

See discussions, stats, and author profiles for this publication at: <https://www.researchgate.net/publication/381724183>


# Prompting LLM to Enforce and Validate CIS Critical Security Control

Conference Paper · June 2024  
DOI: 10.1145/3649158.3657036

CITATION  
1

READS  
312


3 authors:



**Mohiuddin Ahmed**  
University of North Carolina at Charlotte

7 PUBLICATIONS 235 CITATIONS


SEE PROFILE



**Jinpeng Wei**  
Florida International University

52 PUBLICATIONS 757 CITATIONS

SEE PROFILE



**Ehab Al-Shaer**  
Carnegie Mellon University

260 PUBLICATIONS 7,306 CITATIONS

SEE PROFILE



# Prompting LLM to Enforce and Validate CIS Critical Security Control

Mohiuddin Ahmed  
mahmed27@charlotte.edu  
University of North Carolina at  
Charlotte  
Software and Information Systems  
Charlotte, NC, USA

Jinpeng Wei  
jwei8@charlotte.edu  
University of North Carolina at  
Charlotte  
Software and Information Systems  
Charlotte, NC, USA

Ehab Al-Shaer  
ehab@cmu.edu  
Carnegie Mellon University  
School of Computer Science  
Pittsburgh, PA, USA

## ABSTRACT

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 (CoT) prompting to generate measures and metrics and generated knowledge prompting for metrics implementation. Our evaluation shows that prompt engineering to extract measures, 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.

## CCS CONCEPTS

• Security and privacy → Security requirements; Access control.

## KEYWORDS

Critical Security Control; LLM; Prompt Engineering; Account Management.

### ACM Reference Format:

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, 12 pages. <https://doi.org/10.1145/3649158.3657036>

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. Copyrights for components of this work owned by others than the author(s) must be honored. Abstracting with credit is permitted. To copy otherwise, or republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee. Request permissions from [permissions@acm.org](mailto:permissions@acm.org).

SACMAT 2024, May 15–17, 2024, San Antonio, TX, USA

© 2024 Copyright held by the owner/author(s). Publication rights licensed to ACM.

ACM ISBN 979-8-4007-0491-8/24/05

<https://doi.org/10.1145/3649158.3657036>

## 1 INTRODUCTION

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 [2]. 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 [5, 6]. Some third-party vendors offer tools and services to help organizations implement and enforce the controls [1, 11]. 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 [14].

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 [10]. 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 [3, 5], 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 [8]. 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 [4]. 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 log data sources, 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 [12]. Therefore, we leverage LLM to extract measures, metrics, and monitoring implementation mechanisms in this paper. 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 to develop a chain-of-thought (CoT) prompt to generate measures and metrics and 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 effort on manual evaluation where a different LLM is used for evaluation. With the help of Spearman, Pearson, and Kendall's Tau 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 paper is organized as follows: Section 2 surveys existing research work on CIS CSCs and prompt engineering with LLMs to extract information and reasoning; Section 3 describes proposed CSC enforcement validation approach; Section 4 provides a case study of the whole CSC enforcement validation approach; Section 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 6 summarizes our contributions and limitations. In this paper, all prompts are developed and executed in ChatGPT, and we will use LLM and ChatGPT interchangeably for the rest of the paper.

## 2 RELATED WORKS

**Critical Security Control.** In [11], 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 [10], 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 [18], 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.* [13] 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.* [8] 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.* [17] 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.* [12] show 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.* [20] show 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 [24]. The authors at [15, 22] used the LLM as an evaluator to evaluate text generated by another LLM. Stern *et al.* [19] 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.

## 3 OVERVIEW OF THE CSC VALIDATION

This section outlines our CSC validation approach and its goals, as illustrated in Figure 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.

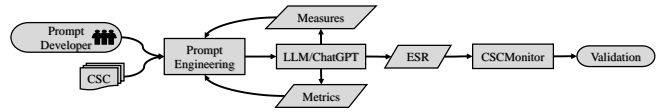


Figure 1: CSC validation approach

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 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 1 and 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.

### 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 actor’s 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.

### 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 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, 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 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.

### 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 KMIs 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) and generated knowledge 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 [20, 21], CoT elicits reasoning as an emergent property of LLM. According to [20], 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.

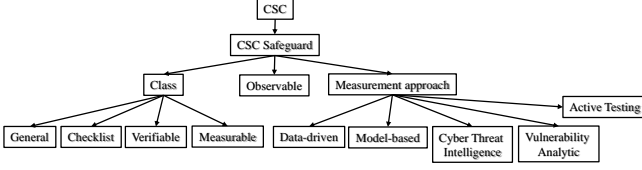


Figure 2: CSC Ontology

**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 [20]. We develop a prompt template (Section 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) 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) 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 [12]. 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 [4] 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 which 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 [16] in association with CoT to elicit metric implementation approaches from the LLM by prompting.



Figure 3: CoT prompting flow

To elicit CSC measure implementation using the LLM as shown in Figure 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.

### 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 distributed hierarchical event monitoring agent architecture to overcome those challenges as proposed in [9]. 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 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 [12].

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 3.2. The input prompt and output from the LLM are given in Figure 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 ontology. The LLM could not distinguish between measures and metrics in this zero-shot prompting. Some of the measure outputs in Figure 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 CSC ontology, and measure and metric. 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 [24]. For example, to extract CSC ontology by following the CoT prompting in Figure 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.

**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 measurement or validation approach 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 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 [11] in addition to other novel measures and metrics as shown in Tables 1 and 2.

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 [16].

To demonstrate measure implementation using generated knowledge prompting as shown in Figure 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 2) the number of unrecognized network boundaries in the network. To monitor the two measures mentioned earlier, the security analyst needs to know which cyber observables 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 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.

#### 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.

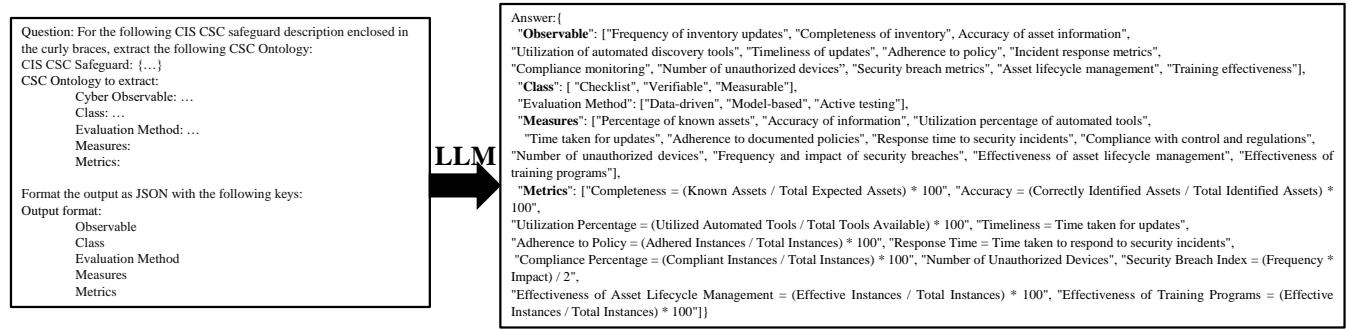


Figure 4: Zero-shot prompting for CSC Ontology

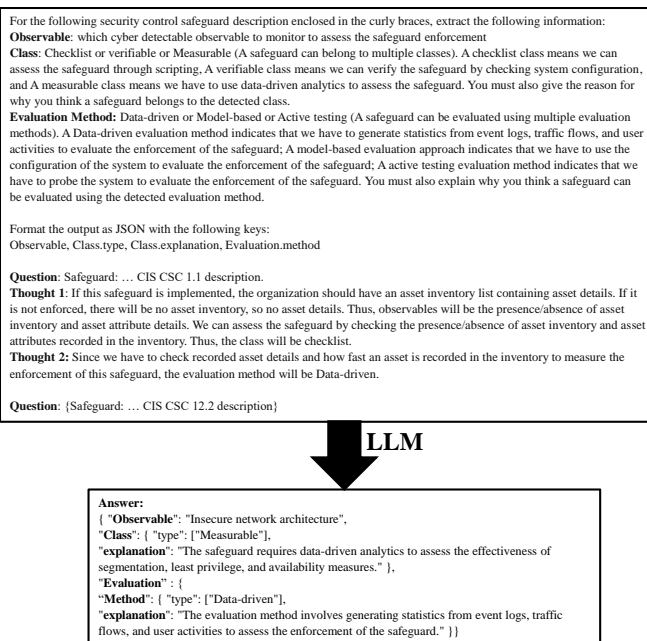


Figure 5: CoT prompting for CSC Ontology

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 a reward model that scores model (LLM) responses during training, this shows a new direction to use LLMs themselves as evaluators [15, 22].

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 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 [22].

We used the CIS control-assessment-specification [4] GitHub repository to generate the golden dataset as the ground truth of LLM-generated measures and metrics as shown in Figure 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 [15, 22], 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



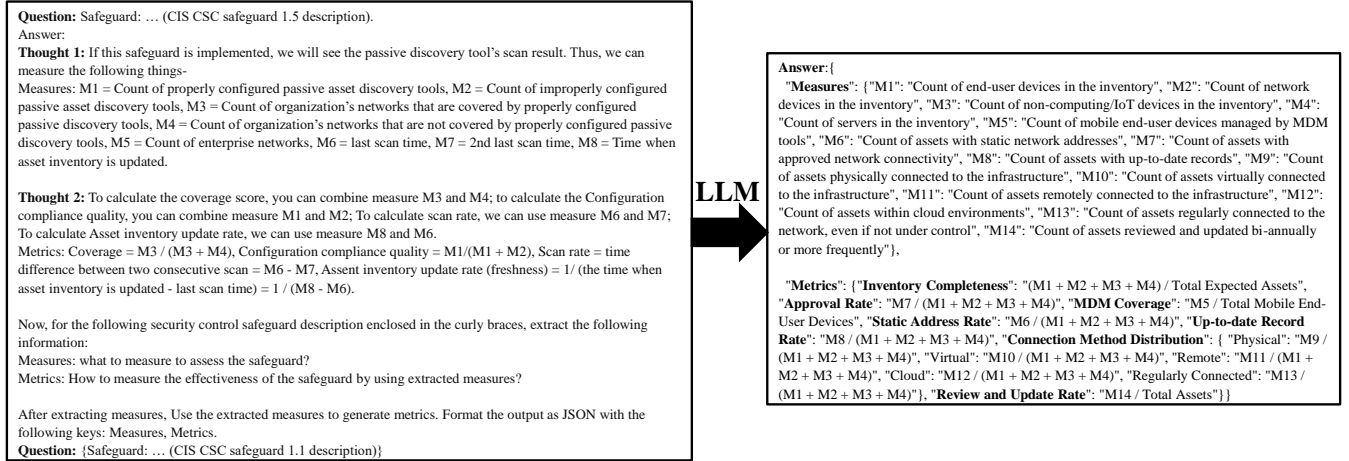


Figure 6: CoT prompting for Measures and Metrics

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

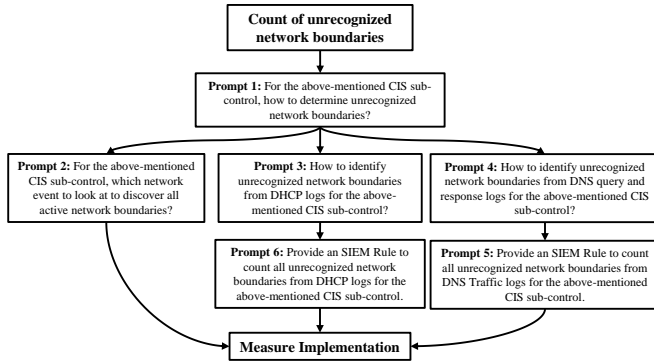
Table 1: Human and LLM-generated Measures and Metrics for Safeguard 5.1

Figure 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's Tau correlation coefficients using human and LLM evaluation scores.

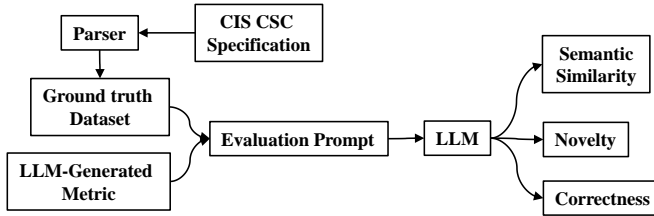
**Semantic Similarity:** From the result in Figure 10 (a), we found that LLM-generated metrics differ from the human-labeled ones

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).

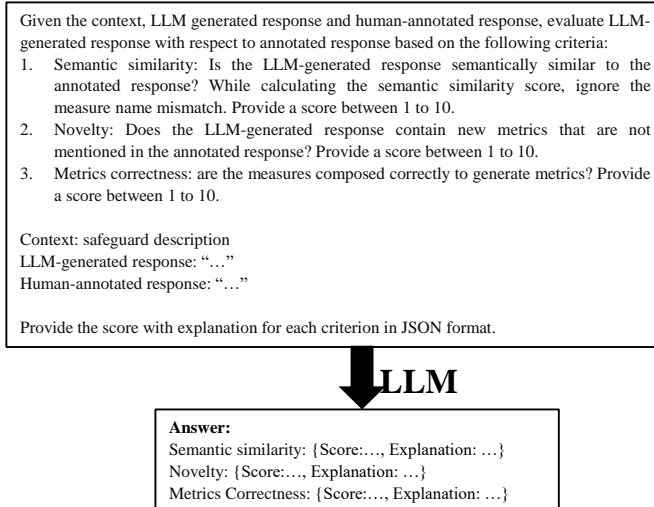
**Novelty:** LLM-generated metrics provide new metrics for all of the safeguards we evaluated (Figure 10 (b)). Since the human-generated metric is incomplete and provided only as a guideline



**Figure 7: Generated knowledge prompting for Metric Implementation**



**Figure 8: Evaluation of generated Metrics and Measures with LLM**



**Figure 9: Prompting to evaluate Measures and Metrics**

for security analysts, the LLM always generates new metrics based on the safeguard description.

**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 9. We also manually evaluated the LLM-generated measures and metrics. The goal here is to check whether LLM can compose different specific measures

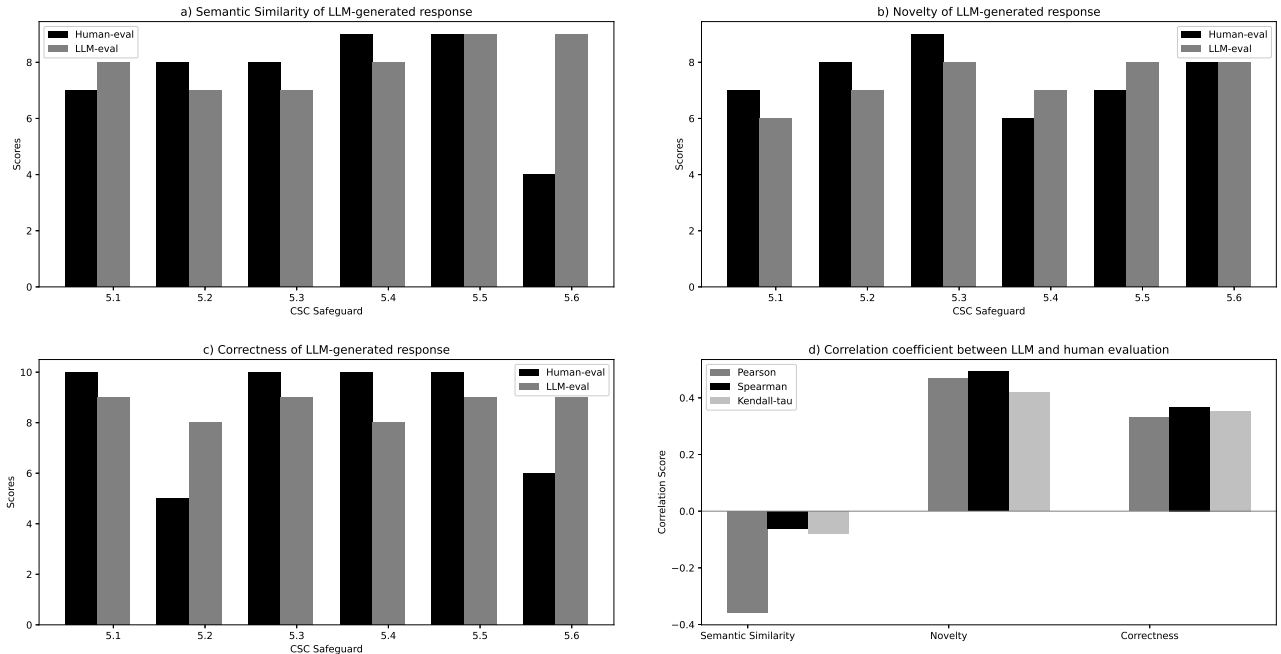
together to generate metrics. From Figure 10 (c), we see that the correctness score of safeguards 5.1 and 5.3 to 5.6 for both manual and LLM are high (more than 5). Initially, the correctness score for LLM-generated metrics for safeguard 5.2 is less than 5. We investigated it and found out that the metrics 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 9) for the same safeguard. We did this experiment for 10 safeguards. We calculate the Pearson, Spearman, and Kendall’s 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’s Tau) approach.

From Figure 10 (d), we have Pearson, Spearman, and Kendall’s Tau correlation coefficients 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 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’s Tau correlation coefficient. These positive correlation coefficients indicate that using LLM as an evaluator is reliable and aligns with human evaluation.

From Figure 10 (d), the correlation coefficients of 0.47, 0.49, and 0.42 for Pearson, Spearman, and Kendall’s Tau indicate 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’s 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



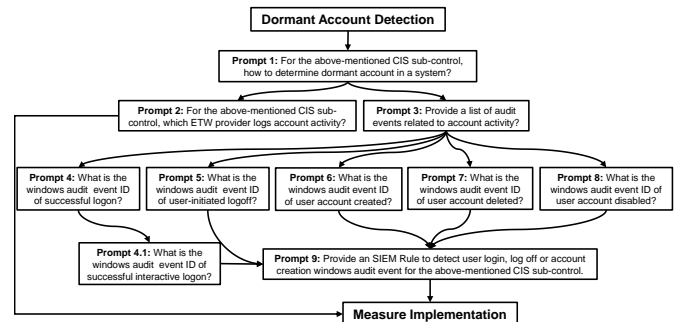
**Figure 10: Semantic Similarity, Novelty, Correctness evaluation between LLM-generated and human-labeled metrics, and Correlation between human evaluation and LLM evaluation (all the evaluation done with ChatGPT-3.5)**

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 [7].

**General Applicability of our Prompts for Security Controls Provided by Entities other than CIS.** While this paper 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 [7]. 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 [7] with ChatGPT-3.5 and ChatGPT-4 show the general applicability of our prompt for security controls provided by organizations other than CIS.

## 5.1 Metric Implementation Demonstration using LLM

In this section, we demonstrate measure and metric implementation with the help of chain-of-thought and generated knowledge 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.



**Figure 11: Generated knowledge prompting for dormant account detection implementation**

In the first step, we generate measures and metrics for safeguard 5.3 using the prompt in Figure 6. The LLM-generated metrics

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		$(M11 + M22) / 2$

**Table 2: Measures and Metrics for CSC 5.3 generated by LLM**

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 three measures: 1) Total dormant accounts, 2) Number of deactivated dormant accounts, and 3) 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 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 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:

$$\begin{aligned} &(\text{event\_id} == \text{logon} \ \&\& \ \text{logon.type} == \text{interactive}) \\ &|| (\text{event\_id} == \text{logoff}) \ || (\text{event\_id} == \text{account\_creation}) \end{aligned} \quad (1)$$

$$\text{event\_id} == \text{logon} \ \&\& \ \text{logon.type} == \text{interactive} \quad (2)$$

$$\text{event\_id} == \text{logoff} \quad (3)$$

$$\text{event\_id} == \text{account\_creation} \quad (4)$$

Given the monitoring signature: equation 1, the console agent will decompose it to primitive monitoring task signatures: equation 2, 3, and 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:

$$\text{event\_id} == \text{account\_deletion} \ || \ \text{event\_id} == \text{account\_disabled} \quad (5)$$

$$\text{event\_id} == \text{account\_deletion} \quad (6)$$

$$\text{event\_id} == \text{account\_disabled} \quad (7)$$

Similar to the previous monitoring signature for dormant account detection, the CA will decompose the monitoring task: equation 5 to primitive monitoring tasks: equation 6 and 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. For each account 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 2. Using those statistics about the measure, CA also reports the score for metrics M11 and M22 from Table 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.

## 6 CONCLUSION AND DISCUSSION

In summary, this paper 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's Tau 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 [23], 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.

## REFERENCES

- [1] 2021. Rapid7 Global service. <https://www.rapid7.com/solutions/compliance/critical-controls/>.
- [2] 2024. Center for Internet Security- Critical Security Control, 2021. <https://www.cisecurity.org/controls/cis-controls-list>.
- [3] 2024. CIS Benchmark. <https://www.cisecurity.org/cis-benchmarks>.
- [4] 2024. CIS Controls Assessment Specification. <https://controls-assessment-specification.readthedocs.io/en/stable/index.html>.
- [5] 2024. CIS Controls Measurement Companion Guide. <https://www.cisecurity.org/insights/white-papers/a-measurement-companion-to-the-cis-critical-controls>.
- [6] 2024. CIS Controls Self Assessment Tool (CIS CSAT). <https://www.cisecurity.org/controls/cis-controls-self-assessment-tool-cis-csat>.
- [7] 2024. LLM chat trnascript. <https://github.com/MohiuddinSohel/CSC-Assessment-Prompting>.
- [8] Mohiuddin Ahmed and 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, 3 pages. <https://doi.org/10.1145/3314058.3317730>
- [9] Mohiuddin Ahmed, Jinpeng Wei, and Ehab Al-Shaer. 2023. SCAHunter: Scalable Threat Hunting Through Decentralized Hierarchical Monitoring Agent Architecture. In *Intelligent Computing*, Kohei Arai (Ed.). Springer Nature Switzerland, Cham, 1282–1307.
- [10] Louise Axon, Arnau Erola, Alastair Janse van Rensburg, Jason RC Nurse, Michael Goldsmith, and Sadie Creese. 2021. Practitioners' views on cybersecurity control adoption and effectiveness. In *Proceedings of the 16th International Conference on Availability, Reliability and Security*. 1–10.
- [11] Roy Bar-Haim, Lilach Eden, Yoav Kantor, Vikas Agarwal, Mark Devereux, Nisha Gupta, Arun Kumar, Matan Orbach, and Michael Zan. 2023. 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)* (Mumbai, India) (*CODS-COMAD 23*). Association for Computing Machinery, New York, NY, USA, 167–175. <https://doi.org/10.1145/3570991.3571008>
- [12] Tom Brown, Benjamin Mann, Nick Ryder, Melanie Subbiah, Jared D Kaplan, Prafulla Dhariwal, Arvind Neelakantan, Pranav Shyam, Girish Sastry, Amanda Askell, Sandhini Agarwal, Ariel Herbert-Voss, Gretchen Krueger, Tom Henighan, Rewon Child, Aditya Ramesh, Daniel Ziegler, Jeffrey Wu, Clemens Winter, Chris Hesse, Mark Chen, Eric Sigler, Mateusz Litwin, Scott Gray, Benjamin Chess, Jack Clark, Christopher Berner, Sam McCandlish, Alec Radford, Ilya Sutskever, and Dario Amodei. 2020. Language Models are Few-Shot Learners. In *Advances in Neural Information Processing Systems*, H. Larochelle, M. Ranzato, R. Hadsell, M.F. Balcan, and H. Lin (Eds.), Vol. 33. Curran Associates, Inc., 1877–1901. [https://proceedings.neurips.cc/paper\\_files/paper/2020/file/1457c0d6bfc967418bfb8ac142f64a-Paper.pdf](https://proceedings.neurips.cc/paper_files/paper/2020/file/1457c0d6bfc967418bfb8ac142f64a-Paper.pdf)
- [13] Ashutosh Dutta and Ehab Al-Shaer. 2019. "What", "Where", and "Why" Cybersecurity Controls to Enforce for Optimal Risk Mitigation. In *2019 IEEE Conference on Communications and Network Security (CNS)*. IEEE, 160–168.
- [14] Stjepan Gros. 2019. A Critical View on CIS Controls. *2021 16th International Conference on Telecommunications (ConTEL)* (2019), 122–128. <https://api.semanticscholar.org/CorpusID:203736481>
- [15] Yen-Ting Lin and Yun-Nung Chen. 2023. LLM-Eval: Unified Multi-Dimensional Automatic Evaluation for Open-Domain Conversations with Large Language Models. *arXiv:2305.13711 [cs.CL]*
- [16] Jiacheng Liu, Alisa Liu, Ximing Lu, Sean Welleck, Peter West, Ronan Le Bras, Yejin Choi, and Hannaneh Hajishirzi. 2022. Generated Knowledge Prompting for Commonsense Reasoning. *arXiv:2110.08387 [cs.CL]*
- [17] Swaroop Mishra, Daniel Khashabi, Chitta Baral, Yejin Choi, and Hannaneh Hajishirzi. 2021. Reframing Instructional Prompts to GPTk's Language. *ArXiv abs/2109.07830* (2021). <https://api.semanticscholar.org/CorpusID:237532411>
- [18] Wadkur Kurniawan Sedano and Muhammad Salman. 2021. Auditing Linux Operating System with Center for Internet Security (CIS) Standard. In *2021 International Conference on Information Technology (ICIT)*. IEEE, 466–471.
- [19] William Stern, Seng Jhing Goh, Nasheen Nur, Patrick J Aragon, and Thomas Mercer. 2024. Natural Language Explanations for Suicide Risk Classification Using Large Language Models. (2024).
- [20] Jason Wei, Xuezhi Wang, Dale Schuurmans, Maarten Bosma, brian ichter, Fei Xia, Ed H. Chi, Quoc V Le, and Denny Zhou. 2022. Chain of Thought Prompting Elicits Reasoning in Large Language Models. In *Advances in Neural Information Processing Systems*, Alice H. Oh, Alekh Agarwal, Danielle Belgrave, and Kyunghyun Cho (Eds.). [https://openreview.net/forum?id=\\_VjQlMeSB\\_J](https://openreview.net/forum?id=_VjQlMeSB_J)
- [21] Shunyu Yao, Dian Yu, Jeffrey Zhao, Izhak Shafran, Thomas L. Griffiths, Yuan Cao, and Karthik Narasimhan. 2023. Tree of Thoughts: Deliberate Problem Solving with Large Language Models. *CoRR abs/2305.10601* (2023). <https://doi.org/10.48550/arXiv.2305.10601>
- [22] Zhiyuan Zeng, Jiatong Yu, Tianyu Gao, Yu Meng, Tanya Goyal, and Danqi Chen. 2023. Evaluating Large Language Models at Evaluating Instruction Following. *ArXiv abs/2310.07641* (2023). <https://api.semanticscholar.org/CorpusID:263834884>
- [23] Yue Zhang, Yafu Li, Leyang Cui, Deng Cai, Lemao Liu, Tingchen Fu, Xinting Huang, Enbo Zhao, Yu Zhang, Yulong Chen, et al. 2023. Siren's song in the ai ocean: A survey on hallucination in large language models. *arXiv preprint arXiv:2309.01219* (2023).
- [24] Zihao Zhao, Eric Wallace, Shi Feng, Dan Klein, and Sameer Singh. 2021. Calibrate before use: Improving few-shot performance of language models. In *International Conference on Machine Learning*. PMLR, 12697–12706.