DISTRIBUTED HIERARCHICAL EVENT MONITORING FOR SECURITY ANALYTICS

by

Mohiuddin Ahmed

A dissertation submitted to the faculty of
The University of North Carolina at Charlotte
in partial fulfillment of the requirements
for the degree of Doctor of Philosophy in
Computing and Information Systems

Charlotte

2024

Approved by:

Dr. Jinpeng Wei

Dr. Bei-Tseng Chu

Dr. Yongge Wang

Dr. Yasin Raja

ABSTRACT

MOHIUDDIN AHMED. Distributed Hierarchical Event Monitoring for Security Analytics. (Under the direction of DR. JINPENG WEI)

The unprecedented increase in the number and sophistication of cyber-attacks (e.g., advanced persistent threats or APTs) has called for effective and efficient threat-hunting techniques and robust security defenses. Various events (host level or network level) can be readily captured today. Analyzing such events can offer great insights into both ongoing attacks and the security posture of the system under protection. This dissertation presents a distributed hierarchical event monitoring agent architecture to facilitate two important aspects of cyber defense: efficient threat hunting and the enforcement assessment of critical security controls (CSCs).

Efficient and Scalable Threat Hunting. Although the end hosts and networking devices can record all benign and adversarial actions and use them for threat hunting, it is infeasible to monitor everything. The existing centralized threat-hunting approach continuously collects monitored logs and transfers them to the central server, which incurs high memory usage and communication overhead and thus creates scalability issues on the monitored network. Besides, single event matching on the end-host devices to detect attacks generates false alerts, causing the *alert fatigue* problem. To overcome the limitations of existing tools and research works (i.e., monitoring everything, memory requirement, communication overhead, and many false alerts), we present a distributed hierarchical monitoring agent architecture in this dissertation. This architecture detects attack techniques at the agent level, classifies composite and primitive events, and disseminates detected attack techniques or subscribed event information to the upper-level agents or managers. This solution advances the current methodologies in threat hunting through the adoption of hierarchical event filtering-based monitoring, significantly enhancing the scalability of monitoring tasks and re-

ducing memory usage and communication overhead without compromising the accuracy of the state-of-the-art centralized threat-hunting approaches. Our evaluation of both simulated attack use cases and the DARPA OpTC attack dataset shows that the proposed approach reduces communication overhead by 43% to 64% and memory usage by 45% to 60% compared with centralized threat-hunting approaches while enabling local decision-making and maintaining the same accuracy of threat-hunting by state-of-the-art centralized approaches.

CSC Enforcement Assessment. Organizations like NIST and CIS (Center for Internet Security) provide cyber security frameworks (CSF) and critical security controls (CSCs) as best practice guidelines to enforce cybersecurity and defend against attacks. These guidelines use well-defined measures and metrics to validate the enforcement of the CSCs. However, analyzing the implementations of security products to validate CSC enforcement is non-trivial. First, the guidelines are not fixed in order to adapt to the evolution of attack techniques. Second, manually developing measures and metrics to monitor and implementing those monitoring mechanisms are resource-intensive tasks and massively dependent on the security analyst's expertise and knowledge. To tackle those problems, we use large language models (LLMs) as a knowledge base and reasoner to extract measures, metrics, and detailed steps of the monitoring mechanism implementation from CSC descriptions to reduce the dependency on human expertise. Our approach used few-shot learning with chain-of-thought prompting to generate measures and metrics, and then generated knowledge prompting for metrics implementation on top of our distributed hierarchical monitoring agent architecture. Our evaluation shows that using LLMs to extract measures and metrics and monitoring implementation mechanisms can reduce dependency on humans and semi-automate the extraction process. We also demonstrate metric implementation steps using generated knowledge promoting with ChatGPT.

ACKNOWLEDGEMENTS

My gratitude extends to all the collaborators and committee members whose curiosity, insights, and support were instrumental in completing this dissertation. Special thanks to my advisor, Dr. Jinpeng Wei, for his invaluable help in broadening my understanding of research within the security field, his proactive guidance, and his commitment to fulfilling the milestones of my Ph.D. research. I am also grateful to Dr. Ehab Al-Shaer for his mentorship throughout my Ph.D. career and for providing valuable perspectives on my dissertation topics. My appreciation also goes to Dr. Bill Chu and Dr. Yongge Wang for their feedback.

Above all, I sincerely appreciate my wife, Dr. Nasheen Nur, for her reliable support and patience throughout my doctoral journey. This achievement would not have been possible without the unwavering support of my amazing family and friends.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

LIST OF ABBREVIATIONS

CA Console Agent.

CEDA Composite Event Detector Agent.

CIS Center for Internet Security.

CSC Critical Security Control.

CSF Cyber Security Framework.

EFA Event Filtering Agent.

LLM Large Language Model.

NIST National Institute of Standards and Technology.

CHAPTER 1: Introduction

Recent years have witnessed a surge in cybersecurity threats, including the emergence of advanced persistent threats (APTs) [1], characterized by a level of sophistication that is unparalleled in history [2]. The Sophos threat report indicates a significant rise in APTs and ransomware incidents, jumping from 37% in 2020 to 78% in 2021 [3]. These threats employ a variety of attack methodologies and innovative procedures, often involving multiple steps to compromise a target. For instance, one common approach involves the use of spear phishing emails [4] to gain initial access, followed by drive-by download attacks [5] to exploit vulnerabilities, and exfiltration of sensitive data from the breached system [6, 7]. Interestingly, even benign programs can be manipulated by attackers to launch these sophisticated campaigns [8]. These types of cyber attacks successfully circumvent signature-based intrusion detection systems by leveraging zero-day vulnerabilities, exploiting trusted applications, and utilizing threat emulation tools like Metasploit, Cobalt Strike, and Mimikatz. Adopting stealthy tactics, these attacks aim to remain under the radar of anomaly detection systems while pursuing objectives such as data exfiltration and encryption.

The intricate and expansive nature of organizational networks, coupled with the labor-intensive process of investigating attacks, allows attackers to maintain a presence within systems for prolonged periods. Mandiant's research highlights that the global average duration before detection of such threats is 24 days [9], with the impact on organizations increasing dramatically the longer attackers go undetected. According to IBM's security report, the financial repercussions of data breaches from ransomware attacks escalated from $3.86 million in 2020 to $4.24 million in 2021, with breaches taking an average of 287 days to be identified and contained [10]. This

lengthy detection timeframe underscores the inadequacy of traditional intrusion detection systems (IDS) in facilitating prompt and effective threat identification.

To combat these threats, an array of monitoring tools have been deployed to detect and log such malicious activities [11, 12], with the resultant data being stored as logs on the endpoint devices. Several methodologies and tools for centralized and distributed monitoring have been suggested in the literature (e.g., [13, 14, 15, 16, 17]). Despite their specific design intentions and goals, these solutions often fall short in terms of scalability for distributed environments and lack the necessary adaptability to accommodate diverse monitoring requirements. The limitations of these approaches are notable: some are designed exclusively for analyzing network traffic [15], others are tailored towards network fault diagnosis [17, 13], existing threat hunting tools are required to monitor everything in the system, and single event matching may create the *alert fatigue* problem by generating an enormous amount of false alerts. In addition to these tools, several security frameworks and guidelines, such as the NIST Cybersecurity Framework (CSF) and the Center for Internet Security (CIS) Critical Security Controls (CSC), have been developed to fortify systems against such threats. These frameworks recommend strategies such as benchmarking system configurations or analyzing system-generated event logs to ascertain the implementation and efficacy of security controls within an organization's IT infrastructure.

Critical Security Controls (CSCs) are extensively adopted by organizations of various sizes, and an expanding corpus of research addresses their application and reinforcement. A principal obstacle in deploying the CIS CSCs lies in the meticulous and thorough enforcement and implementation of these controls, a process known to be intricate and demanding substantial time investment. It is vital to have a comprehensive grasp of the controls, the procedural steps for their deployment to be established, and continuous assessment of the CSC enforcement quality.

There is a scarcity of support available to facilitate the adoption and reinforcement

of the CIS CSCs. Though the CIS offers a range of tools and resources, including a self-evaluation questionnaire, a detailed checklist, and a guide for implementation [18, 19], there has been no research on assessing the enforcement quality of those tools. Additionally, various third-party vendors provide tools and services designed to assist in the effective implementation and enforcement of these controls [20, 21]. Following the implementation of the CIS CSCs, it is imperative to undertake validation exercises such as vulnerability assessments, penetration testing, and security audits to ascertain the effectiveness and correct application of the controls. Although guidelines exist for the CSC's implementation assessment, studies focusing on the evaluation of enforcement strategies remain scarce [22].

In this dissertation, we present SCAHunter: a distributed hierarchical event monitoring approach that can be used for attack technique detection at lower level (agent level) and TTPs (Tactics, Techniques, and Procedures) detection at the higher level (manager level), and assessment of the CSC enforcement quality. Our SCAHunter detects attacks with the same accuracy as the state-of-the-art centralized threat-hunting approaches while reducing communication overhead by 43% to 64% and memory usage by 45% to 60% compared with centralized threat-hunting approaches. We also present an LLM (Large Language Model) prompting approach to automate measure and metrics generation, and measure and metrics implementation steps extraction from the CIS CSC descriptions. For the automated generation of measures, metrics, and implementation, we prompt LLM with few-shot prompting, chain-of-thought promoting, and generated knowledge prompting. Our evaluation shows that using prompt engineering to extract measures, metrics, and monitoring implementation mechanisms can reduce dependency on humans and semi-automate the extraction process., and LLM-generated measures and metrics align with human-generated measures and metrics.

1.1 Motivation

SCAHunter: Scalable Threat Hunting through a Decentralized Hierarchical Monitoring Agent Architecture (Chapter 3). Within organizations, numerous indicators of security incidents may be overlooked on a daily basis. These indicators are primarily identified through examining network behaviors or analyzing computer security event logs. It is crucial to analyze these indicators as promptly as possible to mitigate the impacts of security incidents. However, the prevalent models for centralized event monitoring and analysis impose significant demands on resources and exacerbate network communication burdens. This is due to the constant data exchange between the low-level log collection agents and the central management console. Furthermore, log management and intrusion detection systems can generate voluminous data sets. Events correlated across various devices in dispersed system locations can further complicate analysis. The necessity to monitor numerous endpoint devices, the presence of multiple sources of event generation within these devices, and their geographical dispersion in a large-scale distributed system present formidable challenges, which include enhancing performance, ensuring the monitoring system's robustness, and achieving scalability.

Prompting LLMs to Enforce and Validate CIS Critical Security Controls (Chapter 4). CIS critical security controls (CSCs) provide only guidelines to enforce cyber security. No automated enforcement or measuring mechanisms for these CSCs have yet been developed. Additionally, analyzing the implementations of security products to validate the enforcement of CSCs is infeasible. Therefore, it is quintessential to develop formal- and data-driven approaches and automated tools to measure the effectiveness and validate the enforcement of CSC deployment. We can formulate the problem in the following way: a company X has invested in Y products to implement Z CSCs. Our goal is to identify metrics and measurement procedures to test and evaluate the quality of CSC enforcement by these products quantitatively.

1.2 System Overview

In this section, we present the overview of our whole framework, as shown in Figure 1.1. The framework consists of the following modules: Event Subscription Policy Rule Generation, Distributed Hierarchical Agent Monitoring System, Prompt Engineering to Extract Measures and Metrics, and Prompt Engineering to Extract Measures and Metrics Implementation. A brief overview of each module is given below.

Event Subscription Policy (ESP) Rule Generation (Chapter 3). To support monitoring tasks, we propose an analytical language that will be used in end-host devices to subscribe for event logs from themselves or other reachable hosts. This language also provides support for the correlation of collected logs. A threat hunter first derives the attack signature from the attack technique description provided in the MITRE ATT&CK framework and threat reports of interest. Then, the threat hunter maps the attack signature to the Event Subscription Policy (ESP) rule by using our analytical language (more details in Chapter 3).

Distributed Hierarchical Agent Monitoring System (Chapter 3). The distributed hierarchical event monitoring system will take the ESP rule as a subscription task, decompose it into primitive event monitoring sub-tasks, and distribute them to the lower-level agents. This monitoring system consists of three types of agents: Event Filtering Agent, Composite Event Detector Agent, and Console Agent. This event monitoring system is used for cyber threat hunting and CSC validation (collecting statistics about specific measures).

- Event Filtering Agent (EFA). EFA monitors different data sources for events requested in the received event subscription request. Those agents are static (we generate them initially) and continue to work until they are terminated or subscription requests are deleted.

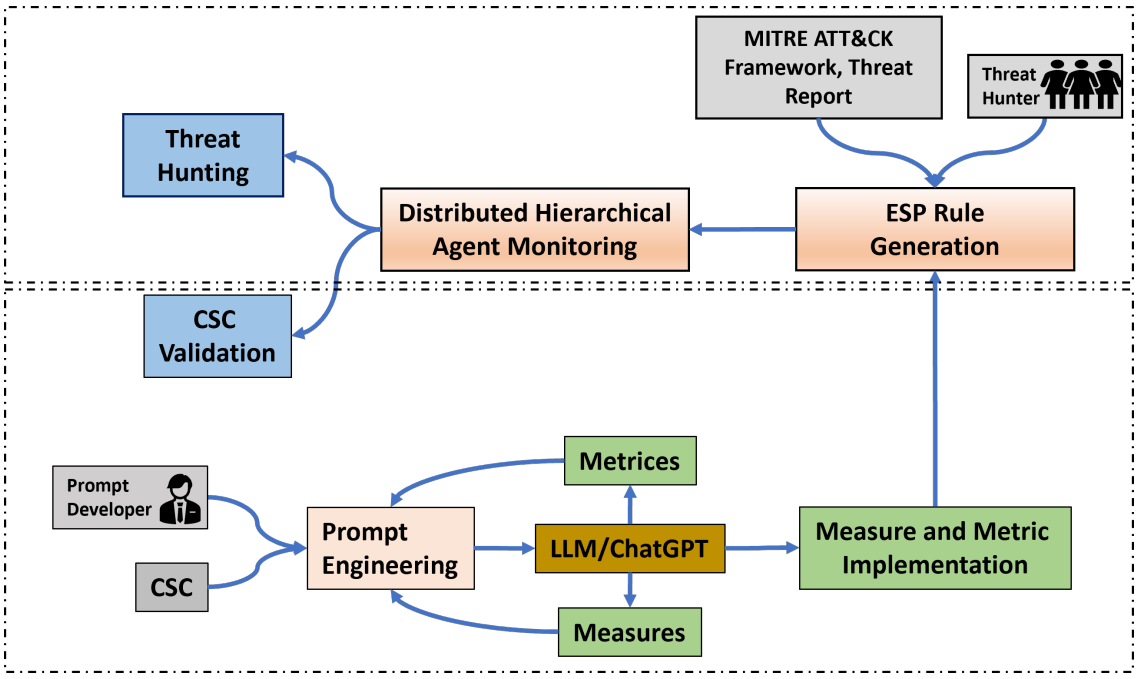

Figure 1.1: System Overview

- Composite Event Detector Agent (CEDA). A CEDA correlates or detects different events based on the composite event mentioned in the event subscription tasks (ESP Rules). The hierarchical agent architecture can have multiple levels of CEDAs. The CEDAs will be generated dynamically based on the subscription tasks (ESP Rules).

- Console Agent (CA). The CA is the main entry point of our proposed agent monitoring architecture. It takes subscription tasks (ESP Rules) from the user, decomposes the task into composite events and primitive events, and generates appropriate CEDAs and configurations for each agent.

Prompt Engineering to Extract Measures and Metrics (Chapter 4). In order to assess the enforcement quality of a CSC safeguard, we need a list of measures and metrics where a measure is a concrete and objective attribute, and a metric is an abstract and subjective attribute calculated from one or multiple measures. We leverage the power of Large Language Models (LLMs) to automatically generate

the measures and metrics given a CSC safeguard description. Specifically, we first generate a CSC ontology by using chain-of-thought prompting and then use it to generate the list of measures and metrics for the corresponding safeguard by using zero-shot and few-shot prompting.

Prompt Engineering to Extract Measures and Metrics Implementation (Chapter 4). The measures and metrics generated in the previous module are not implementable because they require security analyst's help to determine specific data sources and attributes to measure. To remove the need for expert knowledge, we perform additional generated knowledge prompting to generate measure and metric implementation steps. Then, we translate the implementation steps to ESP rules which will be used in a hierarchical monitoring system to collect corresponding statistics about the metric to validate the corresponding CSC safeguard.

1.3 Research Challenges

This dissertation addresses the following challenges to achieve our research goals.

- **On-demand Monitoring:** Current studies [23, 24, 25] have aimed at maximizing data visibility by monitoring an extensive array of sources, an approach that is not always requisite for identifying TTPs. For instance, in the context of detecting malware execution via PowerShell using an Endpoint Detection and Response (EDR) solution, it is not essential to monitor additional data streams such as registries, processes, or file activities. Focusing solely on PowerShell command activity is sufficient for the detection of PowerShell execution TTPs [26].

- **Event Storage and Communication Overhead:** The process of centralized threat hunting involves the persistent aggregation of monitored logs on a central server, leading to significant memory consumption and increased communication overhead for event transmission. Such a methodology poses scalability

challenges within the network being monitored.

- **Efficient Event Correlation:** To identify attacker TTPs, current methodologies and investigations [27] employ a strategy of matching individual events on endpoint devices to trigger alerts. However, this approach of matching single events tends to produce a high volume of false alerts, leading to a phenomenon known as *alert fatigue* within the realm of threat hunting [28]. For instance, both malicious actors and legitimate users may utilize the TTP of executing commands through the Windows command shell to run an executable on the system. Relying solely on matching these single events for detection will result in numerous erroneous alerts.

- **Manual Measures and Metrics Development for CSC Safeguards**: The continual evolution of cyber threats necessitates frequent updates to critical security controls (CSCs), which require the repetitive manual task of extracting measures and metrics to align with newly introduced controls. Additionally, the development of manual measures and metrics heavily relies on security analysts' expertise and prior knowledge, introducing a significant dependence on their skills.

1.4 Our Contributions

In this dissertation, to solve the challenges mentioned in section 1.3, we make the following contributions:

- To overcome the challenges (monitoring everything, memory requirement, communication overhead, and many false alerts) of existing threat-hunting tools and research works,
  - We provide a distributed hierarchical monitoring agent architecture that optimizes monitoring tasks to reduce resource usage and communication overhead.

  - We provide an approximation algorithm to generate a near-optimal agent hierarchy, so that event correlation tasks are distributed among the hosts.
  - We develop an ETW-based agent to monitor signature-specific events so that on-demand monitoring is supported.
  - We demonstrate the threat-hunting process using our proposed agent architecture. We evaluated our proposed architecture using log data generated by running three test scripts provided by Red Canary Atomic Red Team [29], and we created attack signatures for the test scripts following the MITRE ATT&CK technique description during the evaluation. We also evaluated our proposed approach using DARPA OpTC attack dataset [30]. To compare our approach with the existing centralized event monitoring approaches for threat hunting, we also implemented centralized event monitoring using Splunk.

- To solve challenges of automating measures and metrics development and reducing dependency on security analyst's expertise and prior knowledge, we make the following contributions:
  - We propose a CSC safeguard ontology for the things to be extracted from each safeguard description. We provide a prompting template used to extract CSC safeguard ontology where CSC ontology will help develop a chain-of-thought (CoT) prompt to extract implementation steps for a CSC safeguard enforcement.
  - We provide a few-shot prompt to extract measures and metrics given the safeguard description and dependent safeguard. This prompt generates new measures and metrics for safeguard enforcement compliance and safeguard enforcement quality.
  - We provide a prompting template for evaluating LLM-generated measures

and metrics to reduce human labor on manual evaluation where a different LLM is used for evaluation. With the help of Spearman, Pearson, and Kendall Tau's correlation coefficient value, we showed that the LLM evaluation aligns with human evaluation.

- We demonstrate CSC safeguard enforcement implementation for multiple measures and metrics of a safeguard with the help of CoT prompting and generated knowledge prompting.

The dissertation is based upon the following papers:

- Mohiuddin Ahmed, Jinpeng Wei, Ehab Al-Shaer. 2023. "SCAHunter: Scalable Threat Hunting Through Decentralized Hierarchical Monitoring Agent Architecture." In: Arai, K. (eds) Intelligent Computing. SAI 2023. Lecture Notes in Networks and Systems, vol 739. Springer, Cham. https://doi.org/10.1007/978-3-031-37963-5_88.

- Mohiuddin Ahmed, Ehab Al-Shaer. 2019. "Measures and Metrics for the Enforcement of Critical Security Controls: a Case Study of Boundary Defense." In Proceedings of the 6th Annual Symposium on Hot Topics in the Science of Security (Nashville, Tennessee, USA) (HotSoS '19). Association for Computing Machinery, New York, NY, USA, Article 21. https://doi.org/10.1145/3314058.3317730.

- Mohiuddin Ahmed, Jinpeng Wei, and Ehab Al-Shaer. 2024. Prompting LLM to Enforce and Validate CIS Critical Security Control. In Proceedings of the 29th ACM Symposium on Access Control Models and Technologies (SACMAT 2024), May 15-17, 2024, San Antonio, TX, USA. ACM, New York, NY, USA. https://doi.org/10.1145/3649158.3657036.

1.5 Thesis Outline

The remainder of the dissertation is organized as follows:

Chapter 2 reviews critical background knowledge crucial to understand the dissertation, including existing event monitoring tools and approaches, cyber threat hunting using MITRE ATT&CK framework, security best practices: CIS critical security controls, and prompt engineering to extract information from text descriptions.

Chapter 3 presents SCAHunter, a distributed hierarchical agent monitoring architecture that is used for efficient and scalable cyber threat hunting.

Chapter 4 presents our manual and prompting approaches with LLMs to extract measures and metrics and corresponding implementation steps to assess the enforcement of security best practices.

Chapter 5 summarizes our contributions, findings, and limitations.

CHAPTER 2: Background Knowledge

In this chapter, we present existing tools and approaches used in agent monitoring and threat hunting. Moreover, we also provide an overview of existing CSC validation tools.

2.1 Event Tracing for Windows (ETW)

ETW [31] provides a mechanism to collect and store events generated by user-mode applications and kernel-mode drivers. Windows OS provides ETW as a fast, reliable, versatile event-tracing feature. Similar logging mechanisms exist in other operating systems, such as audit.d for Linux systems. In this dissertation, we use ETW for event collection from Windows OS. ETW consists of four components: 1) ETW Provider, 2) ETW Consumer, 3) ETW Session, and 4) ETW Controller, as shown in Figure 2.1. ETW Provider is the conceptual agent responsible for generating and writing events into an ETW Session. When integrating a software component with ETW, an ETW Provider is established to detail the events it generates. During registration, the ETW Provider assigns a unique provider ID to ETW. After registering an ETW provider

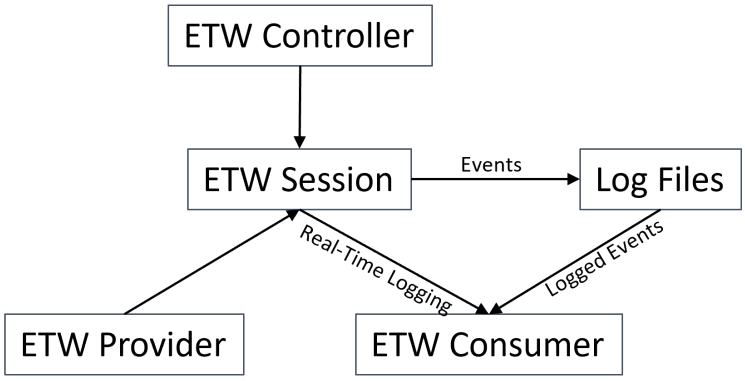

Figure 2.1: ETW Architecture

with the ETW Session, an ETW Controller can be used to enable or disable event tracing from a specific ETW Provider.

The ETW Session accepts and buffers events received from the registered ETW Provider. Typically, it generates a trace file to record these events and can simultaneously transmit them in real-time to consumer applications. To manage the flushing of buffer data to the ETW log file and ETW Consumer, a dedicated write thread is activated within the ETW Session.

ETW Controller is an application that orchestrates ETW provider, ETW Session, and ETW Consumer.

An ETW Consumer pulls events in real time, either from an ETW Session or log files. It can pull events from multiple ETW sessions concurrently.

2.2 Centralized Log Monitoring Agent Infrastructure and SPLUNK

A log management infrastructure typically comprises three modules: log generation, log analysis and storage, and log monitoring. The log generation module involves hosts making their logs available to log servers in the second tier. This is performed in two different ways. The exact method depends on the log type and the host and network controls. In one way hosts run some services to send their log data over the network to log collection servers. Alternatively, hosts allow the log servers to pull the log data from them. The logs are often transferred to the log receivers either in a real-time or near-real-time manner or in occasional batches based on a schedule.

The log analysis and storage module is composed of one or more log servers receiving log data from the hosts. These log receivers are also called collectors or aggregators. To facilitate log analysis, automated methods of converting logs from multiple formats to a single standard format need to be implemented. Syslog format of logging is often used for this purpose.

The log monitoring module contains consoles for monitoring and reviewing log data and the results of automated analysis. Consoles may also be used for report

generation, management dashboards, and log baselines as part of this tier.

Log management infrastructures typically perform several functions that assist in the storage, analysis, and disposal of log data. These functions are normally performed in such a way that they do not alter the original logs. General functions of log management infrastructure include log parsing, event filtering, and event aggregation. On the storage side, log management has to provide for log rotation, log archival, log compression, log reduction, log conversion, log normalization, and log file integrity.

Event correlation, log viewing, and log reporting are some of the analysis functions of a log management infrastructure. Security information and event management (SIEM) software such as SPLUNK provides the log management infrastructure encompassing log analysis, storage, and monitoring tiers. What sets SIEM products (e.g., SPLUNK) apart from traditional log management software is the ability to perform event correlation, alerting, incident management, reporting and forensic investigation based on event analysis. There are many SIEM solutions commercially available today, and these solutions provide different sets of features and additional add-ons. SIEM Technology (e.g., SPLUNK) aggregates the event data produced by security devices, network devices, systems, and applications. The primary data source is log data, but SIEM technology can also process other forms of data. Event data combines contextual information about users, data, and assets. The data is normalized so that events from disparate sources can be correlated and analyzed for specific purposes, such as network security event monitoring, user activity monitoring, or compliance reporting. The technology provides real-time security monitoring, historical analysis, and other support for incident investigation and compliance reporting.

2.3 MITRE ATT&CK Framework

MITRE ATT&CK [26] is an open-source knowledge base and public effort to categorize and discover new attacker action execution procedures, intents, and final goals. There are 156 techniques, 261 sub-techniques, and 12 tactics. Every technique de-

scribes one or multiple ways of achieving a specific capability or goal. Every tactic defines what capability or goal an attacker is trying to achieve by executing corresponding techniques or sub-techniques.

2.4 Critical Security Controls

The Center for Internet Security (CIS) [32] publishes 20 critical security controls (CSCs), which contain a total of 171 sub-controls. According to CIS, these CSCs provide guidelines not only to block the initial compromise of the cyber systems but also to detect already compromised systems and prevent or block post-compromise adversarial actions. These guidelines also provide ways to reduce the attack surface by hardening device configurations, identifying compromised systems, and disrupting the attacker's command and control communications. These CSCs are now widely used by industry and government organizations to enhance and enforce cyber security. There are many solution providers who claim to implement these top 20 CSCs. For instance- Rapid7 global service [20] implements the top 20 CSCs and is verified by SANS as a top solution provider. AlienVault [33] is also known as a top solution provider for top 20 CSCs. Moreover, a genuine effort has been made by the NSA, NIST, and other cyber security counsels to establish these CSCs as standards or best practices for enforcing cybersecurity.

CIS provides CIS Benchmark [32] tools as a prescriptive configuration recommendation for more than 25 vendor product families. They represent the consensus-based effort of cybersecurity experts globally to help protect systems against threats more confidently. They also provide CIS-CAT [32] as a tool to assess the CIS benchmark and CIS-CSAT [32] to track the implementation of the CSC. Those tools check specific system configurations to validate a benchmark; they also map the benchmark to CSC. Though verifying a configuration may provide a quantitative measure of CSC implementation, no qualitative approach to assess the effectiveness of CSC implementation has been developed yet.

2.5 LLM and Prompt Engineering

Large Language Models (LLM) are pre-trained deep learning models in which a transformer neural network consisting of an encoder and decoder with self-attention is used as the underlying neural network architecture. Those models are trained on vast amounts of data, enabling them to perform a range of Natural Language Processing (NLP) tasks, such as question answering, classification, summarization, text understanding, text generation, and reasoning. Since LLMs are trained with vast amounts of data, one of the emergent abilities of LLMs is in-context learning, which can be used for question-answering and reasoning [34].

Prompt Engineering is the process of guiding LLMs to generate desired outputs. Even though an LLM attempts to mimic humans, it requires detailed instructions to create high-quality and relevant output. In prompt engineering, the prompt engineer chooses the most appropriate formats, phrases, words, and symbols that guide the LLM in interacting with users more meaningfully. A prompt is a task instruction described in natural language text that requests the LLM to perform a specific task. To get the desired output from the LLM, the following prompt development trends are encouraged [35]: 1) Use low-level pattern: instead of using terms that require background knowledge to understand, use various patterns about the expected output; 2) Itemize instruction: turn descriptive attributes into bulleted lists. If there are any negation statements, turn them into assertion statements; 3) Break down a task into multiple simpler tasks; 4) Enforce constraint: add explicit textual statements of output constraints. and 5) Specialize the instruction: customize the instructions so that they directly speak to the intended output.

Zero-shot prompting or direct prompting is a form of prompting where no examples or demonstrations of the task are provided. Rather, only instruction about a task is provided to the LLM. Fine-tuned language models are zero-shot learners [36].

Few-shot prompting is a way to elicit responses from LLMs. A few examples of

questions and corresponding answers are fed to the LLM, and the prompter asks the LLM to answer the next question by following the examples. According to [36, 37], LLM performance is improved if a demonstration of the task is given to the model in addition to the task-specific instructions.

Chain-of-Thought Prompting (CoT) is a form of few-shot prompting for reasoning tasks where a prompt consists of a triplet <input, chain of thought, output> that is presented to the LLM as a question. This CoT prompting approach divides the whole prompt into a series of intermediate reasoning steps that lead to the final output. According to Wei *et al.* [34], CoT prompting significantly improves the complex reasoning capabilities of LLMs.

Generated Knowledge Prompting is a form of prompting where knowledge is generated and extracted from the LLM first, then the generated knowledge is provided as additional input to the LLM to answer a question. According to [38], generated knowledge prompting facilitates common sense reasoning tasks in LLMs.

In this dissertation, we used LLM and prompt engineering extensively to extract critical entities and knowledge from security guidelines.

CHAPTER 3: SCAHunter: Scalable Threat Hunting through Decentralized Hierarchical Monitoring Agent Architecture

This chapter presents a scalable, dynamic, flexible, and non-intrusive monitoring architecture for threat hunting. The agent architecture detects attack techniques at the agent level, classifies composite and primitive events, and disseminates seen attack techniques or subscribed event information to the upper-level agent or manager. The proposed solution based on our published works [39] offers improvement over existing approaches for threat hunting by supporting hierarchical event filtering-based monitoring, which improves monitoring scalability. It reduces memory requirement and communication overhead while maintaining the same accuracy of threat hunting in state-of-the-art centralized approaches.

3.1 Introduction

In recent years, there has been an increase in cyber attacks including advanced persistence threats (APTs) and ransomware[1], and the techniques used by the attacker have reached an unprecedented sophistication [3]. According to Sophos threat report [3], APT and ransomware attacks increased from 37% in 2020 to 78% in 2021. These attacks evade signature-based intrusion detection systems by exploiting the zero-day vulnerability, whitelisted applications and threat emulation tools (Metasploit, Cobalt Strike, Mimikatz). They use a low and slow approach to avoid triggering anomaly detection while working on the attack goals such as exfiltration and encryption. Due to the diverse and sprawling nature of organizational network, and time-consuming nature of attack investigation, attackers can dwell in the system for extended periods. Mandiant reports that the global average dwell time of the adver-

sary is 24 days [9]. The damage incurred by the adversary on an organization increases exponentially with increasing dwell time. According to the IBM security threat report [10], data breach damage from ransomware attacks increased from $3.86 million in 2020 to $4.24 million in 2021, and the time to identify and contain the data breach is, on average, 287 days. The high threat detection time indicates that traditional IDS does not make the breakthrough in real-time threat hunting.

Considering the shift in threat actors and unprecedented sophistication in adversary activities, the organization deploys Endpoint Detection and Response (EDR) solutions and System Information and Event Management (SIEM) solutions to record, monitor continuously, and analyze low-level system logs in end-host devices. The EDR solutions detect threats by matching low-level system events against a knowledge base of adversarial TTPs (Tactics, Techniques, and procedures). MITRE ATT&CK framework [26] provides a knowledge base of TTPs developed by domain experts by analyzing real-world APTs. The SIEMs collect low-level system logs or alerts through a collector, sensor, or EDR agent installed in the end-host devices to the manager (central server). A threat hunter uses SIEM to analyze and correlate collected logs to detect adversary activities during the threat hunting process proactively. While such centralized event correlation facilitates causality analysis of attacker activities, it presents the following challenges in threat hunting at a large-scale distributed system:

- **On-demand Monitoring:** Existing researches [23, 24, 25] try to monitor everything to give data visibility as much as possible, which is not necessary for detecting a TTP. For example, while an EDR solution tries to detect PowerShell execution of malware, it is not necessary to monitor other data sources (registries, processes, file operations); instead, monitoring the PowerShell command is enough to detect PowerShell execution TTPs [26].

- **Event Storage and Communication Overhead:** The centralized threat hunting process continuously collects monitored logs to the central server, which

incurs high memory usage and communication overhead to transfer events to the central server. This approach introduces scalability issues on the monitored network.

- **Efficient Event Correlation:** To detect attacker TTP, existing solutions and research [27] use single event matching on the end-host devices and generate alerts. Unfortunately, such a single event matching approach generates many false alerts, causing the alert fatigue problem [28] in threat hunting. For example, adversary and benign users can use Windows command shell execution TTP to execute an executable on the system. Detection based on the single event matching will generate many false alerts.

Recent works on threat hunting use causality analysis [40, 41, 24, 25], cyber threat intelligence [23], and MITRE ATT&CK technique detector [23] to reduce mean-time-to-know during the post-breach threat hunting process. These causality analysis approaches incrementally parse low-level audit logs generated by system-level logging tools (e.g., Sysmon, Event Tracing for Windows, and auditd) into causal graphs (provenance graphs). The causal graph encodes the dependency between subjects (processes, threads) and objects (files, registries, sockets) to provide the historical context the threat hunter needs to correlate attacker activities and understand alerts. In [40, 42], the authors developed detectors or rules for attack techniques and mapped each detector/rule to the MITRE technique. According to a recent survey about EDR solutions by Gartner, all top 10 EDR solutions use MITRE ATT&CK framework to detect adversary behaviors [43]. Such causality analysis is promising for network-wide alert correlation and cyber intelligence.

However, the performance of causality analysis is a limiting factor for real-time threat hunting because of the significant graph construction time ranging from hours to days [25] and the large size of the audit logs (terabytes of logs generated within week [25]). To improve the graph construction time, prior works [40, 41] applied

different graph reduction and compression techniques. In [25], the author reduces memory usage during causality analysis by storing the most recent part of the causal graph on the main memory and the unused casual graph on disk. In [24], the authors generate a host-specific causal graph and network-specific causal graph and perform multi-host analysis only if any host-based sub-graph crosses a predefined risk score. Though graph reduction and compression and hierarchical storage reduce memory usage to store monitored events, the causality analysis on the compressed graph has the following limitations. Prior works perform centralized analysis and want to give data visibility as much as possible. Thus, they try to monitor all data sources in the end-host devices, which raises the issue of monitoring scalability and communication overhead (collecting logs on a central server) in threat hunting. Provenance graph expansion for multi-host analysis in Holmes [40] and Steinerlog [24] creates dependency expansion. Graph alignment in multiple hosts [23] increases the threat detection time exponentially with the increase of event logs and network size.

To overcome the limitations (monitoring everything, memory requirement, communication overhead, and many false alerts) of existing tools and research works, we propose a monitoring architecture (Figure 3.1) using a hierarchical event filtering approach that reduces monitoring load and communication overhead and provides efficient event correlation that can potentially reduce false alerts. The adversary activities follow precedence, meaning that most of the attack techniques have preconditions [44], which are other attack techniques. For example, without performing the initial compromise or execution technique, an attacker will not be able to perform the discovery and command and control technique. Similarly, an attacker cannot perform a collection or exfiltration technique without performing a discovery technique. Following the attack technique association, the SCAHunter provides on-demand monitoring of the data sources corresponding to the monitored attack signature. We provide on-demand monitoring by instrumenting ETW (event tracing

for Windows) through ETW API so that the lower-level agents only log signature-specific events. Similarly, auditd for Linux and Endpoint Security for Mac OS can be used for providing signature-specific on-demand monitoring.

Additionally, events/logs can be correlated in the end host devices if monitored events or logs correlate with them. If a set of events from a set of different hosts are required for the correlation of a monitored signature, a middle-level host nearby (in terms of hop-count) the corresponding monitored hosts can be used for the correlation task. Event correlation at the intermediate host will reduce the memory requirement and communication overhead since only the correlated events will be forwarded to the upper-level agents or manager. It will improve scalability and performance by using hierarchical monitoring and distributed correlation while reducing the monitoring intrusiveness.

Finally, hierarchical event filtering will reduce the number of false alert generation problems in the current research works. Single event matching will generate many false alarms because of similarity with benign user activities. However, it is highly unlikely that a sequence of adversary TTPs will match with benign user activities. For example, execution of payload through services.exe or sc.exe (T1569.002) [26] can be used by the benign user; however, remote execution of payload through sc.exe is highly suspicious behavior. Our proposed hierarchical filtering correlates service creation and remote execution in a middle host, thus generating less number of alerts by distributed event correlation. However, generating the agent hierarchy is an NP-hard problem, which we solve using an approximation algorithm (Algorithm 1) based on the geographical host distribution and predefined monitoring capacity of agents. Our proposed approach does not monitor every data source; instead, it monitors what is required to detect the current attack stage and adds a new monitoring task on-demand based on the threat hunting progress.

Contribution. Our first contribution is to provide a distributed hierarchical mon-

itoring agent architecture that optimizes monitoring tasks to reduce resource usage and communication overhead. Our second contribution is to provide an approximation algorithm to generate a near-optimal agent hierarchy, so that event correlation tasks are distributed among the hosts. Our third contribution is to develop an ETW-based agent to monitor signature-specific events so that on-demand monitoring is supported. Our last contribution is to demonstrate the threat hunting process using our proposed agent architecture. We evaluated our proposed architecture using log data generated by running three test scripts provided by Red Canary Atomic Red Team [29], and we created attack signatures for the test scripts following the MITRE ATT&CK technique description during the evaluation. We also evaluated our proposed approach using DARPA OpTC attack dataset [30]. To compare our approach with the existing centralized event monitoring approaches for threat hunting, we also implemented centralized event monitoring using Splunk.

This chapter is organized as follows: Section 3.2 surveys existing research work on threat hunting and intrusion detection system, Section 3.3 formalizes signature generation and scalable threat hunting with SCAHunter, Section 3.4 explains SCAHunter by describing each component of the system, attack signature decomposition algorithm, an approximation algorithm to generate near-optimal agent hierarchy used by the agent architecture for subscribe-publish based event monitoring and correlation, and also provides a threat hunting demonstration using the SCAHunter, Section 3.5 provides implementation details and evaluation with simulated attack use cases and OpTC attack dataset, and Section 3.7 summarizes our contributions and future research tasks. The remainder of this chapter will use logs, alarms, and events interchangeably. It will also interchangeably use monitoring tasks, composite events, and subscribed events.

3.2 Related Works

Causality Analysis. Sleuth [41], DeepHunter [45], Nodoze [27], OmegaLog [46] and CoNAN [47] used provenance graph generation and centralized analysis on the aggregated logs for attack detection and investigation. Holmes[40] uses correlation among information flow to detect an APT, which can be possible only if the threat hunter aggregates events before performing correlation. Domino[48] combines alerts from different NIDS to detect attacks globally, using a single hierarchy level, i.e., manager-agent architecture. Kelifa et al. [49] proposed a misbehavior detection mechanism for wireless sensor network (WSN) based on clustered architecture where a cluster head is selected based on static metrics monitored by the monitoring nodes. Collaborative IDS (CIDS) [50] aggregate alerts from lower-level IDS to manager IDS, and the manager performs graph-based and network-based analysis to detect intrusions. All of those researches aggregate logs in a central server and perform corresponding analysis, which requires monitoring of all events and incurs communication overhead to transfer the generated events to the manager.

In Swift [25], the author reduces memory usage during causality analysis by storing the most recent part of the causal graph on the main memory and the unused casual graph on disk. In [24], the authors generate a host-specific causal graph and network-specific causal graph and perform multi-host analysis only if any host-based subgraph crosses a predefined risk score. Though graph reduction and compression and hierarchical storage reduce memory usage to store monitored events, the causality analysis on the compressed graph has the following limitations. They try to monitor all data sources in the end-host devices, which raises the issue of monitoring scalability and communication overhead (collecting logs on a central server) in threat hunting. Provenance graph expansion for multi-host analysis in Holmes [40] and Steinerlog [24] creates dependency expansion. Graph alignment in multiple hosts [23] increases the threat detection time exponentially with the increase of event logs and network size.

Event Monitoring. Several centralized and distributed monitoring approaches and tools have been proposed (e.g., [51, 52, 53, 54, 55, 56]) in other domains. Although they have various design-specific goals and objectives, they are not scalable for distributed monitoring, lack the flexibility to express the monitoring demands, and require monitoring every data sources. Those proposed approaches either monitor only network traffic [57], apply to network fault diagnosis [51, 58], or maintain a static agent hierarchy [57]. Though hierarchical monitoring systems exist in other domains (fault detection, malicious sensor node detection), they are not suitable for threat hunting since those systems are suitable for a specific use case.

Adversarial Tactics, Techniques and Procedures. MITRE ATT&CK framework [26] published a public knowledge base of TTPs consisting of 24 tactics, 188 techniques, and 379 sub-techniques used by the APT. Every MITRE ATT&CK tactic published is a high-level attacker goal in a specific kill chain phase. Every MITRE ATT&CK technique consists of one or more procedures the attacker can use to achieve a specific goal, whereas each procedure is a unique way to achieve the corresponding goal. MITRE ATT&CK framework also provides 129 APT groups and a subset of the publicly reported technique used by each APT Group. MITRE also provides the data source to monitor to detect a specific attack technique. Holmes [40] uses MITRE ATT&CK TTP to build a set of rules and perform rule matching on the collected logs. This approach will fail if the initial compromise is not detected and they use centralized analysis, which incurs high memory and communication overhead. Additionally, they analyze alerts after generation, whereas SCAHunter reduces the number of alerts generated by hierarchical filtering.

3.3 Problem Formalization

To formulate the scalable threat hunting with hierarchical agent architecture, we formulate the attack signature and agent hierarchy generation in this section. We formalize attack signatures and event attributes and values, events, event subscription

predicate, and event subscription rule (attack signature).

Basic Notation for Events, Attributes, and Subscriptions. In distributed event monitoring systems, event producers (application collecting event logs from specific data source) frequently generate events to report aspects of the monitored system state. Let's represent events reported by producers as $E = \{e_1, e_2, .., e_n\}$. Each event e_i consists of a set of attributes $a_{i,j}$ that has assigned values $v_{i,j}$ such that i and j represent the event and attribute indices, respectively. For example, the event e_i that has M attributes is defined as follows: $e_i = \{(a_{i,1}, v_{i,1}), (a_{i,2}, v_{i,2}), .., (a_{i,M}, v_{i,M})\}$. Event consumers may submit multiple subscriptions s_i to request monitoring and reporting of the occurrence of specific event instances or correlation of event instances. The set of consumers' subscriptions can be represented as $S = \{s_1, s_2, .., s_n\}$. Each subscription s_i consists of a *logical expression* on *event attributes*. For example, if a *CA* subscribes with the following, $S_i = \{(pName, "cmd.exe") \wedge (isChild, true)\}$, the *CA* is asking for subscription of all *cmd.exe* processes that are children of another process.

Formulation of Event Fragmentation. By following the formulation of events and attributes, an occurrence of event i at time t is defined as combination (conjunctive) of attribute values as follows:

$$\begin{aligned} e_i^t = (a_{i,1} = v_{i,1}^t) \wedge (a_{i,2} = v_{i,2}^t) \wedge, .., \\ \wedge (a_{i,M} = v_{i,M}^t) = \bigwedge_{j=1}^{M} a_{i,j} = v_{i,j}^t \end{aligned} \tag{3.1}$$

If we receive a sequence of events of the same event type c within a specified interval T, the event history h_{cl} of event type c during the interval T can be formally represented as follows:

$$h_{cl} = \bigwedge_{l \in L} \bigvee_{t,k \in T} e_{l-1}^t e_l^k \qquad where\ l \neq 1,\ t < k \tag{3.2}$$

which can be further formalized using the event attributes and values as follows:

$$h_{cl} = \bigwedge_{l \in L} \bigvee_{t,k \in T} \bigwedge_{j \in M} (a^t_{l-1,j} = v^t_{l-1,j})(a^k_{l,j} = v^k_{l,j}) \quad where\, l \neq 0,\ t < k \tag{3.3}$$

L is the number of events in the event sequence requested, M is the number of attributes in the event type c. For $l = 1$, Equation 3.2 and Equation 3.3 are reduced to the following:

$$h_{c1} = \bigvee_{t \in T} e^t_1 \tag{3.4}$$

$$h_{c1} = \bigvee_{t \in T} \bigwedge_{j \in M} (a^t_{1,j} = v^t_{1,j}) \tag{3.5}$$

For instance, a consumer requests for type c event sequence $\{e_1, e_2, e_3\}$ in time interval $T = [1 - 10]$ and the producer reported events e_1 at $t = 2$, e_2 at $t = 6$ and e_3 at $t = 9$, then the equations 3.2 and 3.3 will be evaluated true. However, if either any of the three event is not reported or their reporting times are not in order with the sequence, the equations 3.2 and 3.3 will be evaluated false.

Formulation of Event Subscription Predicate. An event subscription predicate is a set of *logical predicates* that matches the *attribute values* of the one or more event instance occurrences. Therefore, we define the event predicate as a logical expression defined by the user that will be evaluated based on the attribute values of the event occurrences. Each predicate will be evaluated to `true` if and only if the event attribute values satisfy the predicate logical expression. For example, the predicate $p_1 : (pName, =, "cmd.exe")$ will evaluate to `False` if the event generated is not corresponding to the cmd.exe process, but the predicate, $p_2 : (memAllocated, >, 65)$ will evaluate to `True` if the event is corresponding to a memory allocation and a memory of size greater than 65B is allocated. Therefore, we can formally define the event predicate as follows: $p_{ijk} = (a_{ij}, op, v_k)$, where a_{ij} is the attribute j of event i, v_k is

any value of type integer or string that can match the value of the attribute, and *op* is a logical operator (such as $=, <, >, \leq, \geq$) or set operator (such as $\supset, \not\subset, \subseteq, \not\subseteq$, etc). Semantically, if *op* is $=$, then $p_{ijk} \Leftrightarrow a_{ij} = v_k$.

A predicate can specify a relationship between (same or different) attributes of same or different events to detect the occurrence of an *event correlation*. Formally, a predicate that defines a relationship between attribute k in two different events, i and j, can be specified as follows: $p_{ijk} = (a_{ik}, op, a_{jk})$, where a_{ik}, and a_{jk} is the attribute k of events i and j, respectively.

Formulation of Event Subscription Rule. By following the formulation of event subscription predicates, a user can define an Event Subscription Rule (ESR) as the logical Boolean expression of a conjunctive normal form (CNF) using multiple predicates to match an attack signature (subscription) occurrence, S_i, as follows:

$$\begin{aligned} ESR(e_1, .., e_n) = (p(e_i) \vee .. \vee p(e_j)) \wedge (p(e_i) \\ \vee .. \vee p(e_j)) \wedge .. \wedge (p(e_i) \vee .. \vee p(e_j)) \end{aligned} \tag{3.6}$$

which can be formalized in the concise format as follows:

$$ESR(e_1, .., e_n) = \bigwedge_{i \in \mathbb{N}: i \leq n} \bigvee_{j \in \mathbb{N}: j \leq n} p(e_j) \tag{3.7}$$

Formulation of Event Correlation Subscription Rule. The user can also define a relationship that describes the correlation of multiple events. In this case, the user is interested to be notified iff a set of correlated target events are detected within a time window. The subscription of the Event Correlation Rule (ECR) can be formally defined as a conjunctive normal form (CNF) as follows:

$$\begin{aligned} ECR(e_1, .., e_n) = (ESR(e_i) \vee ESR(e_j)) \wedge (ESR(e_i) \\ \vee ESR(e_j)) \wedge .. \wedge (ESR(e_i) \vee ESR(e_j)) \end{aligned} \tag{3.8}$$

This formulation offers the flexibility to define any general ECR subscription request to define an arbitrary logical relationship between event occurrences. Note that, in the simplest form, an ECR might contain a single event detection.

Formulation of Event Subscription Policy. An Event Subscription Policy (ESP) is defined as a list of ECR rules sequentially evaluated according to their priorities based on receiving events to classify and forward notifications to subscribing consumers (CA). If more than one rule is triggered, all corresponding subscription actions will be taken.

Furthermore, ESP of n ECR rules can be formally defined as

$$ESP = \bigvee_{i=1}^{n} ECR_i \quad (3.9)$$

The above formulation offers the maximum flexibility in defining subscriptions for events. It allows specifying various arbitrary lengths of conjunctive predicates between event attributes and any arbitrary disjunctive relation between the predicates.

Therefore, given the attack signature or subscription request S_i as ESR and a network topology, the goal of the hierarchical monitoring architecture is to generate optimal agent hierarchy such that communication cost and memory usage is reduced, and task is distributed among the agents. The optimal agent hierarchy generation problem can be formalized as follows-

$$\begin{aligned} minimize \sum agent\_count(S_i), \\ minimize \sum Data\_Source\_Count(S_i) \end{aligned} \quad (3.10)$$

subject to

$$monitoring\_cost < threshold$$

To develop the hierarchical monitoring architecture, we have to address the following problems: 1) Given the subscription request and network topology, generate the

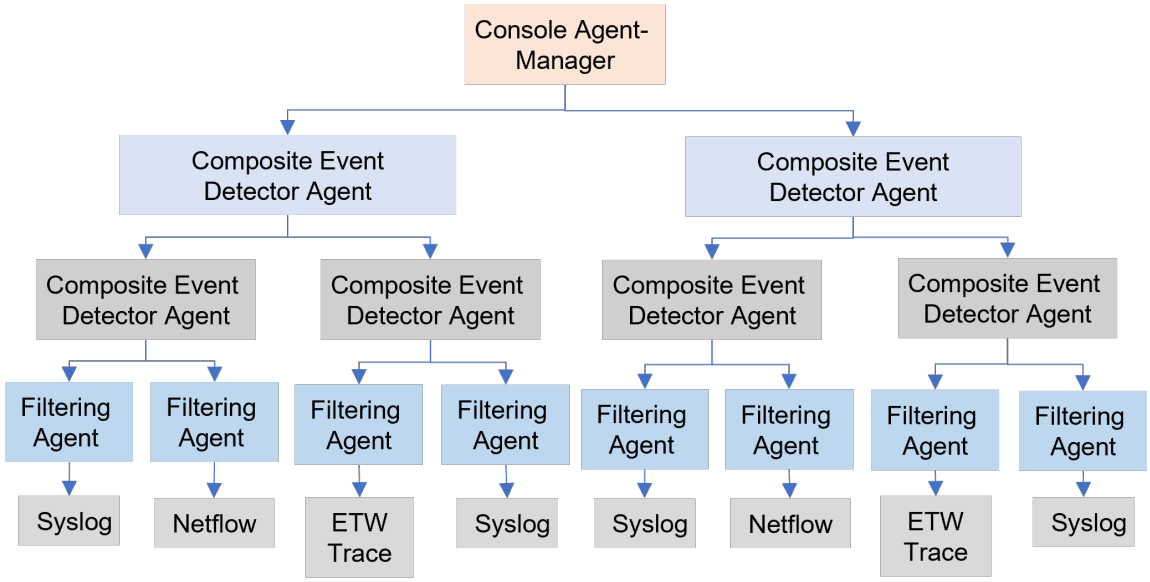

Figure 3.1: Distributed Hierarchical Monitoring Agent Architecture

optimal number of middle-level agents, maintaining monitoring capacity constraints of each agent, 2) Given the subscription request, determine the optimal number of data sources to monitor, 3) develop monitoring agent for the specific data source. The following section provides the agent architecture, optimal agent hierarchy, and data source monitor generation.

3.4 Distributed Hierarchical Monitoring Agent Architecture Overview

In this section, we describe each component of SCAHunter as shown in Figure 3.1. Our SCAHunter consists of three types of agents- console agent or manager (CA), composite event detector agent (CEDA), event filtering agent (EFA), and data sources to monitor. The user of this agent architecture (or a cyber threat hunter) provides a single event or group of events or correlated event subscription requests (i.e., attack signatures) using the CA. The CA decomposes the subscription request based on the formalization provided in Section 3.3. The CA also generates the required number of CEDAs using the decomposed subscription request. It determines appropriate EFAs, and agent hierarchy such that communication overhead is minimum and subscription

task monitoring is distributed across the hosts in the network. Since the optimal agent hierarchy generation is an NP-hard problem, our CA uses an approximation algorithm to generate a near-optimal agent hierarchy. Then, the CA sends the decomposed event subscription requests to the corresponding CEDAs and EFAs through a dedicated configuration channel. Upon receiving a subscription request from the CA, an EFA will start monitoring corresponding data sources. Whenever the EFA detects a subscribed event, it publishes detected events to the upper-level (parent) CEDA through task-specific channels as alerts containing task id and event details. The CEDA or EFA also replies to the CA through a dedicated configuration channel to activate the next monitoring task and deactivate the detected monitoring task to detect a subscription request while supporting on-demand monitoring.

3.4.1 Console Agent (CA) or Manager

The Console Agent takes an attack signature as input (ESR) from the cyber threat hunter at the beginning of the threat hunting process. This agent's first task is to decompose the subscription request received from the threat hunter using the formalization described in Section 3.3. The CA's second task is to determine how many CEDA levels to generate and how many CEDAs to develop and where to place those generated CEDAs based on the decomposed event subscription request, the data source to monitor, and the location of end-host devices. It also needs to determine EFA(s) where the event subscription request will be sent. After determining the CEDAs and EFAs, the CA generates the corresponding CEDA and configures them. After a decision as a reply from CEDA is available, it informs the threat hunter about the detected TTPs or attack techniques.

Since the number of generated CEDAs will impact resource usage and communication overhead within the CA and the network, appropriate CEDA number, level, and place are required. We can formulate the determination of the proper number of CEDAs required as *Generate a minimum number of CEDAs that can cover all*

required EFAs for serving the event subscription request. We can formulate this problem as a set-cover problem that is NP-complete. One way to solve this problem is to use heuristics on each CEDA or EFA's predefined monitoring capacity and the geographical distribution of end-host devices. Specifically, we propose an approximation algorithm (AHG in Algorithm 1) based on these heuristics.

3.4.2 Composite Event Detector Agent (CEDA)

Composite event detector agent reduces network communication overhead (traffic flow) between the CA and EFAs by replying to the upper-level CEDA or the CA if a monitoring task or subscribed event is detected. The hierarchical agent architecture can have multiple levels of CEDAs. If the CA creates one level of CEDAs, each CEDA's child is an EFA, and its parent is the CA. On the other hand, if the CA creates two levels of CEDAs, the child of lower-level CEDA is an EFA, and the parent of it is a higher-level CEDA, and the parent of the higher-level CEDA will be the CA.

3.4.3 Event Filtering Agent (EFA)

The event filtering agent is the lowest level of the agents in the SCAHunter. It monitors different data sources for events requested in the received event subscription request. These agents are static: we generate them initially and continue to work until closed or subscription requests are deleted. MITRE ATT&CK framework [26] provides around 38 different data sources to monitor for detecting different attack TTPs. Thus, to detect all attack techniques and sub-techniques provided by the MITRE ATT&CK framework, we have to develop less than 38 different event filtering agents. We developed EFAs to monitor the following data sources: ETW trace, Netmon, and Sysmon.

3.4.4 Agent Communication Protocol

Since every agent in SCAHunter may consume event logs or alerts from multiple other agents or data sources, SCAHunter uses a publish-subscribe communication pattern for group communication among the agents. The CA, CEDA, and EFA agents may work as producers and consumers. The CA configures CEDA and EFA agents by publishing configuration info to the corresponding agents through the configuration channel. It also consumes alerts from lower-level CEDA or EFA agents through task-specific channels. CEDA and EFA publish detected monitoring tasks as alerts to the upper-level CEDA or CA through task-specific channels. We use a publish-subscribe communication pattern to facilitate the above-mentioned group communication among the agents. Though the proposed agent architecture employs a hierarchical structure for event monitoring and detection, it is a virtual hierarchy constructed by CEDAs and EFAs' group communication over publish-subscribe communication protocols. Using publish-subscribe communication protocols in agent communication will improve agent hierarchys' robustness in agent failures or network partitioning.

Algorithm 1 Agent Hierarchy Generation Algorithm **AHG**(S, MT, A)

Input: attack signature S, monitoring task list MT, agent hierarchy A. Output: agent hierarchy A

if $size(MT) == 1$ **then**
    return $A \cup \{MT\}$
end if
for all pair(m_i, m_j) where $m_i \in MT, m_j \in MT, m_i != m_j$ **do**
    $score, correlationScore \leftarrow$ correlation between m_i and m_j, (score, m_i, m_j)
end for
sort correlationScore based on $score$
$covered \leftarrow \phi$, $capacity \leftarrow \phi$, $ind \leftarrow 0$, $cluster \leftarrow \phi$, $index \leftarrow 0$
for all $(score, m_i, m_j) \in correlationScore$ **do**
    if $m_i \in covered$ and $m_j \in covered$ **then**
        continue
    else if $m_i \in covered$ **then**
        $index, m_i \leftarrow$ find cluster index containing m_i, None
    else if $m_j \in covered$ **then**
        $index, m_j \leftarrow$ find cluster index containing m_j, None
    end if
    if $m_i != None$ and $m_j != None$ **then**
        $cluster_{ind}, capacity_{ind}, covered \leftarrow m_i \cup m_j, 2, covered \cup m_i \cup m_j$
    else if $m_i != None$ and $capacity_{ind} + 1 <= threshold$ **then**
        $cluster_{ind}, capacity_{ind}, covered \leftarrow cluster_{ind} \cup m_i, capacity_{ind} + 1, covered \cup m_i$
    else if $m_j != None$ and $capacity_{ind} + 1 <= threshold$ **then**
        $cluster_{ind}, capacity_{ind}, covered \leftarrow cluster_{ind} \cup m_j, capacity_{ind} + 1, covered \cup m_j$
    else if $m_i != None$ **then**
        $cluster_{ind}, capacity_{ind}, covered \leftarrow m_i, 1, covered \cup m_i$
    else
        $cluster_{ind}, capacity_{ind}, covered \leftarrow m_j, 1, covered \cup m_j$
    end if
    $ind \leftarrow ind + 1$
end for
$A \leftarrow A \cup \{cluster\}$
return AHG(S,cluster, A)

3.4.5 ESR Decomposition and Agent Hierarchy Generation.

A cyber threat hunter provides attack signatures as ESR to the CA for the subscription of related events. However, the CA needs to decompose the provided ESR to generate the required CEDAs and determine corresponding EFAs. The first step in the signature decomposition process is to determine primitive events. Since any predicate containing a relationship between event attribute and specific value is a primitive event predicate (PE_i) to monitor, determining primitive events from signature is trivial. Similarly, for any predicate containing a relationship among multiple identical or different event attributes, we can consider such predicate as a composite event predicate (CE_i) or correlation task. Primitive event decomposition Algorithm 2 ($PED(S)$) takes an attack signature as a monitoring task and converts it to a predicate list (line 1). Next, the PED algorithm extracts primitive events and correlation tasks based on the predicate's event attributes, value, and relationship (lines 3-9).

Given the decomposed primitive event predicates, PE_i, and correlation tasks (composite event predicate CE_i) to monitor, the agent hierarchy generation Algorithm 1, AHG (MT, A), determines the intermediate agents (CEDAs) to monitor correlation among different primitive events. If $MT_{i,j}$ denotes the monitoring task of agent i at level j, we can formalize the intermediate agent generation with the following recurrence relation:

$$MT_{i,j} = \begin{cases} \bigcup_{k \in corrSet} PE_k, & \text{if } j == 0 \\ \bigcup_{k \in corrSet} MT_{k,j-1}, & \text{otherwise} \end{cases} \tag{3.11}$$

Where $corrSet$ is a set of monitoring tasks of lower-level monitoring agents which are highly correlated either by event attribute name or value. For the EFA agent, each primitive event predicate, PE_i, is a monitoring task of the corresponding agent. Therefore, we can determine the monitoring task for agent i at level j by clustering monitoring tasks of a group of agents at level $j-1$ based on the correlation among

the monitoring tasks of the agent at level $j-1$. The AHG algorithm calculates the correlation score for every pair of monitoring tasks in MT (lines 4-6) and sorts the estimated correlation score (line 7). Next, the AHG algorithm greedily selects the pair with the highest correlation score.

It adds each monitoring task in the pair to a cluster if it is not present in the existing cluster yet (e.g., for the pair (m_i, m_j), if m_j is present in an existing cluster but m_i is not, the algorithm tries to add m_i to the cluster of m_j) (lines 17-22). The AHG algorithm creates a new cluster if the existing cluster's size has reached a threshold (lines 23-27). This way, the algorithm greedily clusters all monitoring tasks with a high correlation. In the end, each cluster represents the CEDA of the current agent level. The generated clusters for the current level will be the monitoring tasks for the next level (line 31). The above process continues until only one monitoring task remains (line 1). Since every CEDA is performing a part of the monitoring task, the monitoring task is distributed across the hosts in the network. Moreover, since monitoring tasks in a CEDA have a high correlation, all the predicates related to those monitoring tasks can be calculated with optimal communication; thus, the SCAHunter reduces overall communication overhead. This agent hierarchy generation algorithm starts by taking all primitive events as monitoring tasks.

Algorithm 2 Primitive Event Decomposition Algorithm **PED**(S)

Input: attack signature, S

Output: primitive event to monitor, MT and correlation task, CT

1: *predicates* ← convert signature S to predicate list
2: $MT, CT \leftarrow \phi, \phi$
3: **for all** $p_i \in$ predicates **do**
4: **if** p_i contains relation among event attribute and value **then**
5: $MT \leftarrow MT \cup \{p_i\}$
6: **else**
7: $CT \leftarrow CT \cup \{p_i\}$
8: **end if**
9: **end for**
10: return $MT \cup CT$

The agent hierarchy generation starts by calling $AHG(S, PED(S), \phi)$, or Algorithm 1, with the attack signature S to monitor, the decomposed monitoring tasks calculated by PED(S), or Algorithm 2, in linear time, and an empty cluster list. Algorithm 2 decomposes the given attack signature S to monitor in linear time. Since the AHG algorithm runs recursively to cluster primitive tasks, each recursive call will reduce the problem size to $N, N/c, N/c^2, N/c^3, ..., 1$, where c is the monitoring capacity of each CEDA agent and N is the number of decomposed monitoring tasks (Algorithm 2). Thus, there will be $log_c N$ number of recursive calls to the AHG algorithm. In each of those recursive calls, there will be $N^2 + N^2 log_2 N^2 + N^2 = 2N^2 + 2N^2 log_2 N \approx O(N^2 log_2 N)$ work in the worst case. Thus the run time of the AHG algorithm is $log_c N \times O(N^2 log N) \approx log_2 c \times log_c N \times O(N^2 log N) \approx log_2 N \times O(N^2 log_2 N) = O(N log_2 N)^2$. Moreover, the AHG algorithm uses $O(N^2)$ additional memory to generate the agent hierarchy.

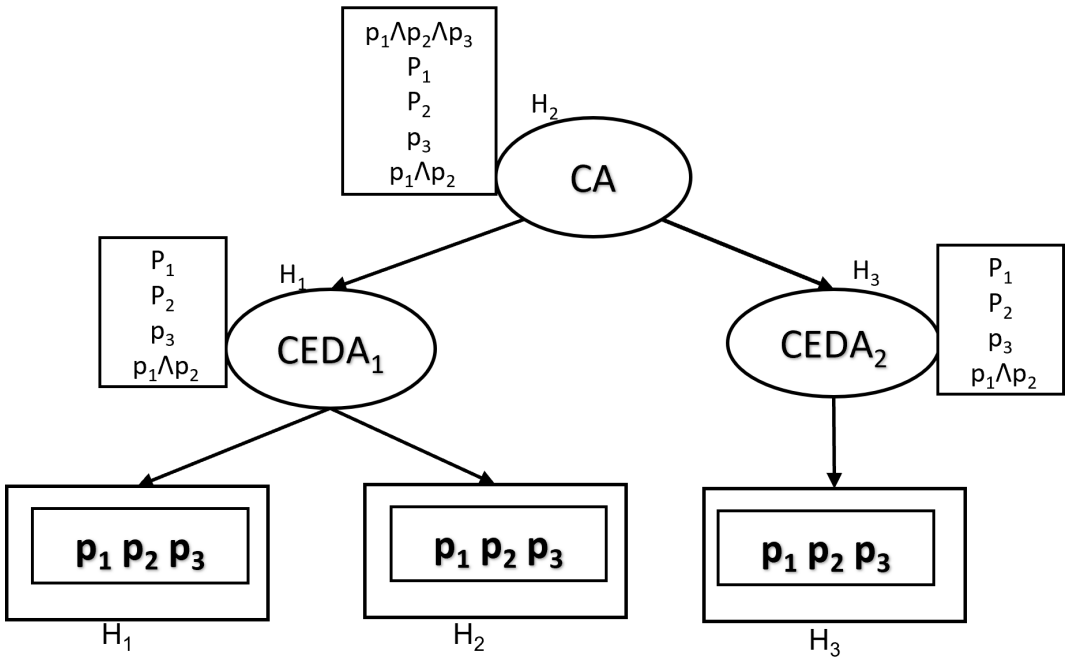

Figure 3.2: Generated agent hierarchy

3.4.6 Distributed Hierarchical Monitoring Use Case Demonstration

This section will demonstrate threat hunting using SCAHunter. We use a hypothetical attack signature ($p_1 \wedge p_2 \wedge p_3$) that contains three primitive event predicates (p_1, p_2, and p_3) to monitor. We demonstrate signature decomposition, hierarchy generation, and threat hunting in a network consisting of three hosts (H_1, H_2, and H_3). We also assume that each generated CEDA agent has a monitoring capacity of two and each PE_i (primitive event predicate) corresponds to separate data sources. Thus, we can say that each PE_i is a monitoring task of EFA. At a high level of abstraction, our signature decomposition Algorithm 2 groups event predicates based on event type and host to generate PE_i, which is the monitoring task of EFA. Next, the AHG algorithm takes all generated PEs as monitoring tasks ($MT_{i,j} \in MT_j$) and generates monitoring tasks ($MT_{k,j-1} \in MT_{j-1}$) for higher-level agents by clustering $MT_{i,j}$ based on the similarity among $MT_{i,j}$. For our signature, $MT_0 = \{p_1, p_2, p_3\}$.

The AHG Algorithm 1 calculates correlation score for each pair of $MT_{i,j}$. For our

case, let's assume p_1 has a higher correlation with p_2 than p_3. Thus, AHG algorithm clusters p_1 and p_2 together first. Since each agent has a monitoring capacity of 2, p_3 requires a separate cluster. Thus, $MT_1 = \{p_1 \wedge p_2, p_3\}$. At the next step, AHG algorithm 1 recursively generates monitoring tasks for the next level. In our case, MT_1 is the monitoring task and AHG algorithm clusters each $MT_{1,j}$. Though $MT_{1,1}$ and $MT_{1,2}$ does not have any similarity between them, we will cluster them together because monitoring capacity allows for up to 2 monitoring tasks. Thus, $MT_3 = \{p_1 \wedge p_2 \wedge p_3\}$ which is the attack signature to monitor. At this step, Algorithm 2 stops and returns the monitoring tasks for each level. The monitoring tasks for each CEDA will be $MT = \{p_1, p_2, p_3, p_1 \wedge p_2\}$. The overall result of our algorithms is illustrated in Figure 3.2.

The hierarchy generation algorithm will determine how many CEDAs to generate and where to place them and what are the communication channels. Since only one monitoring task corresponding to an attack signature will be active at a time, in the beginning only p_1 will be active in all CEDAs and p_1 needs to be monitored in all hosts. Since our use case has three hosts and each agent has a monitoring capacity of 2, we can generate $CEDA_1$ to monitor MR in H_1 and H_2 and $CEDA_2$ to monitor MR in H_3. We launch $CEDA_1$ in H_1 or H_2 and $CEDA_2$ in H_3. $CEDA_1$ communicates with $\{H_1, H_2\}$ using task specific channels $(c1_{p1}, c1_{p2}, c1_{p3})$, and $CEDA_2$ communicates with H_3 using task specific channels $(c2_{p1}, c2_{p2}, c2_{p3})$ and the CA communicates with H_1, H_2, H_3 through configuration channel and with $CEDA_1, CEDA_2$ through task specific channels $(ca_{p1}, ca_{p2}, ca_{p3})$. Now, the CA will activate first monitoring task in $CEDA_1$ and $CEDA_2$ and start the appropriate EFA agent in all hosts to monitor specific ETW provider corresponding to p_1. Suppose $CEDA_1$ detects p_1, it will notify the CA by sending a detection alert containing detected MR_i through a configuration channel. If monitoring of p_1 is not required for any other signatures, the CA will stop monitoring p_1 in $CEDA_1$ and $CEDA_2$. It will also stop consuming

event logs from data source corresponding to p_1 in all hosts by removing corresponding ETW providers from active session of EFAs. The CA also activates the next monitoring task, p_2, in all CEDAs through configuration channels. It also activates EFA corresponding to p_2 if it is not already active. This process continues until all p_1, p_2 and p_3 is detected. This threat hunting approach through SCAHunter reduces event storage requirements through on-demand monitoring, and the hierarchical structure of the agent communication ensures optimal communication overhead.

3.5 Implementation and Evaluation

We implemented the SCAHunter using Python in a Windows 10 machine with 64GB RAM and multiple VM settings (two hosts, three hosts, four hosts, and five hosts). We simulated attacker and benign user activities on all the VM settings. This section provides implementation details of the CA, CEDAs, and EFA agents and evaluates the SCAHunter using three simulated use cases and DARPA OpTC attack dataset [30].

3.5.1 Implementation Details

We implemented the console agent (CA) using Python 3. The CA consists of a signature decomposition and agent hierarchy generation module, configuration module, signature filtering module, and communication module. Given an attack signature in ESR format and network topology as an adjacency list, the signature decomposition module decomposes ESR into primitive predicates and composite predicates. We described the decomposition process in Section 3.4.5. It also generates agent hierarchy based on an approximation algorithm 1. After agent hierarchy generation, the configuration module configures all required agents and monitoring tasks in the corresponding hosts. It also configures the publish-subscribe communication protocol so that agents can communicate with a group of required agents. A CA agent works as both a subscriber (consumer) and publisher (producer). And CA con-

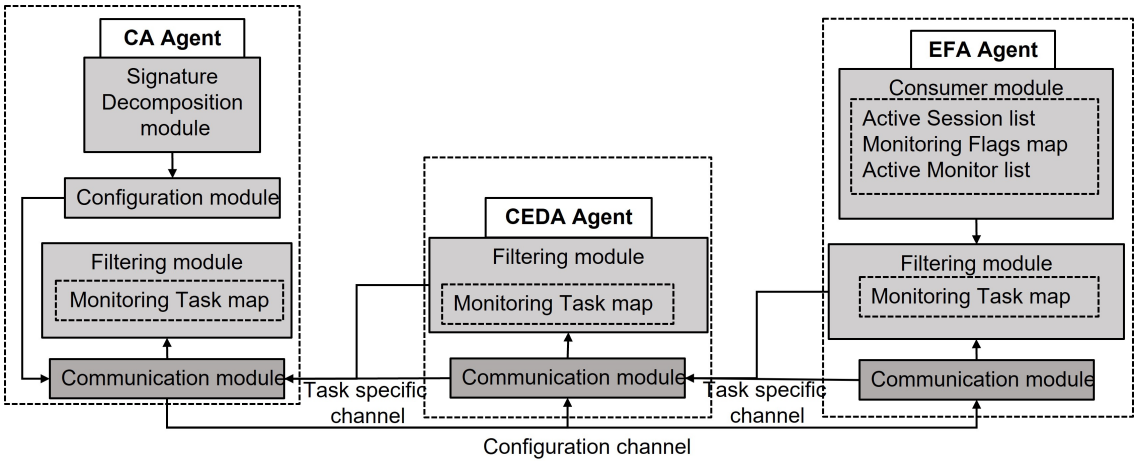

Figure 3.3: Implementation of distributed hierarchical monitoring agent architecture

sumes (subscribe) alerts corresponding to monitoring tasks from lower-level CEDAs or EFAs. The signature filtering module stores monitored tasks in a map data structure to maintain constant time filtering. We implemented the composite event detector agent (CEDA) using python 3. The CEDA consists of a filtering module and communication module. We implemented the filtering module using the same logic as the filtering module of CA while maintaining constant time filtering. CEDAs subscribe (consume) monitored events from lower-level CEDAs or EFAs. It also subscribes for configuration info from the CA.

We implemented the primitive event filtering agent (EFA) using Python 3 and CPython. The EFA consists of an event filtering module, event consumer module, and communication module (Figure 3.3). The event filtering module stores the decomposed monitoring task in a map (python equivalent dictionary) data structure to do the event filtering tasks in constant time. The event consumer module consumes data from different data sources. To ingest data from ETW trace, we use various etw providers. Windows ETW trace provides around 1000 ETW providers to produce event logs corresponding to different data sources. It also provides wintrace API to collect and manage event logs from ETW. ETW trace program consists of a con-

troller, provider, and consumer. Our EFA agent creates an ETW trace session with a specific ETW provider based on the data source required to monitor. The EFA consumer module consumes event data from the created session through the win32 event tracing API. Interaction with ETW event tracing API is implemented using CPython.

Moreover, we provided ways to monitor specific events from an ETW provider through win32 event tracing API. For example, the Sysmon provider generates event logs corresponding to different data sources such as process creation, network traffic, file access, and registry access. Our EFA agent can collect process creation, network connection, or file access events through event tracing API. Our EFA agent also maintains a list of active ETW sessions. If a new monitoring request comes from the CA, the EFA agent creates a new ETW session if required event logs can not be collected from the currently active session. We implemented Sysmon and Netmon EFA agents and ingested data from ETW Windows Sysmon provider, Winsock providers, and Powershell provider. We also implemented the publish/subscribe communication protocols using a message broker library RabbitMQ [59]. The CA subscribes for monitored signature and publishes configuration info and monitoring tasks to the CEDAs or EFAs using RabbitMQ. The CEDA uses RabbitMQ to ingest alerts for the monitored jobs from the lower-level CEDAs or EFAs and publishes the detected monitoring tasks as alerts to the upper-level CEDA or CA. The EFA agent publishes collected event logs from ETW trace sessions to the upper-level CEDA or CA using RabbitMQ.

To compare our proposed event filtering approach with the existing centralized approaches, we also implemented centralized monitoring using Splunk.

3.5.2 Evaluation

We evaluate our approach with three simulated attack scenarios (multiple attacker activity sequences expressed in MITRE ATT&CK techniques) and one public

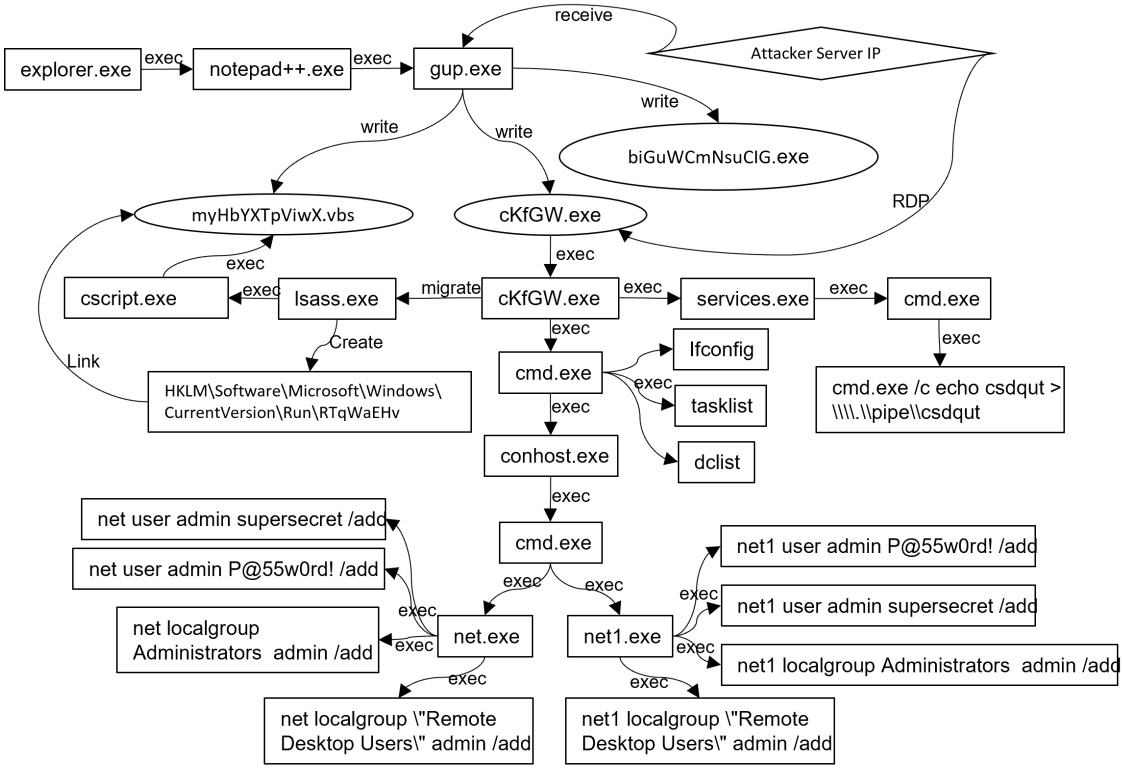

Figure 3.4: Low-level Attacker Activities in OpTC Dataset

attack dataset, the OpTC (Operationally Transparent Cyber) dataset released by DARPA [30]. We use Atomic Red Team's atomic tests [29] to generate attacker activities in a benign user session. The atomic test case includes scripts to execute multiple procedures of each MITRE ATT&CK technique. We generate scripts for a sequence of attacker activities using the MITRE ATT&CK technique by cascading scripts of multiple techniques. In usecase 1, we execute a payload/malware file using Technique PowerShell (T1059.001) [26]. The executed payload performs the account discovery process using technique T1087 [26], and in the end, the malware process executes a scheduled task using technique T1053 [26]. Usecase 2 executes techniques T1189, T1035, T1021.001, T1119, and T1048 [26] using scripts from the Atomic Red Team; we provide the signature for this use case in Appendix A. Usecase 3 uses all the techniques in Usecase 2 plus one defense evasion technique, masquerading (T1036) [26].

The OpTC dataset [30] contains 287 GB event logs collected on five hundred to one thousand hosts over three days. During these three days, the red team performed three APT scenarios. Our evaluation uses one scenario that consists of around 70 GB of log data corresponding to 1.3 billion events. In this APT scenario, the red team updated *notepad++*, which downloads and executes a Meterpreter payload. This payload execution obtains system access through named pipe impersonation. It performs several discovery techniques: system info, installed applications, domain controller, and network shares. It also performs *ARP scanning* to discover neighbor hosts in the same network. Then, the Meterpreter process creates a *Run* registry key and sets the value of the registry key to a downloaded Meterpreter module to establish persistence. Later, this process adds *administrator* and *admin* accounts to the *administrators* and *RDP group*. Finally, the attacker RDPed to the compromised machine from the attacker's server. We develop attack signatures corresponding to each red team activity (Appendix A) and use those signatures to evaluate SCAHunter. The discovered low-level attacker activities are shown in Figure 3.4. In this figure, nodes represent system entities such as processes, registry keys, commands (rectangles), files (ovals), and sockets (diamonds), and edges corresponding to the attacker's actions represent the information flow and causality. To evaluate the proposed threat hunting approach, we try to answer the following three questions:

1. Does threat hunting with distributed hierarchical monitoring reduce communication overhead and storage requirement compared to the state-of-the-art centralized threat hunting approaches? Since a centralized approach tries to monitor everything and aggregate logs in a central server, it has to store all of the logs generated by the EFA agents. In contrast, a distributed monitoring approach records logs only for a single monitor corresponding to the current active monitoring task. There can be one active monitoring tasks for a single attack signature. Moreover, CEDAs corresponding to the monitoring tasks will store results of only detected

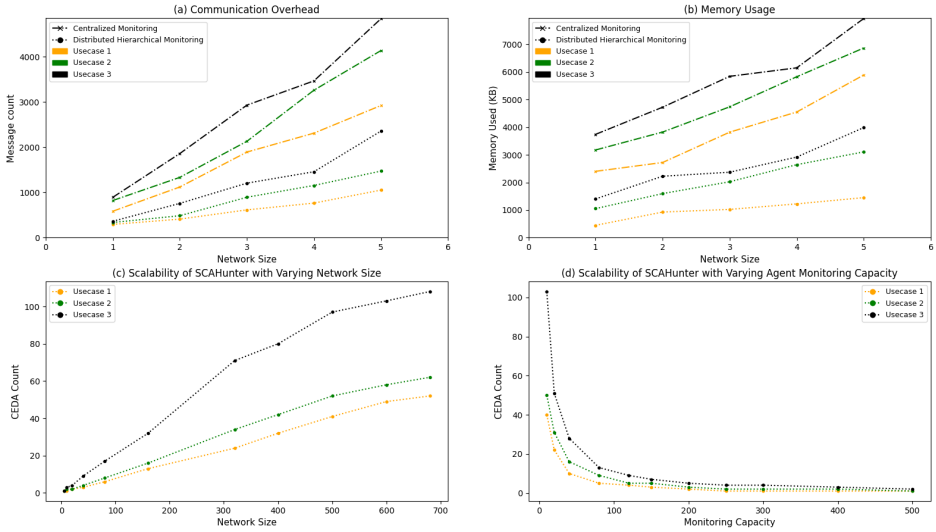

Figure 3.5: Performance and Scalability Evaluation of hierarchical monitoring agent architecture based on Simulated Attack Use Cases

monitoring tasks for the currently active monitor. As a result, the distributed hierarchical monitoring records fewer events than the centralized approach, which is evident in Figure 3.5 (a) for simulated attack use cases and Figure 3.6 (a) for OpTC attack dataset.

Specifically, to confirm the decrease in communication overhead, we measure the number of exchanged messages (alerts, configuration messages) among the agents in three simulated attack use cases for both centralized and proposed distributed approaches. During this evaluation, we maintain a fixed monitoring capacity of 12 for each agent while varying the network size from one to five. For the centralized monitoring approach, we collected event logs from only data sources related to the attack signature. As we can see in Figure 3.5 (a), using the SCAHunter approach, the number of exchanged messages among agents (event count) decreases around 49.74% to 66.9%, 58.04% to 64.58%, and 51.3% to 59.35% for use case 1, use case 2, and use case 3, respectively, compared with the centralized approach. We can also see that the decrease becomes more and more significant as the network size increases. Similarly,

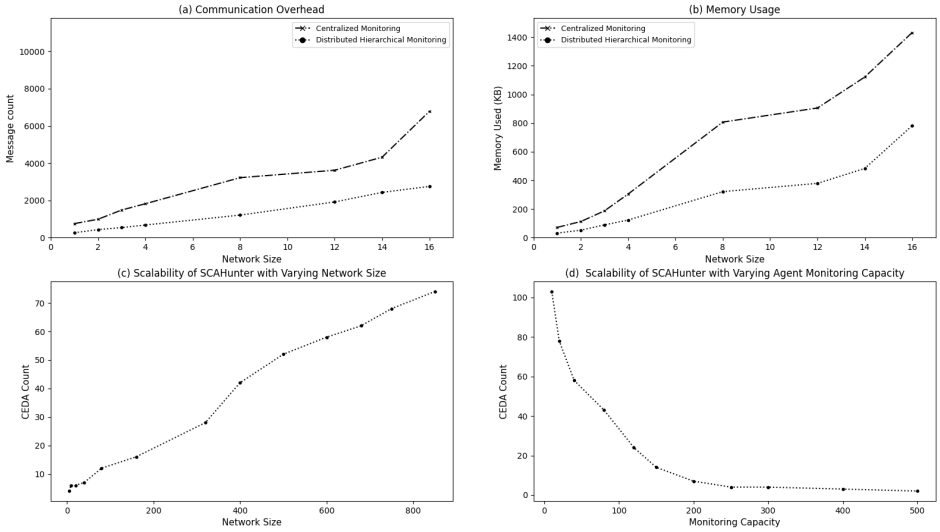

Figure 3.6: Performance and Scalability Evaluation of hierarchical monitoring agent architecture based on OpTC Attack Dataset

we measure the number of exchanged messages among agents using the OpTC attack dataset for centralized and proposed distributed approaches. For the centralized monitoring approach, we collected logs from the following data sources: file creation activity, registry key creation activity, process creation activity, and RDP network connection creation activity. During this evaluation, we maintain a fixed monitoring capacity of 12 for each agent while varying the network size from one to 16. In our proposed approach, the number of exchanged messages among agents decreases by around 43.7% - 64.8% from the centralized approach, as shown in Figure 3.6 (a). Since our proposed approach forwards event logs corresponding to the current active monitoring task, performs distributed event filtering by event correlation in CEDA agents, and forwards only the detected correlated events to the upper-level agents or CA, the communication overhead is lower compared to the centralized approach as shown in Figure 3.5 (a) and Figure 3.6 (a). To show the decrease in memory usage overhead, we measure the memory used by the agents (CA, CEDA, EFA) to store the events corresponding to the detected monitoring tasks in the centralized

approach and our proposed hierarchical approach. The amount of memory used is the number of event logs times the average size of each event log, which we estimate as 500 bytes based on event logs from the ETW Sysmon provider. In the proposed hierarchical monitoring approach, the total memory usage decreases around 65.9% to 81.3%, 54.6% to 66.85%, and 49.75% to 62.48% for simulated attack use case 1, use case 2, and use case 3, respectively, from the centralized monitoring approach as shown in Figure 3.5 (b). Similarly, the proposed SCAHunter approach decreases total memory usage by 45.4% to 60.1% for the OpTC attack dataset compared with the centralized monitoring approach, as shown in Figure 3.6 (b). Since our proposed approach stores only the event logs related to the monitoring tasks and only a subset of the monitoring tasks are active at any moment of threat hunting, the storage required is less for the hierarchical monitoring approach than the centralized approach, as shown in Figure 3.5 (b) and Figure 3.6 (b).

2. Can threat hunting with distributed hierarchical monitoring detect multi-step attacker activities with the accuracy of state-of-the-art centralized threat hunting approaches? Since the centralized approach monitors all required data sources for an attack signature and aggregates generated logs in a central manager, the accuracy of the centralized approach depends on the quality of the attack signature. On the other hand, since the distributed approach decomposes attack signatures such that monitoring tasks are distributed among hosts, an efficient event correlation is required in addition to the signature quality. For the DARPA OpTC attack dataset, both the centralized and SCAHunter threat hunting approaches can detect low-level attacker activities, shown in Figure 3.4. For example, the signature $a.Operation == NewFileWrite \wedge x.event\_id == process\_creation \wedge x.imagePath == a.newFilePath$ is used to detect Meterpreter's payload downloading activity.

For the simulated attack use cases, the centralized approach can detect all three

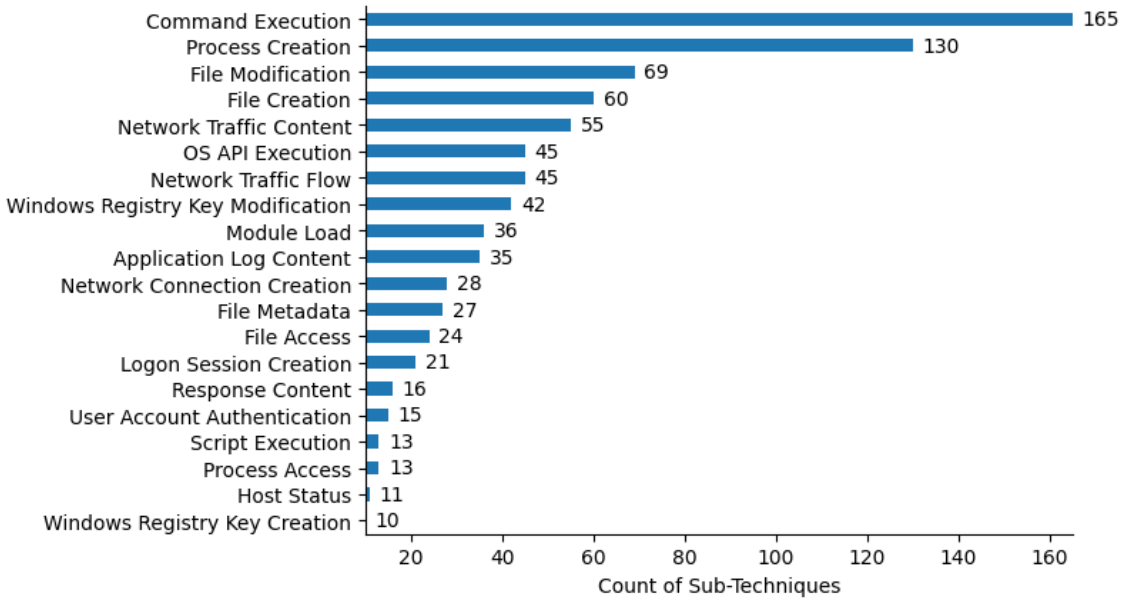

Figure 3.7: Data source to technique coverage

attack signatures. However, the distributed hierarchical monitoring approach initially missed one of the attack signatures. Further investigation of low-level logs reveals that this is due to the on-demand nature of our proposed approach, which does not monitor everything, instead, it activates a monitoring task only if all the previous monitoring tasks in the attack signature are detected. Therefore, there is a time lag in our approach, which we can mitigate by using a memory buffer. In our new implementation, we keep event logs of all required data sources of the current signature for thirty seconds. This user-defined period of log recording is introduced to cover the time lag between detecting one monitoring task and activating the next monitoring task. After introducing this log buffer, our approach detects previously missed attack signatures.

3. Is the proposed hierarchical monitoring architecture scalable? We examine the scalability of threat hunting with the hierarchical monitoring approach as the size of the network topology and the monitoring capacity increase. We measure the number of required CEDAs for the three simulated attack use cases and the OpTC

dataset in a simulated network whose size varies from 1 to 800 hosts. In Figure 3.5 (c), we see that with a 20% to 25% increase in network size (e.g., from 320 to 400), the number of required CEDAs increases 19.5% to 33.3%, 12.4% to 28%, and 23% to 32.73% respectively, for use case 1, use case 2, and use case 3. Similarly, In Figure 3.6 (c), we see that with a 20% to 25% increase in network size, the required number of CEDAs increases by 8.3% to 23.8% for OpTC attack dataset. From Figure 3.5 (c) and Figure 3.6 (c), we see that the number of required CEDAs grows linearly with the network size and tends to flatten out. In all these experiments, we set the monitoring capacity to 12.

We also measure the number of required CEDAs for the three simulated attack use cases and the DARPA OpTC attack dataset with a varying monitoring capacity (from 1 to 600) in a fixed simulated network of size 50. With a 25% increase in the agent monitoring capacity, the number of required CEDAs decreases 11.3% to 19.6%, 12.4% to 28%, and 17.1% to 32.9% respectively, for use case 1, use case 2, and use case 3, as shown in Figure 3.5 (d). Similarly, In Figure 3.6 (d), we see that with a 25% increase in the agent monitoring capacity, the required CEDA decreases by 4.7% to 38.4% for OpTC attack dataset. The number of required CEDAs is inversely co-related with monitoring capacity, as shown in Figure 3.5 (d) and Figure 3.6 (d).

Since the number of required CEDAs is linearly increasing with the increasing size of the network topology, as shown in Figure 3.5 (c) and Figure 3.6 (c), we need to generate around 10% to 33% more CEDAs to support the threat hunting using hierarchical monitoring which is a feasible number of CEDAs. However, the number of required CEDAs is inversely co-related with monitoring capacity, as shown in Figure 3.5 (d) and Figure 3.6 (d). Since the number of required agents to monitor required tasks will decrease with the increasing monitoring capacity, the CEDA generation with the provided approximation algorithm is scalable.

Moreover, Figure 3.7 shows the technique coverage of the top 20 data sources

among 81 data sources. We can see that Command Execution and Process Creation cover 36% to 42% techniques, while File Modification and File Creation cover around 32% techniques. Since most of the data sources cover multiple techniques, developing EFAs for around 25% of the data sources will cover all remaining techniques, which is feasible.

3.6 Static ESP rule generation for Attack Signature

Although we developed multiple primitive event detector agents and EFAs based on ETW trace, we need to develop additional EFA agents to cover all the data sources. Since the single-point-of-failure calls for applying the consensus mechanism, such as the leader election algorithm, which is another research problem to solve, we did not consider the single-point-of-failure for SCAHunter. Our agent architecture will not be optimal if a new attack signature added to the system overlaps with the existing monitored signatures. We plan to explore this topic in the future.

In order to build the attack signature in terms of low-level system events, we follow the technique detectors provided by LogRythm [42]. Following the rules developed by LogRythm, technique T1189 can be detected by inspecting IDS or antivirus logs. Technique T1035, T1021.001, and T1119 [26] can be detected by analyzing Windows event logs or SysMon logs as mentioned by MITRE [26]. Moreover, technique T1048 can be detected by analyzing network traffic and looking for uncommon data flow in the NetMonitor. Therefore, we can define the signature in terms of low-level logs for technique T1189, T1035, T1021.001, T1119, and T1048 [26], and use case 2 using

ESP formalization provided in Section 3.3 as follows:

$$IDSMon.\_event\_id == Malware\_detected \wedge$$
$$Malware\_detected.malwareFileDir ==' *temp' \wedge$$
$$SysMon.\_event\_id == object\_access \wedge$$
$$object\_access.object\_dir ==' *temp' \wedge$$
$$object\_access.procName =' chrome' \quad (3.12)$$

$$SysMon.\_event\_id == service\_creation \wedge$$
$$service\_creation.command ==' SC\ create' \wedge$$
$$service\_creation.imagepath ==' *temp' \quad (3.13)$$

$$SysMon.\_event\_id == RDS\_logon\_success \quad (3.14)$$

$$SysMon.\_event\_id == process\_created \wedge$$
$$process\_created.command ==' *.bat' \wedge$$
$$SysMon.\_event\_id == network\_conn\_created \quad (3.15)$$

$$NetMon.\_event\_id == uncommon\_data\_flow \quad (3.16)$$

$$(Equation\ 3.12) \wedge (Equation\ 3.13) \wedge (Equation\ 3.14) \wedge (Equation\ 3.15) \wedge$$
$$(Equation\ 3.16) \wedge (object\_access.object\_dir == service\_creation.imagePath$$
$$== malware\_detected.malwareFileDir) \wedge (RDS\_logon\_success.src\_ip$$
$$== ids\_event\_id.host\_ip) \wedge (RDS\_logon\_success.session\_id ==$$
$$process\_created.session\_id == network\_conn\_created.session\_id) \wedge$$
$$(uncommon\_data\_flow.network\_protocol ==$$
$$network\_conn\_created.network\_protocol) \quad (3.17)$$

To build the attack signature of the red team activities in OpTC attack dataset, we investigate notepad++ update process and Meterpreter execution process. To detect Meterpreter payload download activities, we can look for new file create and write event logs and process creation using the newly created file.

$$a.Operation == NewFileWrite \wedge x.event\_id == process\_creation$$
$$\wedge\, x.imagePath == a.newFilePath \quad (3.18)$$

Named pipe impersonation can be detected by looking for any cmd.exe process which is a child of services.exe and the commandline arguments of the cmd.exe process contains echo and pipe keywords.

$$b.processName == cmd.exe \wedge b.parentProcess == services.exe$$
$$\wedge\, b.Commandline \in echo \wedge b.Commandline \in pipe$$
$$\wedge\, b.process\_id == x.process\_id \quad (3.19)$$

The system info, installed applications, domain controllers and network share discovery activities can be detected by looking for command line arguments running form the cmd.exe process which is spawn from Meterpreter process.

$$(c.commandline \in [local\_system\_info\_enumeration\_command \cup installed\_application\_enumeration\_command(tasklist) \cup domain\_controller\_enumeration\_command(dclist) \cup network\_share\_enumeration\_command(Get\_SmbShare)] \wedge c.process\_id == b.process\_id) \quad (3.20)$$

We can detect the registry key creation and setting the value of the created key to a newly downloaded payload as follows:

$$g.Operation == NewFileWrite \wedge h.Operation == RegistryKey\_create \wedge g.newFilePath == h.registryKeyCreated.value \wedge g.process\_id == h.process\_id == x.process\_id \quad (3.21)$$

Creating new user account and adding it to specific group can be done by using net utility. Thus the signature will be:

$$i.Commandline \in net\_userAccount\_add\_command \wedge i.process\_id == b.process\_id \wedge i.processName == Net.exe \quad (3.22)$$

3.7 Conclusion, Limitations and Future Work

This chapter describes a distributed hierarchical monitoring architecture for threat hunting using the MITRE ATT&CK framework as a guideline of attacker activities. We formalize the scalable threat hunting problem and provide an approximation

algorithm to generate agent hierarchy, evaluation with three simulated attack use cases and DARPA OpTC attack dataset [30] with varying network size and monitoring capacity, and a threat hunting demonstration using the SCAHunter. The monitoring application for threat hunting in distributed large-scale systems must be scalable to handle many event producers (EFA) and selective monitoring of event producers, highly re-configurable to handle on-demand monitoring requests. An efficient event filtering mechanism must be used to reduce event traffic propagation and monitor intrusiveness.

The SCAHunter improves monitoring scalability using an approximation algorithm by the near-optimal composite event detector (CEDA) and primitive event detector (EFA) hierarchy generation. The proposed SCAHunter is scalable since we generate the minimum number of CEDAs that can communicate with the required EFAs considering the hop count among the CEDAs and EFAs. It also reduces event traffic propagation and monitoring intrusiveness by using the hierarchical structure and optimal generation of agent hierarchy. RabbitMQ ensures the security of agent communication.

CHAPTER 4: Prompting LLMs to Enforce and Validate CIS Critical Security Controls

In this chapter, we present the automation of measures and metrics development and corresponding implementation mechanism extraction from LLMs using prompt engineering for critical security controls (CSCs). We also provide an end-to-end system to assess the enforcement of CIS CSCs using developed metrics and distributed hierarchical monitoring system presented in Chapter 3.

4.1 Introduction

Proper security control enforcement reduces the attack surface and protects the organizations against attacks. Organizations like NIST and CIS (Center for Internet Security) provide critical security controls (CSCs) as a guideline to enforce cyber security. Automated enforcement and measurability mechanisms for these CSCs still need to be developed. Analyzing the implementations of security products to validate security control enforcement is non-trivial. Moreover, manually analyzing and developing measures and metrics to monitor, and implementing those monitoring mechanisms are resource-intensive tasks and massively dependent on the security analyst's expertise and knowledge. To tackle those problems, we use large language models (LLMs) as a knowledge base and reasoner to extract measures, metrics, and monitoring mechanism implementation steps from security control descriptions to reduce the dependency on security analysts. Our approach used few-shot learning with chain-of-thought prompting to generate measures and metrics and generated knowledge prompting for metrics implementation. Our evaluation shows that prompt engineering to extract measures and metrics and monitoring implementation mechanisms can

reduce dependency on humans and semi-automate the extraction process. We also demonstrate metric implementation steps using generated knowledge prompting with LLMs.

The Center for Internet Security (CIS) published a set of defense actions that form a group of defense-in-depth best practices known as critical security controls (CSCs) to detect, prevent, respond to, and mitigate cyberattacks [32]. The CIS is a non-profit organization that provides best practices and security benchmarks for IT systems. The CIS Critical Security Controls (CSCs) are a set of 18 controls and 153 safeguards (previously known as sub-control) considered the most effective at mitigating cyberattacks.

Organizations of all sizes widely use the CSCs, and there is a growing body of literature on implementing and enforcing the controls. One of the critical challenges in implementing the CIS CSCs is ensuring that the controls are appropriately implemented and enforced. This can be a complex and time-consuming process, and it is essential to clearly understand the controls and the steps involved in implementing them. Several resources are available to help organizations implement and enforce the CIS CSCs. The CIS provides tools and resources, including a self-assessment questionnaire, a checklist, and an implementation guide [18, 19]. Some third-party vendors offer tools and services to help organizations implement and enforce the controls [20, 21]. Once the CIS CSCs have been implemented, it is crucial to validate the implementation. This can be done through some methods, such as vulnerability scanning, penetration testing, and security audits. Validation helps ensure that the controls are in place and working as intended. Though there are guidelines for implementing CSCs, there is hardly any research on assessing CSC enforcement [22].

The current state of validating the CIS CSCs is still under development. The CIS works with various stakeholders, including government agencies, industry groups, and security vendors, to ensure that the CSCs are up-to-date and reflect the latest threats.

However, it is essential to note that CSCs are not a silver bullet [60]. Organizations must still implement, maintain, and validate the controls to be effective. Organizations face several challenges in validating the CSCs: 1) The CSCs are constantly being updated as new threats emerge, 2) The CSCs can be complex and time-consuming to implement, and 3) There is no single tool or solution that can automate the validation of the CSCs. Despite these challenges, the CIS CSCs are vital to any organization's cybersecurity program. By implementing CSCs, organizations can significantly reduce their risk of a cyber attack. In general, organizations follow below steps to validate the CIS CSCs: 1) Develop a plan for validating the CSCs, which should include a timeline, budget, and resources. 2) Identify the tools and resources needed to validate the CSCs. 3) Work with a qualified security professional to validate the CSCs. 4) Monitor the effectiveness of the CSCs and make adjustments as needed. By following these steps, organizations can ensure that the CIS CSCs are validated and that their cybersecurity posture is improved.

The implementation and enforcement of the CIS CSCs is an ongoing process. Organizations should regularly review the controls and ensure they remain relevant to their security posture. They should also make sure that the controls are appropriately implemented and enforced. However, no well-defined automated measures and metrics are developed to validate the enforcement of these CSCs. Directly analyzing the implementation of security products to verify and validate the enforcement of those CSCs in security products is an infeasible task. Though guidelines exist to develop measures and metrics to assess CIS CSC enforcement by checking configuration or benchmark validation [61, 19], those guidelines are limited and provided as a sample way to generate measures and metrics. Additionally, those sample measures and metrics check whether a configuration complies with guidelines (Yes or No answer). However, a configuration check will not answer how well a CSC safeguard is enforced (enforcement quality).

In a traditional approach of CSC safeguard enforcement assessment, a security analyst manually analyzes by looking through the safeguard description, tries to understand what the cyber observable will be seen in the system in the presence/absence of this safeguard, determines quantifiable features, and combines those together to generate metrics [52]. For each quantifiable feature, the analyst also needs to know how to measure this specific feature. For example, in safeguard 5.3: "Delete or disable any dormant accounts after a period of 45 days of inactivity, where supported", to assess enforcement of this safeguard, one of the metrics is the percentage of dormant accounts that are still active [62]. To calculate this metric, the analyst needs to know how many dormant accounts are in the system (a measure) and how many are still active (another measure). To estimate the above two measures, the analyst needs to know the cyber observable that will be present in the system if this safeguard is enforced or not enforced. To determine the dormant account in the system, the cyber observable is the presence/absence of dormant accounts, which the analyst can infer by looking at the safeguard description and using his prior knowledge about the user account. Next, the analyst tries to determine how to detect dormant accounts in the system. If the analyst has previous experience in account management or by using Google search, research articles, and papers, he can know the way to determine a dormant account by checking the last activity time of each user account. The next question he tries to answer is how to get a user's last activity time in the system. If the analyst knows ETW (Event tracing for Windows)/auditd (Linux) audit logs, he will see that he can look at the logon, logoff, account delete, and account disable event logs. To monitor those event logs, he also needs to determine which ETW provider will provide those event logs so that he can monitor specific ETW providers to collect particular event logs. After deciding on all the above information, he can either use existing monitoring tools to collect those logs or implement his solution for monitoring those specific events to quantify cyber observable features. In other

words, the security analyst must use his expertise in addition to external materials (Google search, blogs, research articles, etc) to implement specific safeguards, which is time-consuming and highly dependent on the analyst's expertise.

We propose to minimize the dependency on the security analyst's expertise by solving the following challenges: 1) automated extraction of measures and metrics from safeguard description, 2) automated extraction of knowledge base of facts (e.g., security best practices, event monitoring approach, and event ID in event logs) to develop enforcement monitors. Automating the development of measures, metrics, and corresponding enforcement monitors delivers immense value to vendors and enterprises.

Recent advancement in natural language processing opens the door for using large language models (LLMs) as an oracle for such tasks, which can answer questions about specific facts and will act as a knowledge base of facts. The training process of LLM, such as LLAMA and ChatGPT, includes huge corpus and text data and different security controls. The LLM achieves the reasoning capability as an emergent property [37]. Therefore, we leverage LLM to extract measures, metrics, and monitoring implementation mechanisms in this chapter. To the best of our knowledge, this is the first work in the direction of using LLM to generate measures and metrics and elicit metrics implementation steps from LLM through prompt engineering.

We make the following technical contributions:

- We propose a CSC safeguard ontology: the things to extract from each safeguard description. We provide a prompting template used to extract CSC safeguard ontology where CSC ontology will help develop a chain-of-thought (CoT) prompt to extract implementation steps for a CSC safeguard enforcement.

- We provide a few-shot prompt to extract measures and metrics given the safeguard description and dependent safeguard. This prompt generates new measures and metrics for safeguard enforcement compliance and safeguard enforce-

ment quality.

- We provide a prompting template for evaluating LLM-generated measures and metrics to reduce human effort on manual evaluation where a different LLM is used for evaluation. With the help of Spearman, Pearson, and Kendall Tau's correlation coefficient value, we showed that the LLM evaluation aligns with human evaluation.

- We demonstrate CSC safeguard enforcement implementation for multiple measures and metrics of a safeguard with the help of chain-of-thought and generated knowledge prompting.

This chapter is organized as follows: Section 4.2 surveys existing research work on CIS critical security controls and prompt engineering with LLMs to extract information and reasoning; Section 4.3 describes proposed CSC enforcement validation approach; Section 4.4 provides a case study of the whole CSC enforcement validation approach; Section 4.5 evaluates LLM generated measures and metrics using another LLM as evaluator through prompt engineering and provides a demonstration of CSC safeguard enforcement assessment; Section 4.6 summarizes our contributions and limitations. In this chapter, all prompts are developed and executed in GPT-3.5. We will use LLM and ChatGPT interchangeably for the rest of the chapter.

4.2 Related Works

Critical Security Control. In [21], the authors map the organization's security requirements with standard security control frameworks such as NIST and CIS's top 20 critical security controls. In [60], the authors collect security professional's views on the value of security controls to defend against attack by performing interviews. They conduct interviews to determine trends and the effectiveness of CIS's critical security controls. The authors of the survey showed that the top three security controls for effectiveness are CSC 3 (Vulnerability assessment), CSC 4 (Admin privileges), and

CSC 19 (Incident response). They also found out that the most commonly deployed security controls are CSC 8 (Malware defenses), CSC 12 (Boundary defenses), CSC 7 (Web and email defenses), and CSC 11 (Secure network devices). In [63], the authors used the CIS benchmark to audit the Linux operating system with Chef InSpec by checking specific OS configurations. Though the CIS benchmark enforces a particular part of specific security control, it does not cover the whole scope of the corresponding security control. Dutta *et al.* [64] manually map threat actions to security controls. They try to identify appropriate security controls and where and why they should be implemented to optimize risk mitigation. However, they assumed the effectiveness score of security control is provided to their framework and did not validate the effectiveness of the corresponding security control. Ahmed *et al.* [52] manually analyze CSC 12 (Boundary defense) and provide measures and metrics to assess specific sub-controls using their own knowledge and expertise.

Prompt Engineering. Mishra *et al.* [35] identified five strategies to develop a prompt so that the LLM can follow it easily for both zero-shot and few-shot prompting: 1) instead of using terms that require background knowledge to understand, use various patterns about the expected output, 2) turn descriptive attributes into bulleted lists. If there are any negation statements, turning them into assertion statements makes the LLM to follow easily; 3) break down the task into multiple simpler tasks; 4) add explicit textual statements of output constraints, and 5) customize the instructions so that they directly speak to the intended output. Brown *et al.* [37] shows that LLMs (GPT-3) are few-shot learners. The LLM works well when a few demonstrations of the task in inference are provided in the prompt. Wei *et al.* [34] shows that demonstrating the thought process in the prompt to the LLM elicits reasoning capabilities as an emergent property of LLM. To remove the bias of LLM during few-shot prompting towards predicting certain answers, such as those that are placed near the end of the prompt, the authors use contextual calibration [36].

The authors at [65, 66] used the LLM as an evaluator to evaluate text generated by another LLM. Stern *et al.* [67] used prompt engineering with LLM to classify text using few-shot prompting.

All of these works try to identify which security controls are essential to implement in an organization to defend against attack and develop assessment approaches manually, which is a time-consuming and resource-intensive task. However, no previous research has explored the automated assessment of the enforcement of security controls to reduce high reliance on the expertise of security analysts.

4.3 Overview of the CSC Validation

This section outlines our CSC validation approach and its goals, as illustrated in Figure 4.1. The CSC validation consists of the following modules: 1) prompt engineering to extract key measurement indicator (KMI) (measure) and key enforcement indicator (KEI) (metrics) generation, 2) Event Subscription Rule (ESR) generation and monitoring of cyber observables using CSCMonitor, and 3) CSC validation using the generated KEIs. At first, we analyze the CSC safeguard description to extract KMI and KEI both manually and by prompting an LLM. To pinpoint specific details to be extracted from safeguard descriptions, we develop a CSC ontology that outlines the relevant information for each safeguard.

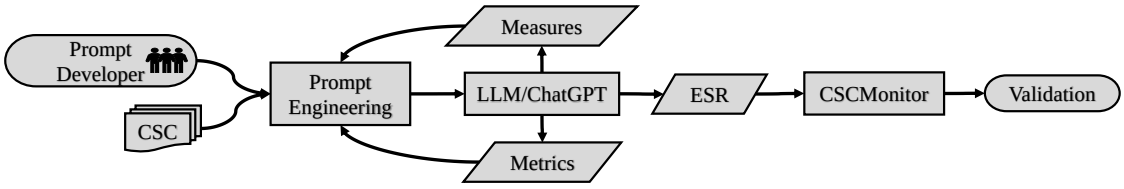

Figure 4.1: CSC validation approach

During prompt engineering with a large language model (LLM), we utilized the CSC ontology that was manually extracted to create the prompt template. Employing techniques such as few-shot prompting, chain-of-thought (CoT) prompting, and generated knowledge prompting, we developed measures and metrics to evaluate the

enforcement of safeguards.

Following the generation of measures and metrics, we established an event subscription rule (ESR) for each measure and employed CSCMonitor (Section 4.3.4) to gather data related to the measure. Ultimately, we assess the enforcement of the safeguard by using the metrics generated and the statistics gathered for each measure. In Figures 4.1 and 4.3, various shapes represent different elements: ovals indicate the start or end, rectangles depict processes, diamonds denote decision points, and parallelograms signify inputs or outputs. The subsequent subsections will detail each phase of the CSC enforcement validation process.

4.3.1 CSC Ontology

To determine critical information that assists in identifying KMI (measure) and KEI (metric) for each safeguard, we manually examined all 153 safeguards. During this manual review of the safeguard descriptions, we sought to answer several key questions: 1) What are the threat actions and related cyber observables targeted by each safeguard? 2) What are the KMI or cyber-measurable features that need assessment for each safeguard? 3) What category does the measurement fall into, and what approach is used for measuring?

After identifying cyber observables for each safeguard, we categorized each CSC safeguard into four classes based on the methods for measuring or verifying the corresponding cyber observables. The first category is *General*, which refers to safeguards that provide broad guidelines without specific details on the defense action. The second, *Checklist*, involves safeguards that can be verified through scripting, like CIS CSC safeguard 6.1. A safeguard falls under *Verifiable* if it can be verified through configuration and vulnerability analytics, such as analyzing network configurations to identify reachable hosts in safeguard 12.1. The last category, *Measurable*, applies to safeguards that can be verified through quantitative data analysis, using logs and network traffic. For example, the use of a time server in Safeguard 6.1 can be determined

by analyzing network traffic logs to known time servers.

Additionally, by analyzing the CSCs, we have identified five types of enforcement measurement approaches. One approach is *Vulnerability Analytic*, which utilizes the targeted service's vulnerability score (CVSS) to assess the deployed CSC's effectiveness. The *Data-driven* method measures the effectiveness of a safeguard using network traffic statistics, including NetFlow data, DNS traffic, and web traffic data. The *Model-based* approach employs system and network configurations to confirm the enforcement of the safeguard. The *Active Testing* approach involves sending a probe to validate a safeguard. Lastly, the *Cyber Threat Intelligence*-based approach uses indicators of compromise (IOCs), threat feeds, and insights on malicious actors' intents, opportunities, and capabilities from CTI reports to evaluate the effectiveness of CSCs.

Therefore, our CSC safeguard ontology consists of cyber observable, class, and measurement approaches.

4.3.2 KMI and KEI Extraction and Measurement: Manual Approach

This section outlines our manual efforts to determine metrics and measurement procedures for each CSC safeguard to ensure their effective deployment (Figure 4.1). Specifically, this section addresses the following objectives to identify metrics and measurements to validate CSC enforcement in the security product under test (PUT):

- Identify the threat actions targeted by each CSC safeguard.
- Identify key measurement indicators (KMIs) or cyber-measurable features for each CSC safeguard.
- Determine the cyber observables of each CSC safeguard that can be used to identify KMIs.
- Determine how to compose KMIs to develop metrics for the key enforcement

indicators (KEIs), which will quantitatively assess the quality attributes of each CSC safeguard enforcement.

- Determine the measurement category and the measurement approach for each CSC safeguard.

To determine critical information to extract from each CSC safeguard, we manually reviewed all 153 safeguards. Based on our observation from manual analysis, we detect four classes of CSC safeguard: General, Checklist, Verifiable Properties, and Measurable, and five different measurement approaches: Data-driven, Model-based, Active Testing, Vulnerability Analytic, and Cyber Threat Intelligence. The observable, safeguard class, and measurement approach are key in determining the measures (cyber measurable features) to monitor for assessing the enforcement quality of a safeguard. Consequently, each CSC safeguard is mapped to a specific observable, class, and measurement approach as depicted in the CSC Ontology shown in Figure 4.2.

To develop the measures and metrics for the validation of CSC enforcement, the first step is to identify the threat model of each CSC safeguard that determines what the adversary can do in the absence of this specific CSC safeguard. For instance, the absence of CSC 13 (Data Protection) implies the lateral movement of the attacker and exfiltration of sensitive data. Subsequently, KMIs will be determined based on the identified cyber observables or artifacts linked to each CSC safeguard. Each KMI is a concrete and objective attribute (measurable attribute) for the corresponding CSC safeguard, such as the number of detected malicious IP addresses and the number of the unused IP addresses in the target organization. The following step is to compose different KMIs to develop metrics for KEIs which will be used to assess the quality of the CSC enforcement. Each KEI is an abstract and subjective attribute, such as coverage or percentage of malicious IP addresses that can be detected, and freshness or how fast a new asset is discovered by the PUT.

4.3.3 KMI and KEI Extraction and Measurement: Prompting the LLM

The CIS provides CSCs as a guideline of best security practices for defending an organization against threats. However, security analysts need to check how well security control is implemented in an organization. To determine the enforceability (yes/no) and enforcement quality of a CSC safeguard, a security analyst needs to identify cyber-measurable attributes to monitor, along with measurement metrics and security configurations to examine. Recent advancements in natural language processing open the door to the use of a large language model (LLM) as an oracle for such tasks, which can answer questions about specific facts and act as a knowledge base of fact. The training process of LLM, such as LLaMA and ChatGPT, includes huge corpus and text data and different security controls. We prompt LLM to extract critical information from the CSC description and generate measures and metrics for each safeguard to reduce the dependency on the security analyst's expertise.

Given a CSC safeguard description, we generate a list of measures and metrics for the corresponding safeguard by using zero-shot and few-shot prompting. However, those KMI are not implementable and require security analyst's help to determine specific data sources and attributes to measure. To remove the need for expert knowledge, we perform additional chain-of-thought (CoT) prompting to determine how to implement the monitoring steps of a KMI.

CoT is an emergent ability of model scale such that it does not positively influence performance until used with a model of sufficient scale. To improve performance, the LLM parameter size needs to be greater than 130B. According to [68, 34], CoT elicits reasoning as an emergent property of LLM. According to [34], CoT prompting offers several beneficial features for enhancing reasoning in language models. Firstly, it allows models to break down complex, multi-step problems into simpler, intermediate steps, which can be especially helpful for tasks that require additional computational reasoning. Secondly, it provides a transparent view into the model's thought process,

showing potential paths to a solution and offering a means to troubleshoot incorrect reasoning paths, although fully understanding the underlying computations remains a challenge. Thirdly, this method is useful for tackling tasks like math word problems, commonsense reasoning, and symbolic manipulation, and could theoretically be applied to any problem that humans can solve using language. Lastly, chain-of-thought reasoning can be easily triggered in large pre-trained language models by incorporating chain-of-thought examples into the training data used for few-shot prompting.

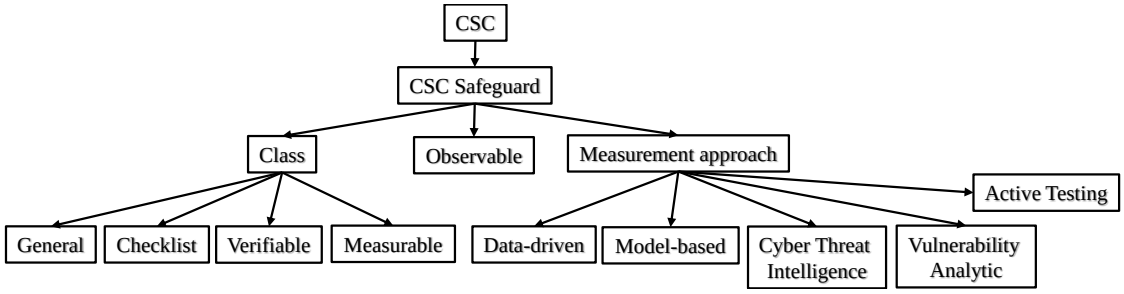

Figure 4.2: CSC Ontology

Prompting for Safeguard Ontology: We generated the CSC ontology by manually analyzing all 153 CSC safeguards. CIS updates and even rearranges the description of the safeguard over time. For example, we started our manual analysis with CIS CSC version 6 and the currently updated CIS CSC version is 8. In this new version, the CSC safeguard description is updated. In such a scenario, we generate new ontology by prompting LLM with a few-shot prompting technique, providing examples of manually extracted CSC ontology for a different safeguard as described in [34]. We develop a prompt template (Section 4.4) consisting of all the fields of CSC ontology for a target safeguard; to provide the chain of thoughts during few-shot prompting, we also give an example (Section 4.4) with extracted ontology and human reasoning steps during extraction as input. During this prompting step, we used a few examples of manually generated CSC ontology to demonstrate reasoning steps to LLM during the CoT prompting process. This generated CSC ontology will be used to generate a prompt to extract measure implementation.

Prompting for Measures and Metrics Generation. By following the same approach of CoT prompting described above for ontology generation, we use CoT prompting to generate measures and metrics. We develop a prompt template (Section 4.4) consisting of measures and metrics generation thought process. In this prompting, we demonstrate our thought process during manual metrics and measure generation by following the CoT prompting approach in [37]. The prompt provides one example of measure and metric generation by dividing the multi-step task into separate thoughts.

Prompting for Safeguard Enforcement Implementation: Though we generated the measures and metrics from manual analysis and prompt engineering, the measures are not implementable. Expertise and knowledge are still needed to determine and implement specific measures to monitor. For example, controls-assessment-specification [62] from CIS provides a metric for safeguard 1.4 to measure DHCP log quality. To calculate the score for DHCP log quality, they used two measures: 1) number of DHCP servers in the organization and 2) number of DHCP servers with DHCP logging enabled. In order to implement the DHCP log quality metric, a security analyst needs to know how to identify a DHCP server and count the total number of DHCP servers. They also need to know which configuration to check to verify if a DHCP server is configured with DHCP logging enabled. If access to configuration is not provided, the analyst needs to know what audit/event logs to examine to identify whether a DHCP server is enabled with DHCP logging. Finally, if the analyst wants to implement the metrics through passive monitoring, they need to be aware of the appropriate approach to do so. In such cases, to alleviate the time required and dependency on an analyst's expertise, we propose to use the LLM as a source of the knowledge base of different metric implementations. We suggest using generated knowledge prompting [38] in association with CoT to elicit metric implementation approaches from the LLM by prompting.

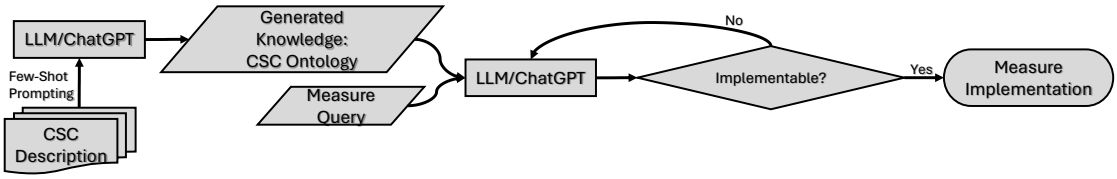

Figure 4.3: CoT prompting flow

To elicit CSC measure implementation using the LLM as shown in Figure 4.3, we first perform few-shot prompting with chain-of-thought to generate new knowledge. In our case, this newly generated knowledge will be CSC ontology. This generated knowledge and specific measure query will be used as context to generate CSC implementation approaches from the LLM. For each prompting in this step, the LLM may generate multiple approaches. The security analyst needs to try all of the approaches or the most viable ones (based on his own judgment) to elicit a measure-specific implementation approach. This process will continue until the security analyst is satisfied with the specific granular-level measure implementation approach. Here, using the LLM as a knowledge base and reasoner rather than depending on human expertise and Google search for research articles will reduce the amount of time to discover the implementation approach.

4.3.4 CSCMonitor: Hierarchical Monitoring of Extracted Measures

Though monitoring the cyber measurable attribute to calculate the measure statistics can be done using a centralized monitoring system and security information and event management (SIEM) technology such as SPLUNK, it incurs additional communication overhead and memory usage and is unsuitable for selective monitoring. We used a distributed hierarchical event monitoring agent architecture to overcome those challenges as proposed in Chapter 3. Our hierarchical event monitoring agent architecture consists of three types of agents: 1) Console agent (CA): This agent works as a manager to decompose monitoring tasks to primitive events and assign each decomposed task to a lower-level agent. It also determines the appropriate agent

hierarchy based on the correlation among the decomposed tasks and the host location. 2) Composite event detector agent (CEDA): this agent correlates detected events in lower-level agents and forwards the detected subscribed task result to the upper-level agent (CEDA or CA). There can be multiple levels of CEDA agents based on the monitoring task (measure signature). 3) Event filtering agent (EFA): this agent is a lower-level agent that ingests event logs directly from the event producer (ETW, Netflow and auditd, etc.) and monitors primitive events, and publishes the detected events to the higher level agent (CA or CEDA) based on the subscribed task. Such hierarchical event monitoring will improve monitoring scalability and reduce communication and memory usage overhead by enabling local decision-making.

4.4 CSC enforcement validation using Prompt engineering: a case study

To demonstrate CSC enforcement validation implementation using prompt engineering with the LLM, this section provides a step-by-step prompt for extracting CSC ontology, generating measures and metrics using extracted ontology, and, at the end, generating metrics implementation approaches with the help of generated knowledge prompting in association with chain-of-thought prompting. This section will demonstrate CSC safeguard implementation through prompt engineering using CIS CSC safeguard 1.1, 12.2 from version 8, and sub-control 12.1 from version 7.

To generate CIS CSC enforcement implementation, the first step is to extract knowledge (in our case, it is the CSC ontology) from the CIS CSC safeguard description. After generating the CSC ontology for each safeguard, we will create measures and metrics for each safeguard using the extracted safeguard ontology as the generated knowledge. We use generated knowledge to develop the measures and metrics; we use both zero-shot and few-shot prompting. In our observation, the LLM can not extract CSC ontology with zero-shot prompting if we only provide the definition of each field of CSC ontology. However, the LLM can generate feasible CSC ontology if we give a few manually extracted CSC safeguard ontologies as examples and ask for

CSC ontology for a different safeguard that aligns with the current knowledge about LLM, which states that LLMs are few-shot learners [37].

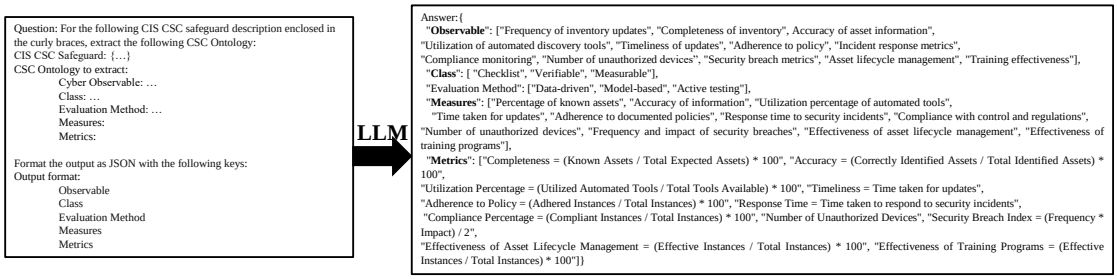

Figure 4.4: Zero-shot prompting for CSC Ontology

The first step in extracting CSC ontology (generated knowledge) from the safeguard 1.1 descriptions is to create a prompt template. The template consists of a query, ontology definition, and output format. The definition of each of the ontology fields is given in Section 4.3.2. The input prompt and output from the LLM are given in Figure 4.4. The result shows that the LLM could not distinguish between measures and metrics where measures are objective and metrics are subjective, a combination of multiple measures. This is a zero-shot prompting since we did not provide any thought process or examples of extracted CSC. The LLM could not distinguish between measures and metrics in this zero-shot prompting. Some of the measure outputs in Figure 4.4, such as "Percentage of known assets", "Accuracy of information", "Effectiveness of training program", and "Effectiveness of asset lifecycle management", are also in the metrics outputs. One reason for such a wrong extraction could be that the LLM could not distinguish measures and metrics when only a definition is provided. However, when we offered a few examples of extracted CSC ontology, it could follow the extraction strategy we used in the examples and successfully identify accurate measures and metrics. We also observe that when the context of a query is extended, the LLM emphasizes the context more at the end of the context window as suggested in [36]. For example, to extract CSC ontology by following the CoT prompting in Figure 4.5, we have to provide a few examples

of the CSC ontology extraction process. When we give multiple safeguard ontology extraction as examples, the LLM's answer is more aligned with the last example.

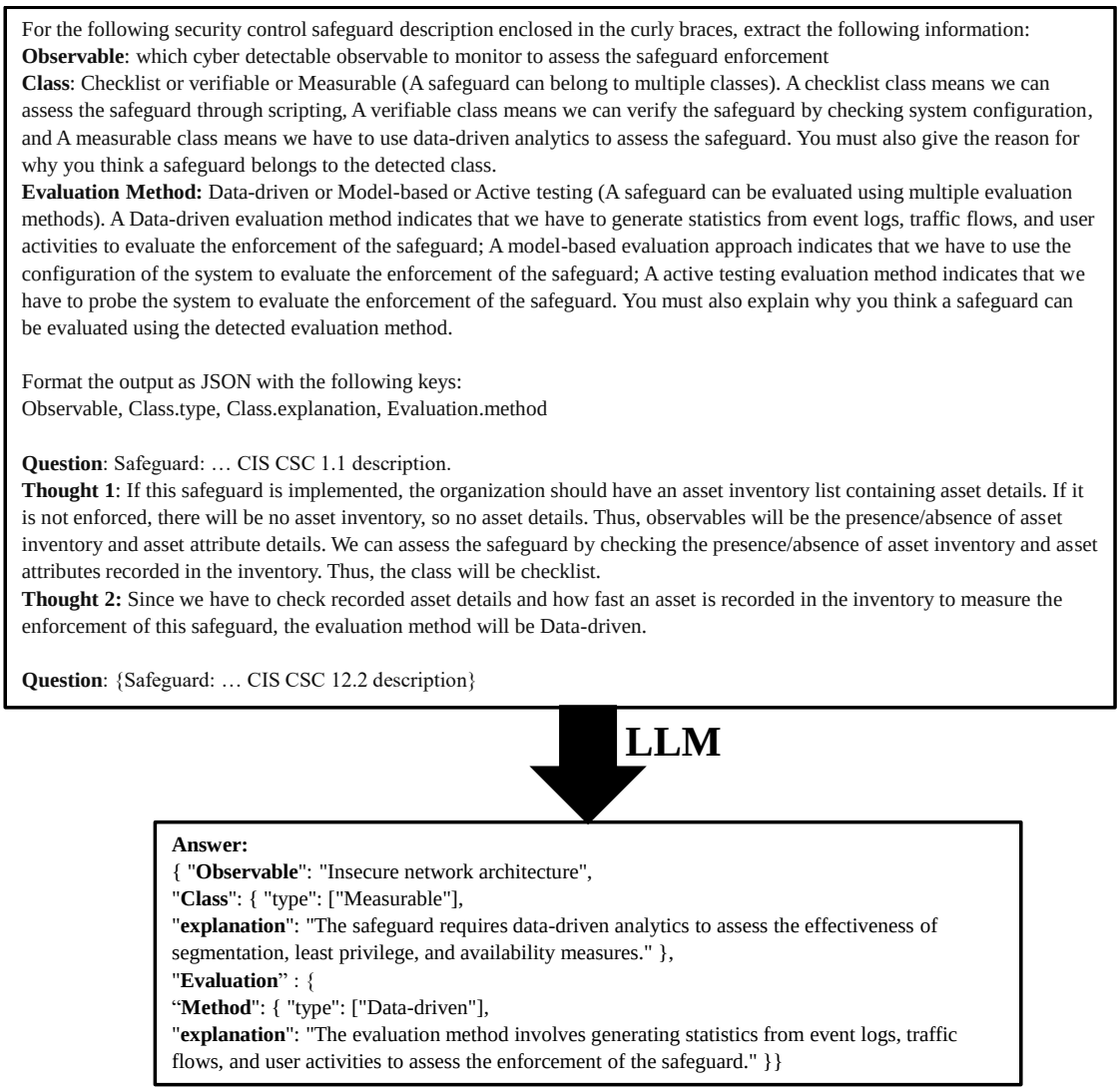

Figure 4.5: CoT prompting for CSC Ontology

Two-step prompting. To overcome this limitation, we divide our CSC ontology extraction prompt into two parts. In the first step, we extract observable, class, and evaluation methods by providing extraction examples of those three CSC ontology fields. Then we use those extracted CSC fields as generated knowledge context and create another prompt that queries about measures and metrics. In this prompt, we provide a chain of thought for extracting only measures and metrics by offering

examples of a specific safeguard and queries about measures and metrics of a different safeguard. We also provide the extracted fields of this safeguard in the previous prompt as context. As shown in Figure 4.6, this two-step prompting process can generate accurate measures and metrics for a specific CSC safeguard. Our prompting experiment can generate all the existing measures and metrics provided by CIS CSC specification project [21] in addition to other measures and metrics as shown in Tables 4.1 and 4.2.

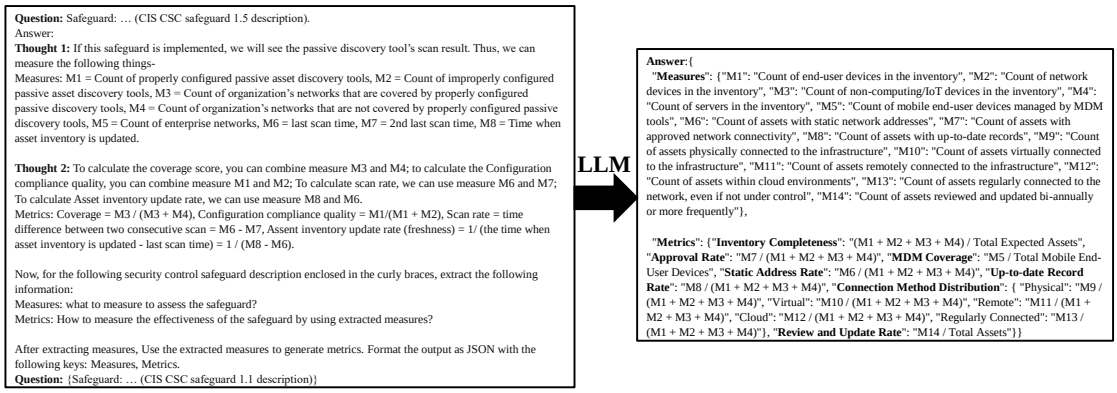

Figure 4.6: CoT prompting for Measures and Metrics

Although we generated measures and metrics by using CoT prompting, the generated metrics and measures are not implementable yet. In order to calculate a score for a metric, an analyst needs to monitor each of the corresponding measures, determine the cyber observable to monitor corresponding to the measure and signature of the observable, and at the end, how to implement these monitoring steps. To extract measure and metric implementation steps, we use a sequence of prompts in the fashion of generated knowledge prompting as mentioned in [38].

To demonstrate measure implementation using generated knowledge prompting as shown in Figure 4.7, we use CSC sub-control (safeguard) 12.1 from CIS CSC version 7. One metric to measure this safeguard's enforcement quality is to calculate network boundary asset inventory accuracy. To calculate the score, we have to determine two measures: 1) the number of total network boundaries in the inventory list and

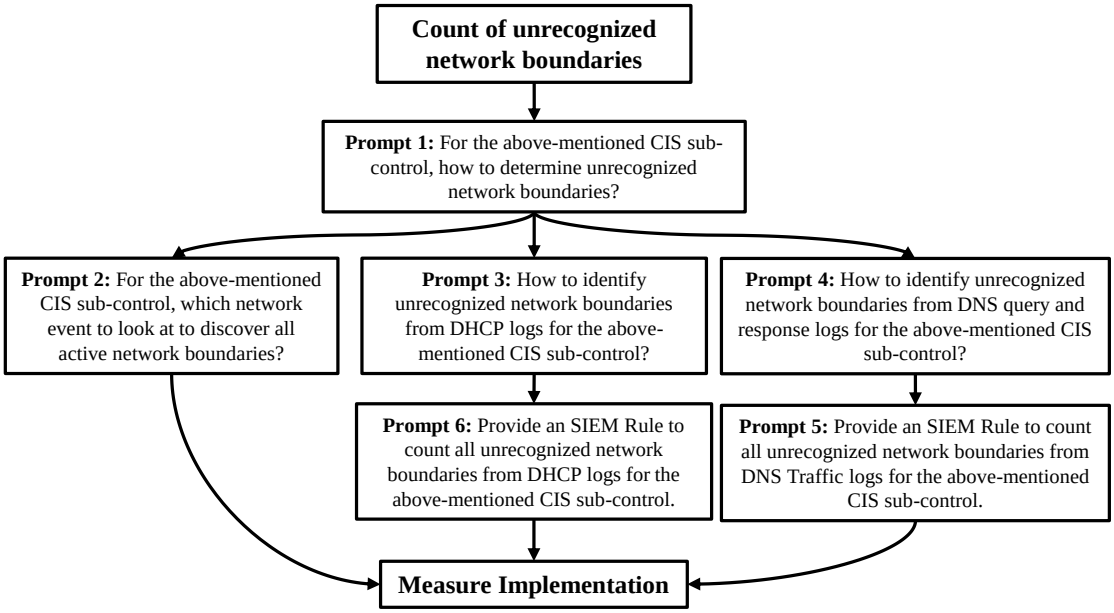

Figure 4.7: Generated knowledge prompting for Metric Implementation

2) the number of unrecognized network boundaries in the network. To monitor the two measures mentioned earlier, the security analyst needs to know what the cyber observable is to monitor to detect network boundaries and how to implement these monitoring steps. In such cases, we use LLM to generate implementation steps and use LLM as both reasoner and knowledge base. In the first prompt, we will query the LLM about determining network boundaries. In our observation, LLM provides multiple approaches to detecting network boundaries.

However, the answer is vague at this prompting level, and we have to dig deeper to extract specific implementation steps. In our case, we query the LLM for corresponding network events and data sources. The LLM outputs specific network events and two data sources for those events: 1) DHCP logs and 2) DNS query logs and responses (prompts 2, 3, and 4 in Figure 4.7). At this step, the answer of LLM is specific to the measures and is implementable. In the following prompt, we query the LLM about the monitoring signature (SIEM rule) for each type of event and traffic log (prompts 6 and 7). In each prompting step, we use the output from the previous prompt as generated knowledge for the current prompt. This process of prompting continues in

multiple paths until the analyst is satisfied with a specific implementation.

4.5 Evaluation

In this section, we will evaluate the quality of the LLM-generated measures and metrics and demonstrate the metric implementation for CSC safeguard 5.3. Our experiment used three VMs, each with five user accounts.

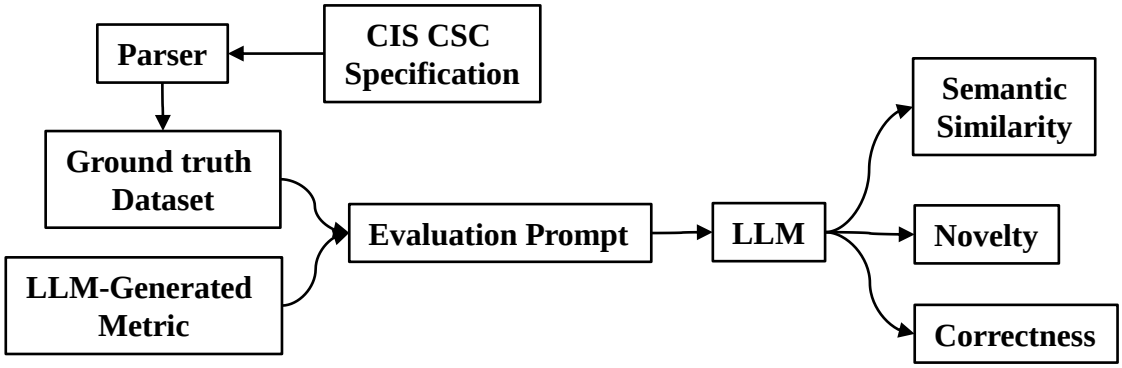

Figure 4.8: Evaluation of generated Metrics and Measures with LLM

In traditional machine learning and deep learning, establishing ground truths is crucial for assessing the accuracy of model predictions or classifications. However, for large language models (LLMs), defining a clear-cut ground truth is more complex. LLMs are typically required to generate human-like text, where there often isn't a single "correct" answer that allows for direct comparison. One way of evaluation is to involve humans and ask them to rate the LLM responses. This is what most of us do when we manually look at the responses of two models or two prompts and determine which one looks better. Although this approach is quick to start, it quickly becomes time-consuming and highly subjective. Given that RLHF (Reinforcement Learning from Human Feedback) involves creating an (LLM) reward model that scores model responses during training, this shows a new direction to use LLMs themselves as evaluators [65, 66].

We prompt the LLM to act as an evaluator, comparing the gold standard answer with the new LLM-generated answer. The LLM is tasked to respond with a binary

'Yes' or 'No' for the key features being evaluated (semantic similarity, novelty, and correctness in Figure 4.8) with rationale. These three metrics have been chosen so we will be able to evaluate how semantically similar the LLM-generated output is to the human-generated output, whether the LLM can create new measures and metrics that are not generated by humans, and whether the LLM can compose measures correctly to generate metrics when compared against our ground truth [66].

We used the CIS control-assessment-specification [62] GitHub repository to generate the golden dataset as the ground truth of LLM-generated measures and metrics as shown in Figure 4.8. This project provides sample measures and metrics for each CIS CSC safeguard. We collected measures and metrics for each safeguard by crawling the project and parsing it using a Python script. In addition to the parsed measures and metrics from the repository, we used observable, measurement approach from our manual analysis results. This parser generates a JSON document for each safeguard containing observables, measurable class, measurement approach, measures, and metrics.

One way to evaluate the generated measures and metrics is to manually compare them with the corresponding ground truth, a.k .a. the golden dataset. However, the manual comparison is a time and resource-intensive task. Thus, it does not scale with the increasing number of safeguards. There exist benchmarking frameworks such as ROUGE and BLEU to evaluate text generation systems. However, those benchmarking frameworks evaluate at the word level or look for syntactic or lexical similarity between generated text and ground truth text. Since it is improbable that LLM-generated text will match with ground truth at the lexicon level, we are interested in evaluating the semantic similarity of the generated text. By following the approaches in LLM-EVAl and Flask [65, 66], we used the LLM as an evaluator for the generated measures and metrics. We evaluated the LLM-generated measures and metrics in the following criteria: 1) semantic similarity, how semantically similar

the LLM and golden dataset are; 2) metrics correctness: whether the measures are composed correctly to generate metrics in the LLM-generated measures and metrics; 3) novelty: whether the LLM suggests new measures and metrics other than the one mentioned in the golden dataset. We use an evaluation prompt as shown in Figure 4.9 where we prompted the LLM to assign a score against each criterion by comparing the LLM-generated measures and metrics and golden dataset for a specific safeguard. In this prompting, LLM is used as the judge or evaluator of measures and metrics generated in a different prompt. However, the evaluation score generated by LLM does not guarantee any reliable scoring. To check whether the evaluation generated by the LLM is aligned with human evaluation, we manually assign scores for 10 safeguards in each criterion by comparing LLM-generated metrics and measures with the golden dataset. Ultimately, we calculate the Pearson, Spearman, and Kendall Tau correlation coefficients using human and LLM evaluation scores.

Semantic Similarity: From the result in Figure 4.10 (a), we found that LLM-generated metrics differ from the human-labeled ones when limited context is given. For example, human-generated metrics for safeguard 5.6 are different from the LLM-generated ones because the dependent safeguard is considered when humans are generating the metrics, but for LLM, we did not provide any dependent safeguard for a specific safeguard. Thus, LLMs do not generate good metrics if the safeguard description is not enough (to generate metrics for this safeguard, we have to consider some dependent safeguards). However, when any dependent safeguard consideration is unnecessary, the LLM-generated metrics cover all the human-generated metrics (e.g., safeguards 5.1 to 5.5).

Novelty: LLM-generated metrics provide new metrics for all of the safeguards we evaluated (Figure 4.10 (b)). Since the human-generated metric is incomplete and provided only as a guideline for security analysts, the LLM always generates new metrics based on the safeguard description.

Given the context, LLM generated response and human-annotated response, evaluate LLM-generated response with respect to annotated response based on the following criteria:

1. Semantic similarity: Is the LLM-generated response semantically similar to the annotated response? While calculating the semantic similarity score, ignore the measure name mismatch. Provide a score between 1 to 10.
2. Novelty: Does the LLM-generated response contain new metrics that are not mentioned in the annotated response? Provide a score between 1 to 10.
3. Metrics correctness: are the measures composed correctly to generate metrics? Provide a score between 1 to 10.

Context: safeguard description
LLM-generated response: "…"
Human-annotated response: "…"

Provide the score with explanation for each criterion in JSON format.

LLM

Answer:
Semantic similarity: {Score:…, Explanation: …}
Novelty: {Score:…, Explanation: …}
Metrics Correctness: {Score:…, Explanation: …}

Figure 4.9: Prompting to evaluate Measures and Metrics

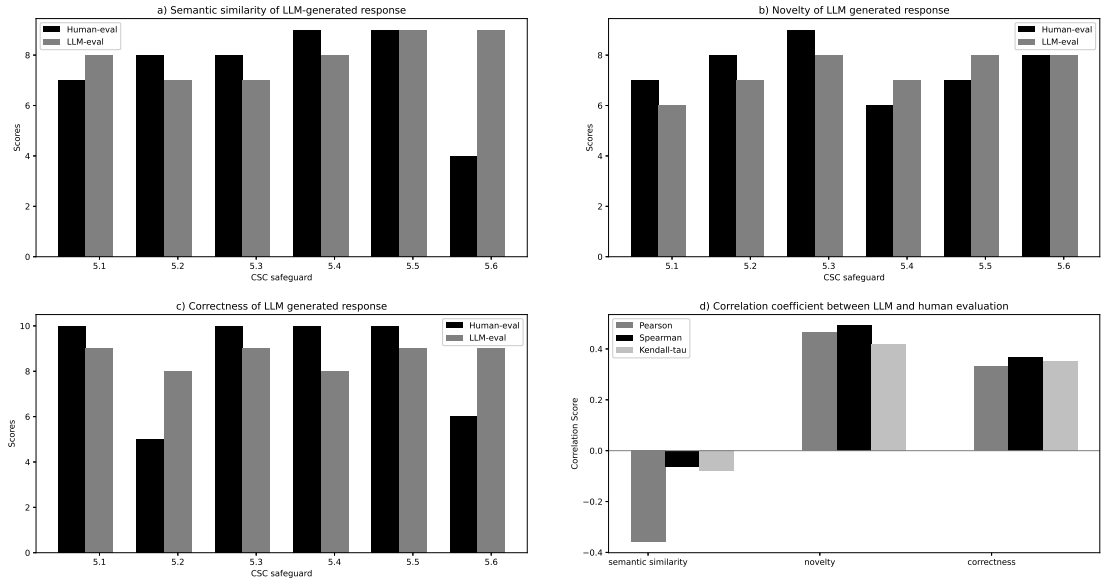

Figure 4.10: Semantic Similarity, Novelty, Correctness evaluation between LLM-generated and human-labeled metrics, and Correlation between human evaluation and LLM evaluation (all the evaluation done with ChatGPT-3.5)

Correctness: To validate the correctness of LLM-generated measures and metrics, we asked the LLM evaluator through zero-shot prompting by following the prompt in Figure 4.9. We also manually evaluated the LLM-generated measures and metrics. The goal here is to check whether LLM can compose different specific measures together to generate metrics. From Figure 4.10 (c), we see that the correctness score of safeguards 5.1 and 5.3 to 5.6 for both manual and LLM are high (more than 5). Initially, the correctness score for LLM-generated metrics for safeguard 5.2 is less than 5. We investigated it and found out that the metrics for safeguard 5.2 in the golden dataset are wrong; after fixing the wrong metrics, the correctness score for the LLM increased to 8.

Reliability: We evaluated the quality of the LLM-generated measures and metrics using another LLM. However, whether the evaluation of LLM is reliable or not is an important question. To confirm the LLM's evaluation aligns with human understanding and assessment of LLM-generated measures and metrics, we manually provide a score of 0-10 under semantic similarity, novelty, and correctness criteria for each safeguard measure and metric. We also evaluate the LLM-generated metrics using zero-shot prompting (prompt in Figure 4.9) for the same safeguard. We did this experiment for 10 safeguards. We calculate the Pearson, Spearman, and Kendall Tau correlation coefficient to determine the correlation between human and LLM scoring. Since from the limited evaluation, it is not clear whether the parametric or non-parametric correlation approach is the appropriate one (normal distribution and linear relationship of the score under each criterion), we calculated the correlation coefficient for both parametric (Pearson) and non-parametric (Spearman and Kendall Tau) approach.

From Figure 4.10 (d), we have a Pearson, Spearman, and Kendall Tau correlation coefficient of -0.39, -0.06, and -0.08 for semantic similarity evaluation. The negative rho value indicates the LLM evaluation does not align with the human evaluation

of the LLM-generated measures and metrics. We further investigate the human-labeled dataset and the LLM-generated one. Human-labeled measures and metrics for safeguard 5.6 are quite different. Humans consider other dependent safeguards to generate the measures and metrics for safeguard 5.6, but LLM generates the safeguard only considering safeguard 5.6. Since LLM did not get the dependent safeguard as context while generating the measures and metrics, the measures and metrics are very different from the human-labeled one. During the evaluation of semantic similarity evaluation in Figure 4.10 (a), we got a score of 5 and 8 for human and LLM evaluation, respectively. To overcome this discrepancy, we regenerate the measures and metrics by providing dependent safeguard as context and evaluate it using both LLM and humans. This time the rho value improved to 0.6, 0.5, and 0.3 for Pearson, Spearman, and Kendall Tau correlation coefficient. This positive correlation coefficient indicates that using LLM as an evaluator is reliable and aligns with human evaluation.

From Figure 4.10 (d), the correlation coefficient of 0.47, 0.49, and 0.42 for Pearson, Spearman, and Kendall Tau indicates that LLM evaluation for novelty aligns with human evaluation of LLM-generated measures and metrics. Similarly, the correlation coefficients of 0.33, 0.37, and 0.35 for Pearson, Spearman, and Kendall Tau indicate that LLM evaluation for correctness aligns with human evaluation of LLM-generated measures and metrics.

Improvement over Different Language Models. So far, all the responses we discussed were generated using ChatGPT-3.5 (gpt-3.5-turbo-0125). To see the improvement in similar tasks among different LLMs, we further executed the same prompts with ChatGPT-4 (gpt-4-0613). The visible improvement of ChatGPT-4 is in extracting CSC ontology (observable, measurable features). When extracting observables, ChatGPT-3.5 sometimes reports the wrong observable whereas ChatGPT-4 always reports the correct observable (for all 6 safeguards we tested). In terms of measures and metrics generation, the answer from ChatGPT-4 covers all correct ones

from ChatGPT-3. Moreover, It provides more accurate metrics than the ones generated by ChatGPT-3.5. All chat transcripts are available at [69].

General Applicability of our Prompts for Security Controls Provided by Entities other than CIS. While this chapter primarily presents use cases and illustrations related to CIS CSC, our methodology can generally be applicable to other security control guidelines, provided that these guidelines offer a comparable level of specificity in delineating observables and features. We generate measures and metrics for NIST CSF controls using our original prompts [69]. Specifically, we generate measures and metrics for NIST CSF controls PR.AC-1, PR.AC-4 and PR.AC-5 by prompting ChatGPT-3.5 and ChatGPT-4. Our original prompts for CIS CSC can generate measures and metrics for all three NIST CSF security controls with the same accuracy. However, the accuracy of ChatGPT-4 is higher than ChatGPT-3.5, which is expected as ChatGPT-4 is a more advanced model. Since we developed the prompts with CoT prompting, the reasoning steps are demonstrated to LLM during prompt generation. We demonstrated the thought process for CIS CSC safeguard and asked LLM to generate measures and metrics for NIST CSF controls. The chat transcripts [69] with ChatGPT-3.5 and ChatGPT-4 show the general applicability of our prompt for security controls provided by organizations other than CIS.

4.5.1 Metric Implementation Demonstration using LLM

In this section, we demonstrate measure and metric implementation with the help of chain-of-thought prompting with LLM. Given a description of the safeguard, we used chain-of-thought prompting to generate measures and metrics for the safeguard to assess the enforcement quality of the corresponding safeguard. However, the security analyst needs to implement those metrics in the organization to assess the enforcement quality of the safeguard. We will demonstrate the process of implementing the metrics for the safeguard 5.3 using LLM.

In the first step, we generate measures and metrics for safeguard 5.3 using the

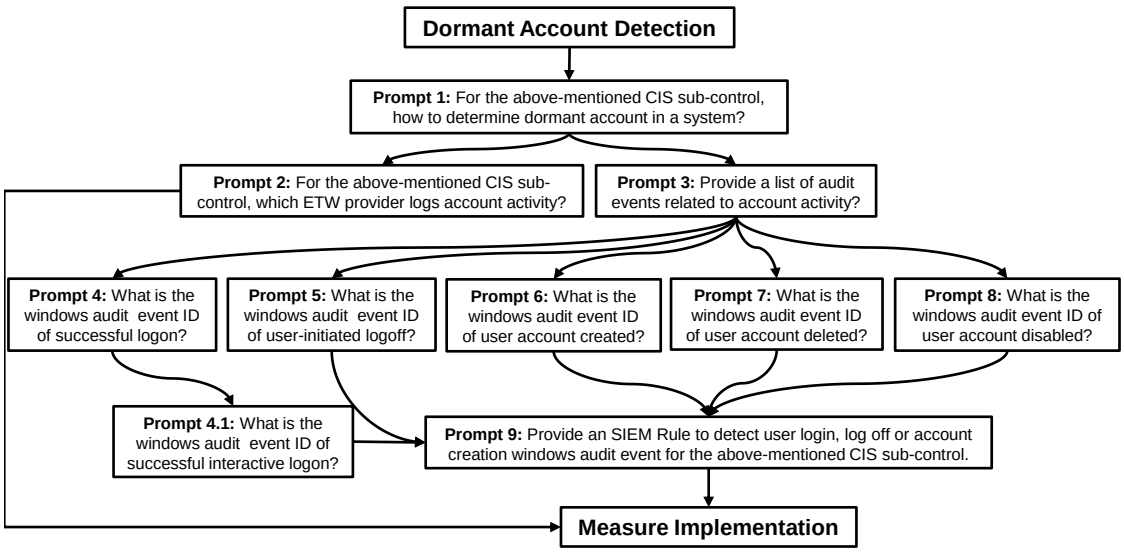

Figure 4.11: Generated knowledge prompting for dormant account detection implementation

prompt in Figure 4.6. The LLM-generated metrics contain three separate metrics, whereas the CIS CSC specification provides one metric. The metrics are 1) Dormant Account Deactivation Compliance, the percentage of deactivated dormant accounts; 2) Timeliness of Deactivation, the percentage of timely deactivation; and 3) Overall Compliance Score, the average of Dormant Account Deactivation Compliance score, and Timeliness of Deactivation score. To calculate metric 1, the LLM suggested two measures: 1) Total dormant account and 2) number of deactivated dormant accounts. To calculate metric 2, the LLM generates two measures: 1) number of deactivated dormant accounts and 2) number of timely (within 45 days) deactivated dormant accounts. Thus, to calculate both metrics 1 and 2, a security analyst needs to monitor 3 measures: 1) Total dormant accounts, 2) the number of deactivated dormant accounts, and 3) the number of timely (within 45 days) deactivated dormant accounts. How to collect statistics corresponding to those three measures highly depends on the analyst's skills and prior knowledge. We will use generated knowledge prompting with an LLM to extract those monitoring implementation steps as shown in Figure 4.11.

From the list of required measures to monitor, we see that we have to implement a way to detect dormant accounts in the system. So the first prompt for LLM will be to ask it about detection techniques for dormant accounts. The LLM provides multiple ways to detect dormant accounts, such as defining dormant criteria, reviewing account activity logs, and checking last login timestamps and account access patterns. Reviewing the LLM answer, we see that all techniques are consolidated around looking at account activity logs. However, at this stage, LLM does not provide details of the account activity logs to look at. So, using our knowledge from the previous prompt answer, we have to generate another prompt to ask the LLM about the specific account activity logs. Since we have to look at the activity logs, we will ask the LLM about the ETW provider that logs the account activity. We also have to ask the LLM about specific account activity event id such as logon event ID (4624), logoff event ID (4634), and account disabled event ID (4725).

At this point of prompting, we know the event IDs and the ETW provider to monitor account activity. We then use CSCMonitor (Section 4.3.4) to implement these monitoring steps. CSCMonitor takes a monitoring task as the signature. We use our extracted required event IDs to build a signature. We can use LLM to build the event signature to monitor. However, the event signature depends on monitoring tools, making generating an exact signature using LLM infeasible. For our work, we manually created the signature using extracted event IDs. After receiving a monitoring task, CSCMonitor decomposes it into primitive tasks to monitor and assign them to lower-level agents. The goal of using CSCMonitor for our monitoring architecture is to reduce transferring excessive amounts of event logs to the central server, i.e., the console agent of CSCMonitor. To detect dormant accounts, we have to know the last account activity for each user account in the system. Among the user activity, we are interested in logon, logoff, and account creation events. Thus, the monitoring

signature for the console agent of CSCMonitor will be:

$$\begin{aligned}(\text{event\_id} == \text{logon} \;\&\&\; \text{logon.type} == \text{interactive}) \\ \|\; (\text{event\_id} == \text{logoff}) \;\|\; (\text{event\_id} == \text{account\_creation})\end{aligned} \tag{4.1}$$

$$\text{event\_id} == \text{logon} \;\&\&\; \text{logon.type} == \text{interactive} \tag{4.2}$$

$$\text{event\_id} == \text{logoff} \tag{4.3}$$

$$\text{event\_id} == \text{account\_creation} \tag{4.4}$$

Given the monitoring signature 4.1, the console agent will decompose it to primitive monitoring task signatures 4.2, 4.3, and 4.4.

Each host is assigned those primitive tasks to monitor by the console agent. Upon detection of any primitive task, the host forwards the corresponding event details to the upper-level agent. Since we are interested in account details and last activity time, the lower-level agent forwards only the account details from the detected events to the console agent. The console agent saves each account's details and recent event occurrence time. The console agent counts the total number of dormant accounts from the recent event occurrence time for each user account, current time, and dormant threshold.

To determine the timely deactivation of dormant accounts, we have to monitor account disable (4725) and account deletion event (4726). Thus, the monitoring signature for the account deactivation/deletion event is:

$$\text{event\_id} == \text{account\_deletion} \;\|\; \text{event\_id} == \text{account\_disabled} \tag{4.5}$$

$$\text{event\_id} == \text{account\_deletion} \tag{4.6}$$

$$\text{event\_id} == \text{account\_disabled} \tag{4.7}$$

Similar to the previous monitoring signature for dormant account detection, the CA will decompose the monitoring task 4.5 to primitive monitoring tasks 4.6 and 4.7. The CA assigns each of those primitive monitoring tasks to a lower-level agent to monitor corresponding events. Upon detection of the corresponding primitive monitoring task, the EFA (event filtering agent) will forward the user account details and corresponding activity occurrence time to the console agent and save it in its persistent memory. Form each accounts deletion or deactivation time, CA counts the number of timely dormant account deletion/deactivation.

In the end, the CA of CSCMonitor reports dormant account statistics of measures 1, 2, and 3 for Table 4.2. Using those statistics about the measure, CA also reports the score for metrics M11 and M22 from Table 4.2.

One of the three VMs used for this demonstration contains the CA agent running, and each VM has an EFA agent always running. Those EFA agents forward the detected events to the CA.

4.6 Conclusion and Discussion

In summary, this chapter describes prompt engineering to generate measures and metrics that will be used to validate CIS critical security control enforcement. We generate CSC ontology, measures, and metrics using a few-shot prompting with CoT to validate CSC safeguard enforcement. Moreover, we elicit measure and metric implementation steps from the LLM by using a chain of zero-shot prompting where each prompt uses knowledge from previous prompts as context (generated knowledge prompting) for the current prompt.

We evaluate the LLM-generated measures and metrics using LLM. To evaluate LLM-generated output, we provide an evaluation prompt that evaluates each generated measure and metric under three criteria: semantic similarity, novelty, and correctness. We calculate the correlation scores using Pearson, Spearman, and Kendall Tau's functions to ensure the LLM evaluation is reliable and aligned with human evaluation. The correlation coefficient indicates that LLM evaluation aligns with the human evaluation of generated measures and metrics.

Though we did not get any wrong answers during our experiment when the appropriate prompt was given, the hallucination of LLM is a known problem [70], and we can not predict when it will happen. The answer generated by LLM can be wrong from time to time. Since humans do the implementation steps of measures based on the answers from LLM, this hallucination problem will be detected during the implementation steps if it happens. Another limitation of this work is that LLM is sensitive to prompts. A Prompt works for a specific LLM version, but it does not mean it will work for all future versions or a different LLM. For different LLM or different LLM versions, we may have to check whether the existing prompt works or not and may have to fine-tune it to align it with the specific LLM.

Table 4.1: Human and LLM-generated Measures and Metrics for Safeguard 5.1

Safeguard	CIS CSC Specification (Human-Generated) Metric	LLM-Generated Metric
5.1	Completeness of Inventory = The percentage of minimum elements included in the inventory = (Count of elements provided in inventory) / 4	Account Inventory Accuracy = the accuracy of the account inventory in representing all accounts in the enterprise = Number of Correct Entries / Total Entries) * 100
	Completeness of Inventory Details = The percentage of accounts with complete information = (Count of accounts in inventory with complete information) / 2	Authorization Validation Frequency = how frequently the recurring validation of accounts is performed = (Number of Validations within Time Period / Total Time Periods) * 100
	Accuracy of Inventory = The percentage of accurately listed accounts in the inventory = Count of unauthorized accounts / Count of identified current accounts	Recurring Validation Frequency = how frequently the recurring validation of accounts is performed = (Number of Validations within Time Period / Total Time Periods) * 100
		User and Administrator Account Ratio = the ratio of user accounts to administrator accounts in the inventory = Number of User Accounts / Number of Administrator Accounts
		Completeness of Account Details = the completeness of account details recorded in the inventory = (Number of Complete Entries / Total Entries) * 100

Table 4.2: Measures and Metrics for CSC 5.3 generated by LLM

Name	Measures	Metrics
Dormant Account Deactivation Compliance, the percentage of deactivated dormant accounts	1. M1 = Total dormant account. 2. M2 = Number of deactivated dormant accounts	M11 = M2 / M1
Timeliness of Deactivation, the percentage of timely deactivation	1. M3 = Number of timely (45 days) deactivated dormant accounts	M22 = M3 / M2
Overall Compliance Score		(M11 + M22) / 2

CHAPTER 5: Conclusions

The centralized nature of existing monitoring system, corresponding communication overhead and resource consumption call for the design of a distributed hierarchical monitoring architecture that reduces the communication overhead among the agents and resource consumption. Moreover, the monitoring system should provide sufficient expressibility in terms of analytical language to express user subscription requests. In this dissertation, we provide an SCAHunter: a distributed hierarchical agent architecture that can monitor end-host and networking devices based on user subscription requests to hunt cyber threats and assess the critical security control enforcement quality. The proposed event monitoring system will reduce the communication overhead and resource consumption in event monitoring, cyber threat hunting, and CSC validation and effectiveness measurement through the use of distributed hierarchical monitoring.

Firstly, we propose a distributed hierarchical agent infrastructure for event monitoring that optimizes monitoring tasks to reduce resource usage and communication overhead. We also provide an analytical language to facilitate the required expressibility of the user/threat hunter's subscription request. The agent infrastructure uses the analytical language to specify user task requests.

Secondly, to serve the user request, the monitoring system must decompose the user subscription request specified with the provided analytical language and determine the optimal number of CEDAs that cover all required EFAs so that event correlation tasks are distributed among the hosts. Since CEDA generation is an NP-hard problem, we provide an approximation algorithms to generate an optimal agent hierarchy. We also develop ETW-based agents to monitor signature-specific events so that on-demand

monitoring is supported.

Thirdly, we evaluated our proposed architecture using log data generated by running three test scripts provided by Red Canary Atomic Red Team. We created attack signatures for the test scripts following the MITRE ATT&CK technique description during the evaluation. We also evaluated our proposed approach using the DARPA OpTC attack dataset. To compare our approach with the existing centralized event monitoring approaches for threat hunting, we also implemented centralized event monitoring using Splunk.

Fourthly, we provided prompt engineering techniques to generate measures and metrics, which will reduce manual work on metrics development and dependency on the security analyst's expertise and prior knowledge. To generate CSC ontology, we used CoT prompting. To generate measures and metrics, we used a few shot promoting with CoT prompting. In the end, we generated measures and metrics implementation steps using generated knowledge prompting.

We further assessed the enforcement quality of security controls by implementing a CSC assessment monitor and using implementation steps generated during generated knowledge prompting and monitoring the system using our distributed hierarchical monitoring agent architecture.

In this dissertation, the issue of a single point of failure within our distributed system was not addressed, a challenge that can potentially be remedied through the use of consensus algorithms, presenting a further avenue for research. The hallucination of LLM is a known problem, and the answers from LLM during automated measures and metrics generation may not always be accurate over time. Therefore, it may be necessary for security analysts to verify the reliability of the measures and metrics produced during their implementation. An additional limitation identified of this dissertation is the sensitivity of LLMs to the specificity of prompts. The efficacy of existing prompts may vary across different LLMs or their versions, necessitating

adjustments or fine-tuning to ensure compatibility with the specific LLM.

REFERENCES

[1] "Symmantec. attack listing." https://www.symantec.com/security-center/a-z.

[2] L. D. Ping Chen and C. Huygens, "A study on advanced persistent threats," in *In IFIP International Conference on Communications and Multimedia Security*, 2014.

[3] "The state of ransomware 2022." https://www.sophos.com/en-us/content/state-of-ransomware. [Online; accessed 15 July, 2022].

[4] C. G. Z. L. C. P. S. Stevens Le Blond, Adina Uritesc and E. Kirda, "A look at targeted attacks through the lense of an ngo," in *In USENIX Security Symposium*, 2014.

[5] C. K. Marco Cova and G. Vigna, "Detection and analysis of drive-by-download attacks and malicious javascript code," in *In International Conference on World Wide Web (WWW)*, 2010.

[6] P. P. H. D. H. Y. S. L. B. D. M. Brown Farinholt, Mohammad Rezaeirad and K. Levchenko, "To catch a ratter: Monitoring the behavior of amateur darkcomet rat operators in the wild," in *In IEEE Symposium on Security and Privacy*, 2017.

[7] L. C. B. G. M. S. R. K. C. K. Brett Stone-Gross, Marco Cova and G. Vigna, "Your botnet is my botnet: analysis of a botnet takeover," in *In ACM Conference on Computer and Communications Security(CCS)*, 2009.

[8] M. Ahmed, J. Wei, Y. Wang, and E. Al-Shaer, "A poisoning attack against cryptocurrency mining pools," in *Data Privacy Management, Cryptocurrencies and Blockchain Technology* (J. Garcia-Alfaro, J. Herrera-Joancomartí, G. Livraga, and R. Rios, eds.), (Cham), pp. 140–154, Springer International Publishing, 2018.

[9] "Special report m-trends 2021." https://www.mandiant.com/resources/m-trends-2021. [Online; accessed 15 July, 2022].

[10] "How much does a data breach cost?." https://www.ibm.com/security/data-breach. [Online; accessed 15 July, 2022].

[11] Y. Li, J. Xia, S. Zhang, J. Yan, X. Ai, and K. Dai, "An efficient intrusion detection system based on support vector machines and gradually feature removal method," in *Expert Syst. Appl*, 2009.

[12] C. G. H. R. S. A. Peng Ning, Dingbang Xu, "Building attack scenarios through integration of complementary alert correlation methods," in *Expert Syst. Appl*, 2004.

[13] H. A.-W. E. Al-Shaer and K. Maly, "Hifi: a new monitoring architecture for distributed systems management," in *19th IEEE International Conference on Distributed Computing Systems (Cat. No.99CB37003)*, pp. pp. 171–178, 1999.

[14] M. W. B. J. B. J. G. H. A. G. Kevin Patrick, Mahaffey Timothy and A. C. Abey, "Distributed monitoring, evaluation, and response for multiple devices," in *United States Patent, US9753796B2*, September 5 2017.

[15] L. D. C. L. M. J. M. B. R. S. M. S. Y. V. Akshay Adhikari, Scott Vincent Bianco, "Distributed monitoring and analysis system for network traffic," in *United States Patent, US7031264B2*, April 18 2006.

[16] D. H. H. Liu and Y. Zhang, "Cooperative control based on distributed attack identification and isolation," in *2020 16th International Conference on Control, Automation, Robotics and Vision (ICARCV)*, pp. pp. 1268–1273, 2020.

[17] C. K. T. P. F. Boem, R. M. G. Ferrari and M. M. Polycarpou, "A distributed networked approach for fault detection of large-scale systems," in *IEEE Transactions on Automatic Control, vol. 62, no. 1*, pp. pp. 18–33, January 2017.

[18] "Cis controls self assessment tool (cis csat)." https://www.cisecurity.org/controls/cis-controls-self-assessment-tool-cis-csat, 5 January 2024.

[19] "Cis controls measurement companion guide." https://www.cisecurity.org/insights/white-papers/a-measurement-companion-to-the-cis-critical-controls, 5 January 2024.

[20] "Rapid7 global service." https://www.rapid7.com/solutions/compliance/critical-controls/. [Online; accessed 15 march, 2022].

[21] R. Bar-Haim, L. Eden, Y. Kantor, V. Agarwal, M. Devereux, N. Gupta, A. Kumar, M. Orbach, and M. Zan, "Towards automated assessment of organizational cybersecurity posture in cloud," in *Proceedings of the 6th Joint International Conference on Data Science & Management of Data (10th ACM IKDD CODS and 28th COMAD)*, CODS-COMAD 23, (New York, NY, USA), pp. 167–175, Association for Computing Machinery, 2023.

[22] S. Gros, "A critical view on cis controls," *2021 16th International Conference on Telecommunications (ConTEL)*, pp. 122–128, 2019.

[23] S. M. Milajerdi, B. Eshete, R. Gjomemo, and V. Venkatakrishnan, "Poirot: Aligning attack behavior with kernel audit records for cyber threat hunting," in *Proceedings of the 2019 ACM SIGSAC Conference on Computer and Communications Security*, pp. 1795–1812, 2019.

[24] B. Bhattarai and H. Huang, "Steinerlog: Prize collecting the audit logs for threat hunting on enterprise network," in *Proceedings of the 2022 ACM on Asia Conference on Computer and Communications Security*, ASIA CCS '22, (New York, NY, USA), p. 97â108, Association for Computing Machinery, 2022.

[25] W. U. Hassan, D. Li, K. Jee, X. Yu, K. Zou, D. Wang, Z. Chen, Z. Li, J. Rhee, J. Gui, and A. Bates, "This is why we can't cache nice things: Lightning-fast threat hunting using suspicion-based hierarchical storage," in *Annual Computer Security Applications Conference*, ACSAC '20, (New York, NY, USA), p. 165â178, Association for Computing Machinery, 2020.

[26] "Adversarial tactics, techniques & common knowledge." https://attack.mitre.org/. [Online; accessed 15 march, 2022].

[27] W. U. Hassan, S. Guo, D. Li, Z. Chen, K. Jee, Z. Li, and A. Bates, "Nodoze: Combatting threat alert fatigue with automated provenance triage," in *Network and Distributed Systems Security Symposium*, 2019.

[28] W. U. Hassan, A. Bates, and D. Marino, "Tactical provenance analysis for endpoint detection and response systems," in *2020 IEEE Symposium on Security and Privacy (SP)*, pp. 1172–1189, 2020.

[29] "Atomic red team: Mitre attack technique detector." https://github.com/redcanaryco/atomic-red-team/. [Online; accessed 15 march, 2022].

[30] R. Arantes, C. Weir, H. Hannon, and M. Kulseng, "Operationally transparent cyber (optc)," 2021.

[31] "Etw: Evetn tracing for windows." https://learn.microsoft.com/en-us/message-analyzer/etw-framework-conceptual-tutorial. [Online; accessed 15 March, 2024].

[32] "Center for internet security- critical security control, 2021." https://www.cisecurity.org/controls/cis-controls-list, 5 January 2024.

[33] "Alienvault ossim." https://www.alienvault.com/products/ossim. [Online; accessed 15 march, 2022].

[34] J. Wei, X. Wang, D. Schuurmans, M. Bosma, brian ichter, F. Xia, E. H. Chi, Q. V. Le, and D. Zhou, "Chain of thought prompting elicits reasoning in large language models," in *Advances in Neural Information Processing Systems* (A. H. Oh, A. Agarwal, D. Belgrave, and K. Cho, eds.), 2022.

[35] S. Mishra, D. Khashabi, C. Baral, Y. Choi, and H. Hajishirzi, "Reframing instructional prompts to gptk's language," *ArXiv*, vol. abs/2109.07830, 2021.

[36] Z. Zhao, E. Wallace, S. Feng, D. Klein, and S. Singh, "Calibrate before use: Improving few-shot performance of language models," in *International Conference on Machine Learning*, pp. 12697–12706, PMLR, 2021.

[37] T. Brown, B. Mann, N. Ryder, M. Subbiah, J. D. Kaplan, P. Dhariwal, A. Neelakantan, P. Shyam, G. Sastry, A. Askell, S. Agarwal, A. Herbert-Voss, G. Krueger, T. Henighan, R. Child, A. Ramesh, D. Ziegler, J. Wu, C. Winter, C. Hesse, M. Chen, E. Sigler, M. Litwin, S. Gray, B. Chess, J. Clark, C. Berner, S. McCandlish, A. Radford, I. Sutskever, and D. Amodei, "Language models are few-shot learners," in *Advances in Neural Information Processing Systems* (H. Larochelle, M. Ranzato, R. Hadsell, M. Balcan, and H. Lin, eds.), vol. 33, pp. 1877–1901, Curran Associates, Inc., 2020.

[38] J. Liu, A. Liu, X. Lu, S. Welleck, P. West, R. L. Bras, Y. Choi, and H. Hajishirzi, "Generated knowledge prompting for commonsense reasoning," 2022.

[39] M. Ahmed, J. Wei, and E. Al-Shaer, "Scahunter: Scalable threat hunting through decentralized hierarchical monitoring agent architecture," in *Intelligent Computing* (K. Arai, ed.), (Cham), pp. 1282–1307, Springer Nature Switzerland, 2023.

[40] S. M. Milajerdi, R. Gjomemo, B. Eshete, R. Sekar, and V. Venkatakrishnan, "Holmes: real-time apt detection through correlation of suspicious information flows," in *2019 IEEE Symposium on Security and Privacy (SP)*, pp. 1137–1152, IEEE, 2019.

[41] M. N. Hossain, S. M. Milajerdi, J. Wang, B. Eshete, R. Gjomemo, R. Sekar, S. Stoller, and V. Venkatakrishnan, "{SLEUTH}: Real-time attack scenario reconstruction from {COTS} audit data," in *26th {USENIX} Security Symposium ({USENIX} Security 17)*, pp. 487–504, 2017.

[42] "Logrythm:threat hunting use cases." https://logrhythm.com/use-cases/. [Online; accessed 15 April, 2022].

[43] "Endpoint detection and response solution survey." https://www.gartner.com/reviews/market/endpoint-detection-and-response-solutions. [Online; accessed 3 August, 2022].

[44] R. Al-Shaer, J. M. Spring, and E. Christou, "Learning the associations of mitre att&ck adversarial techniques," in *2020 IEEE Conference on Communications and Network Security (CNS)*, pp. 1–9, 2020.

[45] R. Wei, L. Cai, L. Zhao, A. Yu, and D. Meng, "Deephunter: A graph neural network based approach for robust cyber threat hunting," in *Security and Privacy in Communication Networks* (J. Garcia-Alfaro, S. Li, R. Poovendran, H. Debar, and M. Yung, eds.), (Cham), pp. 3–24, Springer International Publishing, 2021.

[46] W. U. Hassan, M. A. Noureddine, P. Datta, and A. Bates, "Omegalog: High-fidelity attack investigation via transparent multi-layer log analysis," in *Network and Distributed System Security Symposium*, 2020.

[47] C. Xiong, T. Zhu, W. Dong, L. Ruan, R. Yang, Y. Chen, Y. Cheng, S. Cheng, and X. Chen, "Conan: A practical real-time apt detection system with high accuracy and efficiency," *IEEE Transactions on Dependable and Secure Computing*, 2020.

[48] V. Yegneswaran, P. Barford, and S. Jha, "Global intrusion detection in the domino overlay system," tech. rep., University of Wisconsin-Madison Department of Computer Sciences, 2003.

[49] K. Benahmed, M. Merabti, and H. Haffaf, "Distributed monitoring for misbehaviour detection in wireless sensor networks," *Security and Communication Networks*, vol. 6, no. 4, pp. 388–400, 2013.

[50] Y.-S. Wu, B. Foo, Y. Mei, and S. Bagchi, "Collaborative intrusion detection system (cids): a framework for accurate and efficient ids," in *19th Annual Computer Security Applications Conference, 2003. Proceedings.*, pp. 234–244, IEEE, 2003.

[51] E. Al-Shaer, H. Abdel-Wahab, and K. Maly, "Hifi: a new monitoring architecture for distributed systems management," in *Proceedings. 19th IEEE International Conference on Distributed Computing Systems (Cat. No.99CB37003)*, pp. 171–178, 1999.

[52] M. Ahmed and E. Al-Shaer, "Measures and metrics for the enforcement of critical security controls: a case study of boundary defense," in *Proceedings of the 6th Annual Symposium on Hot Topics in the Science of Security*, pp. 1–3, 2019.

[53] P. T. N. Hong and S. Le Van, "An online monitoring solution for complex distributed systems based on hierarchical monitoring agents," in *Knowledge and Systems Engineering*, pp. 187–198, Springer, 2014.

[54] M. M. Alam and W. Wang, "A comprehensive survey on data provenance: State-of-the-art approaches and their deployments for iot security enforcement," *Journal of Computer Security*, no. Preprint, pp. 1–24, 2021.

[55] M. Andreolini, M. Colajanni, and M. Pietri, "A scalable architecture for real-time monitoring of large information systems," in *2012 Second Symposium on Network Cloud Computing and Applications*, pp. 143–150, IEEE, 2012.

[56] M. N. Alsaleh, J. Wei, E. Al-Shaer, and M. Ahmed, "Gextractor: Towards automated extraction of malware deception parameters," in *Proceedings of the 8th Software Security, Protection, and Reverse Engineering Workshop*, SSPREW-8, (New York, NY, USA), Association for Computing Machinery, 2018.

[57] F. D. Sacerdoti, M. J. Katz, M. L. Massie, and D. E. Culler, "Wide area cluster monitoring with ganglia.," in *CLUSTER*, vol. 3, pp. 289–289, 2003.

[58] F. Boem, R. M. Ferrari, C. Keliris, T. Parisini, and M. M. Polycarpou, "A distributed networked approach for fault detection of large-scale systems," *IEEE Transactions on Automatic Control*, vol. 62, no. 1, pp. 18–33, 2016.

[59] A. Wood, *Rabbit MQ: For Starters*. North Charleston, SC, USA: CreateSpace Independent Publishing Platform, 2016.

[60] L. Axon, A. Erola, A. Janse van Rensburg, J. R. Nurse, M. Goldsmith, and S. Creese, "Practitioners' views on cybersecurity control adoption and effectiveness," in *Proceedings of the 16th International Conference on Availability, Reliability and Security*, pp. 1–10, 2021.

[61] "Cis benchmark." https://www.cisecurity.org/cis-benchmarks, 5 January 2024.

[62] "Cis controls assessment specification." https://controls-assessment-specification.readthedocs.io/en/stable/index.html, 5 January 2024.

[63] W. K. Sedano and M. Salman, "Auditing linux operating system with center for internet security (cis) standard," in *2021 International Conference on Information Technology (ICIT)*, pp. 466–471, IEEE, 2021.

[64] A. Dutta and E. Al-Shaer, ""what","where", and "why" cybersecurity controls to enforce for optimal risk mitigation," in *2019 IEEE Conference on Communications and Network Security (CNS)*, pp. 160–168, IEEE, 2019.

[65] Y.-T. Lin and Y.-N. Chen, "Llm-eval: Unified multi-dimensional automatic evaluation for open-domain conversations with large language models," 2023.

[66] Z. Zeng, J. Yu, T. Gao, Y. Meng, T. Goyal, and D. Chen, "Evaluating large language models at evaluating instruction following," *ArXiv*, vol. abs/2310.07641, 2023.

[67] W. Stern, S. J. Goh, N. Nur, P. J. Aragon, and T. Mercer, "Natural language explanations for suicide risk classification using large language models," 2024.

[68] S. Yao, D. Yu, J. Zhao, I. Shafran, T. L. Griffiths, Y. Cao, and K. Narasimhan, "Tree of thoughts: Deliberate problem solving with large language models," *CoRR*, vol. abs/2305.10601, 2023.

[69] "Llm chat trnascript." https://github.com/MohiuddinSohel/CSC-Assessment-Prompting, 12 April 2024.

[70] Y. Zhang, Y. Li, L. Cui, D. Cai, L. Liu, T. Fu, X. Huang, E. Zhao, Y. Zhang, Y. Chen, *et al.*, "Siren's song in the ai ocean: A survey on hallucination in large language models," *arXiv preprint arXiv:2309.01219*, 2023.

APPENDIX A:

A.1 CSC Safeguard

Table A.1: CSC safeguard from version 8 and sub-control from version 7

CSC	Description
1.1	Establish and maintain an accurate, detailed, and up-to-date inventory of all enterprise assets with potential to store or process data, including end-user devices (portable and mobile), network devices, non-computing devices, and servers. Ensure inventory records network address (if static), hardware address, machine name, data asset owner, department for each asset, and whether the asset has been approved to connect to network. For mobile end-user devices, MDM-type tools can support this process where appropriate. This inventory includes assets connected to infrastructure physically, virtually, remotely, and within cloud environments. Additionally, it includes assets that are regularly connected to enterprise's network infrastructure, even if they are not under control of the enterprise. Review and update the inventory of all enterprise assets bi-annually, or more frequently.
1.5	Use a passive discovery tool to identify assets connected to the enterprise's network. Review and use scans to update the enterprise's asset inventory at least weekly, or more frequently.
5.1	Establish and maintain an inventory of all accounts managed in enterprise. The inventory must include both user and administrator accounts. The inventory, at minimum, should contain person's name, username, start/stop dates, and department. Validate that all active accounts are authorized, on a recurring schedule at a minimum quarterly, or more frequently.
5.2	Use unique passwords for all enterprise assets. Best practice implementation includes, at a minimum, an 8-character password for accounts using MFA and a 14-character password for accounts not using MFA.
5.3	Delete or disable any dormant accounts after a period of 45 days of inactivity, where supported.
5.4	Restrict administrator privileges to dedicated administrator accounts on enterprise assets. Conduct general computing activities, such as internet browsing, email, and productivity suite use, from the user's primary, non-privileged account.
5.5	Establish and maintain an inventory of service accounts. At minimum, inventory must contain department owner, review date, and purpose. Perform service account reviews to validate that all active accounts are authorized, on a recurring schedule at a minimum quarterly, or more frequently.
5.6	Centralize account management through a directory or identity service.
12.1	Maintain an up-to-date inventory of all organization's network boundaries.
12.2	Establish and maintain secure network architecture. A secure network must address segmentation, least privilege, availability, at minimum.