Cybersecurity and Threat Intelligence*. First editionth ed. Springer Cham, 2024. ISBN 978-3-031-54496-5.
- [20] SIKORSKI, M. *Practical malware analysis : the hands-on guide to dissecting malicious software*. 1st ed. San Francisco: No Starch Press, 2012. ISBN 978-1-59327-290-6.
- [21] SOCFORTRESS.. *SocFortress Wazuh Rules* [online]. 2024 [cit. 2024-5-1]. Available at: <https://github.com/socfortress/Wazuh-Rules>.
- [22] SOCFORTRESS. *Wazuh and Chainsaw integration for near real time SIGMA detection* [online]. 2024 [cit. 2024-5-1]. Available at: <https://socfortress.medium.com/wazuh-and-chainsaw-integration-for-near-real-time-sigma-detection-6f3e729e892>.
- [23] SPELDE, K. van and SEMPLE, T. *MsgKit* [online]. 2024 [cit. 2024-5-1]. Available at: <https://github.com/Sicos1977/MsgKit>.
- [24] STATCOUNTER. *Desktop Windows Version Market Share Worldwide* [online]. 2024 [cit. 2024-5-1]. Available at: <https://gs.statcounter.com/os-version-market-share/windows/desktop/worldwide#monthly-201801-202307>.
- [25] WAZUH, I. *Getting started with Wazuh - Architecture* [online]. 2024 [cit. 2024-1-31]. Available at: <https://documentation.wazuh.com/current/getting-started/architecture.html>.

- [26] WAZUH, I. *Getting started with Wazuh - Components* [online]. 2024 [cit. 2024-1-31]. Available at: <https://documentation.wazuh.com/current/getting-started/components/index.html>.
- [27] WAZUH, I. *Getting started with Wazuh - Use Cases - Malware Detection* [online]. 2024 [cit. 2024-1-31]. Available at: <https://documentation.wazuh.com/current/getting-started/use-cases/malware-detection.html>.
- [28] WAZUH, I. *Installing Wazuh agents on Windows endpoints* [online]. 2024 [cit. 2024-5-1]. Available at: <https://documentation.wazuh.com/current/installation-guide/wazuh-agent/wazuh-agent-package-windows.html>.
- [29] WAZUH, I. *Integrations guide: Elastic, OpenSearch, and Splunk* [online]. 2024 [cit. 2024-1-31]. Available at: <https://documentation.wazuh.com/current/integrations-guide/index.html>.
- [30] WAZUH, I. *Virtual Machine (OVA)* [online]. 2024 [cit. 2024-5-1]. Available at: <https://documentation.wazuh.com/current/deployment-options/virtual-machine/virtual-machine.html>.
- [31] WHITMAN, M. E. *Principles of Information Security / : Michael E. Whitman, Herbert J. Mattord*. Sixth editionth ed. Boston: Cengage Learning, 2018. ISBN 978-1-337-10206-3.
- [32] WILLEMS, C., HOLZ, T. and FREILING, F. *Toward Automated Dynamic Malware Analysis Using CWSandbox*. LOS ALAMITOS: IEEE, 2007. ISSN 1540-7993.