Automating AWS Security Controls: Leveraging Generative AI for Gherkin Script Generation

Chen Ling
Emory University
chen.ling@emory.edu

Mina Ghashami
Amazon
ghashami@amazon.com

Kyuhong Park
Amazon
kyuhongp@amazon.com

Ali Torkamani
Amazon
alitor@amazon.com

Nivedita Mangam
Amazon
nmangam@amazon.com

Bhavya Jain
Amazon
bkjain@amazon.com

Malini SS
Amazon
sasism@amazon.com

Felix Candelario
Amazon
fcandela@amazon.com

Farhan Diwan
Amazon
fdiwan@amazon.com

Mingrui Cheng
Amazon
cmingrui@amazon.com

***Abstract*—Security controls are mechanisms or policies designed for cloud based services to reduce risk, protect information, and ensure compliance with security regulations. The development of security controls is traditionally a labor-intensive and time-consuming process. This paper explores the use of Generative AI to accelerate the generation of security controls. We specifically focus on generating Gherkin codes which are the domain-specific language used to define the behavior of security controls in a structured and understandable format. By leveraging large language models and in-context learning, we propose a structured framework that reduces the time required for developing security controls from 2-3 days to less than one minute. Our approach integrates detailed task descriptions, step-by-step instructions, and retrieval-augmented generation to enhance the accuracy and efficiency of the generated Gherkin code. Initial evaluations on AWS cloud services demonstrate promising results, indicating that GenAI can effectively streamline the security control development process, thus providing a robust and dynamic safeguard for cloud-based infrastructures.**

***Index Terms*—security controls, generative AI, large language models.**

I. Introduction

In today's rapidly evolving digital landscape, ensuring the security and integrity of cloud-based infrastructures has emerged as an urgent priority. The vast and intricate nature of modern cloud environments, combined with the rising sophistication of cyber threats, necessitates the deployment of dynamic and robust security controls. These controls must evolve alongside the growing complexity of cloud services to effectively protect sensitive data, maintain system integrity, and prevent malicious attacks.

Cloud security controls are designed to detect, prevent, and mitigate risks to information systems. They serve as critical safeguards that protect against unauthorized access, data breaches, and system misconfigurations. However, ensuring the effectiveness and accuracy of these controls remains a significant challenge. Ensuring that these policies are correctly implemented and consistently enforced is paramount to a secure cloud infrastructure.

Traditionally, the development of security controls and policies has been a labor-intensive process. Security engineers are required to stay constantly updated on emerging threats and vulnerabilities, perform detailed threat modeling, and create custom code to implement security controls. This manual process is both time-consuming and resource-intensive, often leading to delays in deploying critical security measures. However, with recent advancements in machine learning and automation, new opportunities have emerged to streamline the creation and management of cloud security controls.

One promising development is the use of machine learning models to automatically generate access control policies based on existing logs and security data. For example, the work of [16] introduced a novel approach using metagraphs to verify access control policies. Metagraphs allow for precise modeling of relationships between services and users, facilitating the detection of potential errors or conflicts within policies. This enables organizations to verify that deployed policies align with high-level security specifications, ensuring accuracy and reducing the likelihood of misconfigurations.

Similarly, [23] presented an innovative system that automates the generation of access control policies for microservices by analyzing access logs. Using advanced techniques such as Word2Vec [8] for semantic understanding, DBSCAN [17] for clustering service interactions, and topological graph generation, the system dynamically infers relationships between services and generates the necessary policies. This approach allows cloud security policies to evolve in real time, adapting to changes in system architecture and usage patterns.

Moreover, [22] developed Text2Policy, a system that extracts access control policies directly from natural language documents using natural language processing (NLP) techniques. This system automates the conversion of high-level security requirements into enforceable specifications like XACML (eXtensible Access Control Markup Language), streamlining the implementation process and ensuring policy compliance.

These advancements demonstrate the potential of machine learning and automation to revolutionize the development and enforcement of cloud security controls. More recently, there is a growing interest in leveraging generative AI to streamline and enhance the development of security policies. Generative AI automates many labor-intensive aspects of policy creation,

reducing the time and effort needed to establish effective security measures. By accelerating the creation of policies, organizations can quickly adapt to emerging threats, safeguarding their infrastructures more effectively. For instance, [14] proposed a method that utilizes Large Language Models (LLMs) to automate the conversion of natural language security policies into machine-readable formats like XACML and Rego. This automation minimizes human errors and improves policy enforcement in complex cloud environments. Additionally, [5] introduced LLM-CloudSec, a system designed for fine-grained vulnerability detection in cloud applications using LLMs. By leveraging techniques such as Retrieval Augmented Generation (RAG) [11], LLM-CloudSec improves vulnerability classification and performs real-time code analysis.

These innovative systems highlight how machine learning, natural language processing (NLP), and generative AI can significantly advance the automation of security policy generation and verification, equipping organizations with more efficient tools to manage cloud security.

Despite these advancements, automatically generating security controls for AWS cloud services remains a challenging and unsolved problem. In this paper, we address this gap by proposing a novel approach for generating Gherkin scripts [2] to facilitate AWS security control development. Our approach leverages prompt engineering, a technique used to design precise input prompts that guide the behavior of language models to produce desired outputs. Prompt engineering ensures that the model understands the context and objectives of a task, leading to more accurate and relevant responses.

Contributions. Our contribution is as following:

- We utilized LLMs to speed up development of AWS security controls through generating their gherkin scripts. To the best of our knowledge, this is the first time generative AI is used to address this problem.
- We combined few techniques in prompt engineering to improve the performance of LLMs and enhance transparency in the final output they create. These techniques include "*Chain-of-Thoughts Reasoning*" [7] to break down the complex task of generating gherkins into a sequence of simpler steps, "*Retrieval-Augmented Generation (RAG)*" [11] to allow LLM to use an external source such as Boto3 API documentation [1] to retrieve relevant information for a given service and resource, and last but not least, "*In Context Learning*" [9], where we added positive examples of gherkins that are developed by security engineers to the prompt.
- Through quantitative evaluations and with human in the loop, we showed that quality of generated gherkins are high, achieving an average score of at least 3 out of 5, based on an evaluation rubric developed by subject matter experts.

The rest of this paper is organized as follows. In section II, we introduce problem formulation including motivation and scope. In section III we summarized recent related works. In section IV, we talk about gherkin generation with generative AI. Finally in sections V and VI we discuss our evaluation, experiments and results.

II. Problem Formulation

A. Motivation

The traditional development of security controls for AWS is indeed a multi-stage process, each step demanding careful attention to ensure robust and effective controls. The stages, as outlined in Figure 1, are as follows:

1. Identification of Service, Resource, and Control Type: This foundational stage involves security engineers determining the service and resource that need protection, along with the specific type of control required. This step is critical as it defines the scope and focus of the security measures.

2. Writing Gherkin Scripts: Gherkin, a domain-specific language, is used to clearly define the behavior of the security controls. These scripts, written in plain language, help stakeholders understand the expected outcomes, making this stage crucial for aligning technical and non-technical teams. Crafting these scripts demands a deep understanding of the security needs and the environment in which the controls will operate.

3. Review of Gherkin Scripts: Once written, the scripts undergo a meticulous review process by security engineers to ensure they accurately reflect the intended security controls. This step ensures that the scripts are ready for implementation and that they address the identified risks effectively.

4. Code Development: The approved Gherkin scripts are then translated into executable code, which represents the actual security controls to be deployed. This stage transforms the behavior described in the scripts into a functional form that can be integrated into the cloud environment.

5. Deployment and Testing: After code development, the security controls are deployed and tested. Thorough testing ensures that the controls work as expected and effectively mitigate risks. Continuous monitoring is crucial to maintain and adapt the controls to new threats over time.

This traditional process can take several weeks, especially due to the time-intensive steps of reviewing API documentation and validating Gherkin scripts. To overcome these challenges, innovative solutions like Generative AI can significantly expedite the creation of security controls, reducing development timelines while maintaining accuracy and efficacy.

B. Scope

In this paper, we focus specifically on the **Gherkin creation stage** within the broader security control development lifecycle. While the entire process of developing security controls involves multiple intricate steps—from identifying the service and resource to deploying and monitoring the final control—this paper hones in on the creation of Gherkin scripts, a pivotal part of the process.

Gherkin scripts serve as pseudocode for the final security control implementation. Written in a structured, plain language

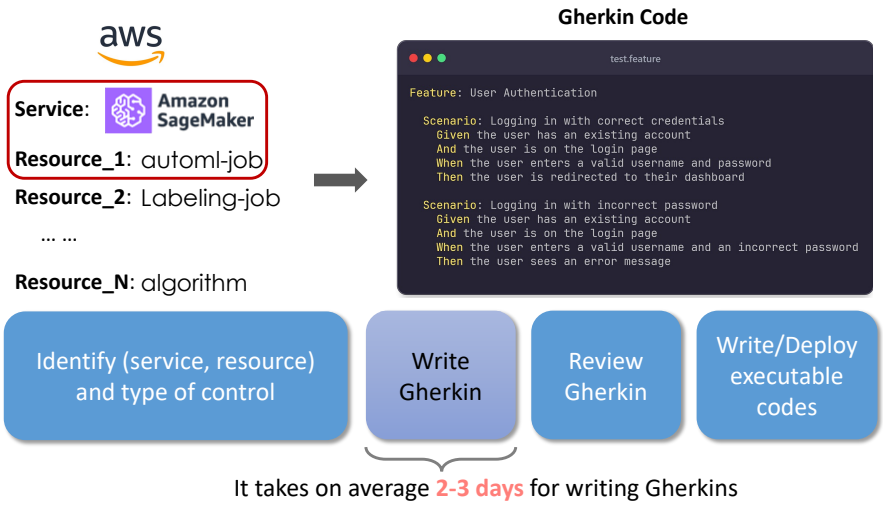

Fig. 1. An example of the security control development process, illustrating that the development of writing Gherkin can take on average 2-3 days.

format, Gherkins define the intended behavior of security controls in a way that is easy for both technical and non-technical stakeholders to understand. They outline how the security system should react under specific conditions, providing a blueprint for the behavior of the final product.

What makes Gherkin scripts so essential is their ability to bridge the gap between high-level requirements and the low-level code that is ultimately deployed in production. Once an accurate Gherkin script is in place, it provides a clear and straightforward pathway to the implementation phase, where the behavior described in the Gherkin is translated into executable code, such as AWS Lambda functions.

The structured format of Gherkin scripts ensures that the translation to Lambda code (or similar functions in other programming environments) is a relatively straightforward and automated process. Regardless of the programming language used (e.g., Python, Node.js, or Java), the logic and conditions outlined in the Gherkin can be directly converted into code, streamlining the implementation process. In essence, once a Gherkin script is accurate and fully reviewed, the development of the final product is simply a matter of turning the pseudocode into its programmatic equivalent.

III. Related Works

Machine Learning and LLMs for Security Policies. The use of machine learning models to automatically generate access control policies has been investigated in previous works. [16] introduced a metagraph-based approach to verify access control policies, which allows for modeling relationships between services and users to detect potential policy errors and misconfigurations. Similarly, [23] proposed Log2Policy, a system that utilizes semantic analysis and clustering techniques to infer relationships between microservices and dynamically generate access control policies. In the realm of NLP, several works have focused on the automatic extraction and generation of security policies from natural language documents. [22] developed Text2Policy, a system that automates the conversion of high-level security requirements into machine-readable formats such as XACML, reducing the manual effort involved in implementing security policies. The application of large language models (LLMs) has also shown promise in generating structured outputs for security tasks. In particular, recent advancements in retrieval-augmented generation (RAG) have enabled LLMs to improve the accuracy of generated content by retrieving relevant information during the generation process. [11] demonstrated how RAG techniques could enhance knowledge-intensive tasks, such as security policy generation and verification, by integrating external sources of information into the model's generation pipeline.

LLM for Structured Output. As Gherkin codes are a type of highly structured output, recent advancements in LLMs [4], [13] for structured output have demonstrated significant progress in several key areas. LLMs are being enhanced through instruction tuning and innovative approaches such as "reflection-tuning," which improves the quality of training data by self-evaluation and enhancement, resulting in better output alignment [12]. Additionally, models like GPT-4 [3] and LLAMA [19] have shown improvements in generating complex structured data by utilizing structure-aware fine-tuning and Chain-of-Thought techniques, significantly reducing formatting errors [18].

Few-shot/Zero-shot LLM Prompting. LLMs for zero/few-shot prompting has shown remarkable advancements in enhancing their reasoning and inference capabilities without requiring task-specific training examples. Notably, the introduction of strategies like Plan-and-Solve Prompting and Zero-shot Chain-of-Thought (CoT) prompting has significantly improved LLM performance on complex reasoning tasks by guiding the models to break down problems into smaller, manageable steps [10], [20], [21]. Furthermore, innovations such as UPRISE (Universal Prompt Retrieval for Improving Zero-Shot Evaluation) have improved the generalization and task adaptability of LLMs by retrieving and using relevant prompts across different models and tasks [6]. Additionally, methods like SelfCheck enable LLMs to verify and correct their step-by-step reasoning autonomously, thus enhancing

their accuracy and reliability in zero-shot settings [15].

IV. Gherkin Generation with Generative AI

Approach Overview. In this work, we adopt an innovative approach using in-context learning with retrieval-augmented generation to streamline the process. Our approach begins by providing the LLM with a detailed task description, breaking it down step-by-step to ensure clarity and precision. We then supply the model with existing examples of security controls, including their associated Gherkins. Finally, we present the LLM with the final query, guiding it to generate the desired security control efficiently.

A. Control Types

In this work, we focus on a set of critical control types identified as essential by subject matter experts (SMEs). These controls are designed to address key security and compliance requirements for cloud environments. The control types include: leftmargin=*

- *Encryption of Data at Rest*: Ensuring that all stored data is encrypted to protect it from unauthorized access.
- *Encryption of Data in Transit*: Securing data during transmission to prevent interception and tampering.
- *Tagging*: Implementing consistent and meaningful tags to manage and organize resources effectively.
- *Resources Running on a Supported Version*: Ensuring that resources are running on supported versions to mitigate vulnerabilities associated with outdated software.
- *Backup Enabled*: Guaranteeing that data is regularly backed up to enable recovery in case of data loss or corruption.
- *Multi-AZ Deployment*: Distributing resources across multiple Availability Zones to enhance fault tolerance and availability.
- *Inbound IP Connection Control*: Restricting inbound IP connections to resources to prevent unauthorized access.
- *Resource Accessibility*: Ensuring that resources cannot be accessed by anyone without proper authorization.
- *Audit Logging Enabled*: Enabling audit logging and specifying the destination for logs to facilitate monitoring and compliance.

These control types have been selected based on their importance in securing cloud environments and ensuring compliance with industry standards and best practices. For a detailed description of each control type, please refer to the appendix.

B. Detailed Task Description

As seen in the left part of Figure 2, the task description provides the LLM with an initial understanding of the overall objective of the task, which involves designing a detailed and structured prompt to guide the LLM through the generation process. The steps are as follows: 1) *Task Definition*: Clearly define the task to the LLM. In this case, the task is to generate Gherkin code for a specific control type (e.g., logging or version support). Emphasize the role of the LLM as an expert security engineer responsible for generating these controls. 2) *Context Provision*. Provide comprehensive context to the LLM about the importance of the security control. This context helps the LLM understand the rationale behind the security controls it needs to generate. 3) *Expected Output.* Define the structure of the output explicitly. The output should be a JSON formatted message with no additional text. The JSON should contain specific placeholders that the LLM needs to fill with relevant information.

C. Step-by-Step Instructions: CoT Reasoning

To ensure the LLM generates accurate and effective Gherkin code for security controls, we adopt a chain-of-thought approach. This approach breaks down the task into detailed, sequential steps, facilitating clear and logical reasoning. The instructions are divided into two main steps: step 1) Service, API Call, and Insecure Configuration Analysis, and step 2) Gherkin Code Generation.

Step 1: Service, API Call, and Insecure Configuration Analysis. First, identify the AWS resource in question, and then identify the relevant API call that provides information about this resource. Next, understand the security best practices related to the control type (e.g., version support and management or encryption of data-at-rest). Finally, formulate a list of checks to verify compliance with security best practices. Each check should be written in a format that the LLM can interpret. These checks will help identify whether a resource is compliant with the specified security standards.

Step 2: Generate Gherkin Code. Based on the analysis from Step 1, the next step is to generate Gherkin code. Gherkin is a language used to write structured, human-readable tests and specifications. This code will help provide a clear and executable structure for security controls. Start by defining the rule components. This includes the Rule Identifier, which is a unique identifier for the rule, and the Rule Name, which is a descriptive name that clearly indicates the rule's purpose. The Description should be a detailed explanation of what the rule checks for and why it is important. The Trigger specifies the condition that initiates the rule, which can be either "Periodic" or "Configuration Changes". If there are any specific parameters required for the rule, these should be defined in the Rule Parameters section. If there are no specific parameters, this section can be left empty.

D. In-Context Learning and Retrieval-Augmented Generation

After providing the detailed task description and the chain-of-thought reasoning instruction, our approach further combines the strengths of in-context learning and retrieval-augmented generation to create a robust framework for generating security controls. The process can be summarized as follows:

We design the in-context demonstrations by creating a detailed prompt that embeds examples of existing security controls to guide the LLM. We use public APIs[1] to gather background information about AWS security best practices and

[1]https://boto3.amazonaws.com/v1/documentation/api/latest/index.html

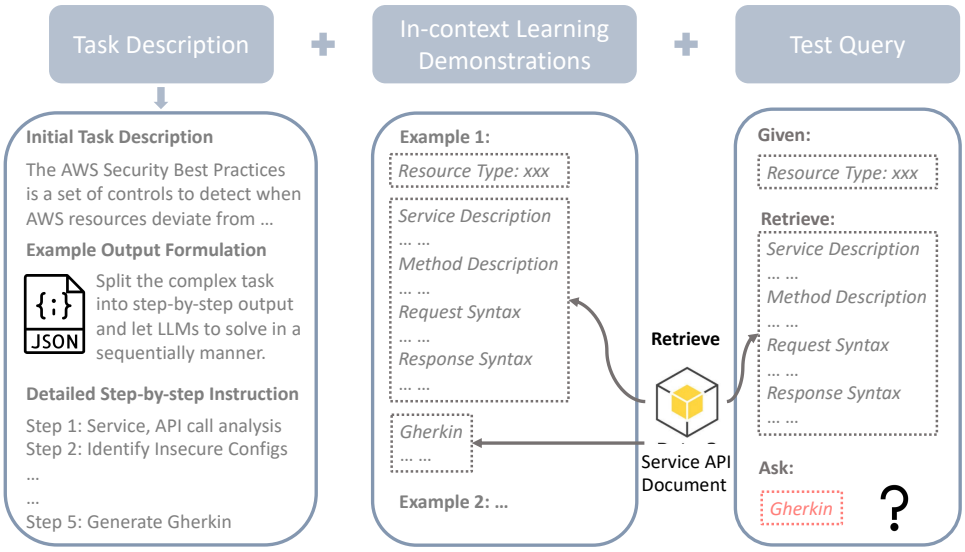

Fig. 2. An illustration of the Gherkin Generation framework.

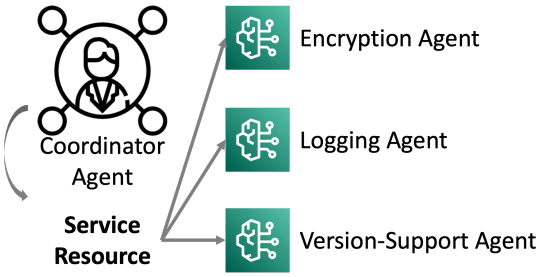

Fig. 3. The illustration of agent-based system for automated determination of control types (detailed in Section IV-A) given the service and resource name.

integrate this information into the prompt to provide the LLM with the necessary context. We then present the LLM with the final query, guiding it to generate the desired security control based on the provided context and examples. The LLM uses the embedded examples and retrieved information to produce accurate and effective Gherkin code. By leveraging these techniques, we can significantly reduce the time and effort required from security engineers to develop robust security controls. The LLM, equipped with detailed prompts and rich context, can efficiently generate Gherkin code that adheres to best practices in cloud security. This innovative approach not only streamlines the development process but also ensures the implementation of effective and reliable security measures in AWS environments.

E. Agent-based Security Control Type Identifier

As mentioned in Section IV-A, security engineers at AWS have identified and categorized nine top-priority security control types for which they are interested in generating Gherkin scripts. These control types have been chosen based on their critical importance in maintaining the security and compliance of cloud environments.

In addition to the primary prompt, we have implemented a feature that allows an LLM agent to determine which security control types are applicable to a given service and resource. This feature addresses a significant challenge faced by security engineers, who often spend considerable time identifying the appropriate control types for specific cloud services and resources. The LLM agent is equipped with detailed descriptions of each Security Control Type, enabling it to make informed decisions about applicability. For instance, the control type "Encryption of Data at Rest" is defined as: "*Data at rest refers to data stored in persistent, non-volatile storage for any duration. Encrypting data at rest helps protect its confidentiality, reducing the risk of unauthorized access. This control checks if the Resource is encrypted at rest. If the Resource is not encrypted at rest, the control will return NON_COMPLIANT. If the Resource is encrypted, it will return COMPLIANT.*"

By leveraging the LLM agent, we can automate the identification and application of relevant security controls, significantly reducing the manual effort required by security engineers. This system enhances the efficiency of the security control development process and ensures that the appropriate measures are in place to protect cloud services and resources.

V. Evaluation System

To ensure the effectiveness and accuracy of Gherkin scripts generated by generative AI, we employ a human-in-the-loop approach for evaluation. This approach leverages expert human judgment to assess the quality of generated Gherkins against a structured rubric. The rubric, shown in Table 1, provides a quantitative framework for evaluating Gherkins based on specific criteria, ensuring a consistent and objective assessment. We provide a more detailed description of each criteria in the appendix.

The overall score for a Gherkin script is calculated using the following formula:

$$score = (S1 + S2 + S3 + S4 + S5) \times (R1 + R2)/2 \quad (1)$$

This formula integrates the evaluations from both the scenario and rule criteria, and the final score is within the range of

Criteria	Evaluation
S1:	The number of scenarios recorded is correct.
S2:	The field specified in the scenario exists.
S3:	The resulting compliance status is possible.
S4:	The configuration of the resource specified by the scenario is possible.
S5:	The conclusion of the scenario is correct.
R1:	The rule name correctly describes the control specified by the collection of scenarios.
R2:	The description correctly describes the control specified by the collection of scenarios.

TABLE I

GHERKIN RUBRIC EVALUATES THE VALIDITY OF THE GENERATED GHERKIN CODE FROM TWO DIMENSIONS: 1) WHETHER THE GENERATED SCENARIOS ARE FEASIBLE; AND 2) WHETHER THE RULE IDENTIFIER AND DESCRIPTION CAN CORRECTLY REFLECT THE CONTROL SPECIFIED BY THE SCENARIOS.

	Encryption of Data-at-rest	Logging
Scenario Score	4.19	4.07
Rule Score	0.72	0.75
Final Score	3.02	3.05

TABLE II

THE AVERAGE SCORE OF GENERATED GHERKIN EVALUATION

$[0, 5]$. The acceptance threshold is ≥ 2.5 for the generated Gherkins, which indicates security engineers would need light supervision/revision to make the generated Gherkins into production.

VI. EXPERIMENTS

The experiments are conducted in collaboration with domain experts in AWS based on the evaluation metric as identified in Section V. We use data from all available AWS services and resources. For the use of LLM, we leverage CLAUDE-3-SONNET hosted on AWS Bedrock[2].

We first demonstrate two histograms of the evaluated Gherkins by domain experts. As can be seen from Figure 4, both histograms show that the majority of the generated Gherkins for Encryption and Logging types fall within the acceptable range of requiring slight to moderate revisions (score ≥ 2.5). The distribution patterns are similar, with most scores clustering around 3, suggesting that the generation process is relatively consistent in its output for both types. However, the consistency and concentration around the score of 3 indicate that further refinement in the generation process could help in reducing the amount of necessary revision.

We further break down the average score for two categories of security control (i.e., Encryption of data-at-rest and Logging). For each category, security engineers randomly picked ten generated Gherkins and reviewed them. The scores are depicted in Table II. Note that if the Final Score is greater than 4, then the generated Gherkin can be sent to development with little modifications. If the final score is above 2.5, the Gherkin would need moderate to slight amount of revision.

The table shows that the generated Gherkin scenarios for *Encryption of Data-at-Rest* and *Logging* are fairly accurate, with Scenario Scores of 4.19 and 4.07, respectively. However, the Rule Scores are low (0.72 and 0.75), indicating improvements are needed. Finally, the final score calculated by Eq. 1 of both control types are 3.42, and 3.32, respectively.

[2]https://aws.amazon.com/bedrock/

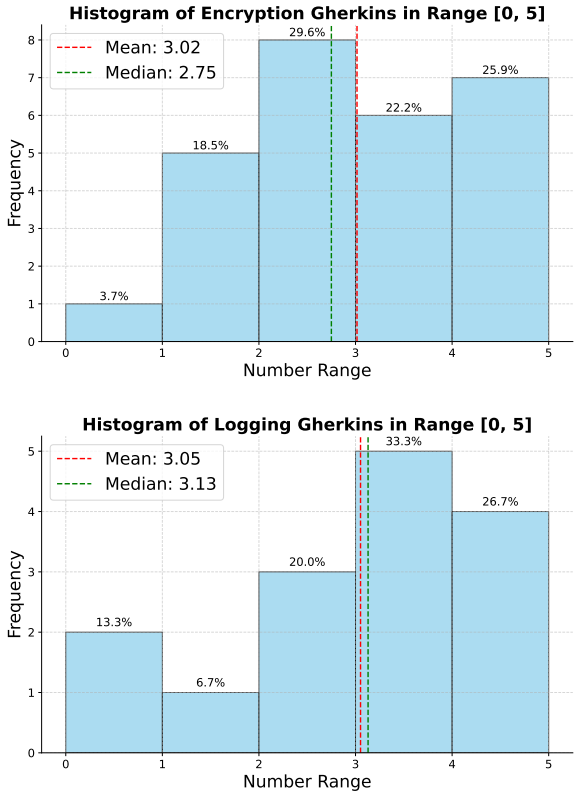

Fig. 4. The average score of generated Gherkin for the Encryption of data-at-rest Type and Logging Type.

VII. CONCLUSION

This study presents a novel framework for speeding up the generation of security controls using Generative AI, with a specific focus on producing Gherkin scripts. By incorporating large language models, detailed task descriptions, chain-of-thought reasoning, and retrieval-augmented generation, our approach can speed up the labor-intensive nature of traditional security control development from several days to less than one minute. Moreover, the evaluation results indicate that our method can meet the high standard of security controls

evaluated by domain experts, which can be sent to deployment with minor revisions as indicated in Table II. In summary, this framework significantly reduces the time and effort required from security engineers while maintaining the quality and reliability of the generated controls. This advancement highlights the potential of Generative AI to enhance the development and deployment of security measures in cloud environments, providing improved protection and adaptability against evolving cyber threats.

REFERENCES

[1] https://boto3.amazonaws.com/v1/documentation/api/latest/index.html.

[2] https://cucumber.io/docs/gherkin/.

[3] Josh Achiam, Steven Adler, Sandhini Agarwal, Lama Ahmad, Ilge Akkaya, Florencia Leoni Aleman, Diogo Almeida, Janko Altenschmidt, Sam Altman, Shyamal Anadkat, et al. Gpt-4 technical report. *arXiv preprint arXiv:2303.08774*, 2023.

[4] Guangji Bai, Zheng Chai, Chen Ling, Shiyu Wang, Jiaying Lu, Nan Zhang, Tingwei Shi, Ziyang Yu, Mengdan Zhu, Yifei Zhang, et al. Beyond efficiency: A systematic survey of resource-efficient large language models. *arXiv preprint arXiv:2401.00625*, 2024.

[5] Daipeng Cao and W Jun. Llm-cloudsec: Large language model empowered automatic and deep vulnerability analysis for intelligent clouds. In *IEEE INFOCOM 2024-IEEE Conference on Computer Communications Workshops (INFOCOM WKSHPS)*, pages 1–6. IEEE, 2024.

[6] Daixuan Cheng, Shaohan Huang, Junyu Bi, Yuefeng Zhan, Jianfeng Liu, Yujing Wang, Hao Sun, Furu Wei, Denvy Deng, and Qi Zhang. Uprise: Universal prompt retrieval for improving zero-shot evaluation. *arXiv preprint arXiv:2303.08518*, 2023.

[7] Zheng Chu, Jingchang Chen, Qianglong Chen, Weijiang Yu, Tao He, Haotian Wang, Weihua Peng, Ming Liu, Bing Qin, and Ting Liu. A survey of chain of thought reasoning: Advances, frontiers and future. *arXiv preprint arXiv:2309.15402*, 2023.

[8] Kenneth Ward Church. Word2vec. *Natural Language Engineering*, 23(1):155–162, 2017.

[9] Qingxiu Dong, Lei Li, Damai Dai, Ce Zheng, Zhiyong Wu, Baobao Chang, Xu Sun, Jingjing Xu, and Zhifang Sui. A survey on in-context learning. *arXiv preprint arXiv:2301.00234*, 2022.

[10] Takeshi Kojima, Shixiang Shane Gu, Machel Reid, Yutaka Matsuo, and Yusuke Iwasawa. Large language models are zero-shot reasoners. *Advances in neural information processing systems*, 35:22199–22213, 2022.

[11] Patrick Lewis, Ethan Perez, Aleksandra Piktus, Fabio Petroni, Vladimir Karpukhin, Naman Goyal, Heinrich Küttler, Mike Lewis, Wen-tau Yih, Tim Rocktäschel, et al. Retrieval-augmented generation for knowledge-intensive nlp tasks. *Advances in Neural Information Processing Systems*, 33:9459–9474, 2020.

[12] Ming Li, Lichang Chen, Jiuhai Chen, Shwai He, Heng Huang, Jiuxiang Gu, and Tianyi Zhou. Reflection-tuning: Data recycling improves llm instruction-tuning. *arXiv preprint arXiv:2310.11716*, 2023.

[13] Chen Ling, Xujiang Zhao, Jiaying Lu, Chengyuan Deng, Can Zheng, Junxiang Wang, Tanmoy Chowdhury, Yun Li, Hejie Cui, Tianjiao Zhao, et al. Domain specialization as the key to make large language models disruptive: A comprehensive survey. *arXiv preprint arXiv:2305.18703*, 2305, 2023.

[14] Fabio Martinelli, Francesco Mercaldo, Luca Petrillo, and Antonella Santone. Security policy generation and verification through large language models: A proposal. In *Proceedings of the Fourteenth ACM Conference on Data and Application Security and Privacy*, pages 143–145, 2024.

[15] Ning Miao, Yee Whye Teh, and Tom Rainforth. Selfcheck: Using llms to zero-shot check their own step-by-step reasoning. *arXiv preprint arXiv:2308.00436*, 2023.

[16] Loïc Miller, Pascal Mérindol, Antoine Gallais, and Cristel Pelsser. Verification of cloud security policies. In *2021 IEEE 22nd International Conference on High Performance Switching and Routing (HPSR)*, pages 1–5. IEEE, 2021.

[17] Erich Schubert, Jörg Sander, Martin Ester, Hans Peter Kriegel, and Xiaowei Xu. Dbscan revisited, revisited: why and how you should (still) use dbscan. *ACM Transactions on Database Systems (TODS)*, 42(3):1–21, 2017.

[18] Xiangru Tang, Yiming Zong, Yilun Zhao, Arman Cohan, and Mark Gerstein. Struc-bench: Are large language models really good at generating complex structured data? *arXiv preprint arXiv:2309.08963*, 2023.

[19] Hugo Touvron, Thibaut Lavril, Gautier Izacard, Xavier Martinet, Marie-Anne Lachaux, Timothée Lacroix, Baptiste Rozière, Naman Goyal, Eric Hambro, Faisal Azhar, et al. Llama: Open and efficient foundation language models. *arXiv preprint arXiv:2302.13971*, 2023.

[20] Lei Wang, Wanyu Xu, Yihuai Lan, Zhiqiang Hu, Yunshi Lan, Roy Ka-Wei Lee, and Ee-Peng Lim. Plan-and-solve prompting: Improving zero-shot chain-of-thought reasoning by large language models. *arXiv preprint arXiv:2305.04091*, 2023.

[21] Jason Wei, Xuezhi Wang, Dale Schuurmans, Maarten Bosma, Fei Xia, Ed Chi, Quoc V Le, Denny Zhou, et al. Chain-of-thought prompting elicits reasoning in large language models. *Advances in neural information processing systems*, 35:24824–24837, 2022.

[22] Xusheng Xiao, Amit Paradkar, Suresh Thummalapenta, and Tao Xie. Automated extraction of security policies from natural-language software documents. In *Proceedings of the ACM SIGSOFT 20th International Symposium on the Foundations of Software Engineering*, pages 1–11, 2012.

[23] Shaowen Xu, Qihang Zhou, Heqing Huang, Xiaoqi Jia, Haichao Du, Yang Chen, and Yamin Xie. Log2policy: An approach to generate fine-grained access control rules for microservices from scratch. In *Proceedings of the 39th Annual Computer Security Applications Conference*, pages 229–240, 2023.

VIII. APPENDIX

A. Detailed Description of Control Types

1) **Encryption of data at rest** - Data at rest refers to data that's stored in persistent, non-volatile storage for any duration. Encrypting data at rest helps you protect its confidentiality, which reduces the risk that an unauthorized user can access it. This control detects where resource is encrypted at rest. If the resource resource is not encrypted at rest the control will return NON_COMPLIANT. If the resource is encrypted it will return COMPLIANT
2) **Encryption of data in transit** - Data in transit refers to data that moves from one location to another, such as between nodes in your cluster or between your cluster and your application. Data may move across the internet or within a private network. Encrypting data in transit reduces the risk that an unauthorized user can eavesdrop on network traffic.
3) **Tagging** - A tag is a label that you assign to an AWS resource, and it consists of a key and an optional value. You can create tags to categorize resources by purpose, owner, environment, or other criteria. Tags can help you identify, organize, search for, and filter resources. Tagging also helps you track accountable resource owners for actions and notifications. When you use tagging, you can implement attribute-based access control (ABAC) as an authorization strategy, which defines permissions based on tags. You can attach tags to IAM entities (users or roles) and to AWS resources. You can create a single ABAC policy or a separate set of policies for your IAM principals. You can design these ABAC policies to allow operations when the principal's tag matches the resource tag.
4) **Resources run on supported version** - Running resources on supported software versions ensures optimal

performance, security, and access to the latest features. Regular updates safeguard against vulnerabilities, guaranteeing a stable and efficient user experience.

5) **Backup enabled** - A data backup is a copy of your system, configuration, or application data that's stored separately from the original. Enabling regular backups helps you safeguard valuable data against unforeseen events like system failures, cyberattacks, or accidental deletions. Having a robust backup strategy also facilitates quicker recovery, business continuity, and peace of mind in the face of potential data loss.
6) **Multi AZ** - An AZ is a distinct location within an AWS Region that is insulated from failures in other AZs. Enabling multiple AZs is recommended for enhanced resilience and fault tolerance in the event of infrastructure issues. Multi-AZ architectures aim to minimize the impact of infrastructure failures by dispersing resources across multiple AZs.
7) **Can inbound IP connections made to the resource** - Inbound connections can pose security risks by allowing unauthorized access to a system, leading to data breaches, system compromise, or exploitation of vulnerabilities. Robust security measures such as firewalls, access controls, and encryption are required to mitigate these risks.
8) **Can resource be accessed by anyone** - Publicly accessible resources can be lead to unauthorized access, data breaches or exploitation of vulnerabilities. Restricting access through authentication and authorization measures helps to safeguard sensitive information and maintain the integrity of your resources.
9) **Audit Logging enabled with destination log** - Audit logs track and monitor system activities. They provide a record of events that can help you detect security breaches, investigate incidents, and comply with regulations. Audit logs also enhance the overall accountability and transparency of your organization.

B. Evaluation Criteria

The evaluation rubric is divided into two main categories: Scenario Evaluation (S) and Rule Evaluation (R). Each category contains specific criteria that the human evaluator must consider.

1) Scenario Evaluation (S):

1) *(S1) The number of scenarios recorded is correct*. This criterion assesses whether the generated Gherkin includes the appropriate number of scenarios. Each scenario should represent a distinct and necessary test case for the security control.
2) *(S2) The field specified in the scenario exists*. This checks if all fields referenced in the scenarios are valid and present in the context of the security control being defined. It ensures the relevance and applicability of the scenarios.
3) *(S3) The resulting compliance status is possible*. This criterion evaluates whether the compliance status derived from the scenario is feasible. It ensures that the scenarios result in legitimate compliance outcomes.
4) *(S4) The configuration of the resource specified by the scenario is possible*. This ensures that the configuration actions described in the scenarios can actually be implemented within the given cloud environment. It checks for the practicality of the scenarios.
5) *(S5) The conclusion of the scenario is correct*. This criterion checks if the scenario logically concludes with the correct outcome based on the preceding steps. It verifies the logical flow and correctness of the scenario's outcome.

2) Rule Evaluation (R):

1) (R1) The rule name correctly describes the control specified by the collection of scenarios: This assesses the accuracy and appropriateness of the rule name. The rule name should succinctly and accurately reflect the control described by the scenarios.
2) (R2) The description correctly describes the control specified by the collection of scenarios: This criterion evaluates the clarity and correctness of the rule description, and whether it provides a clear understanding of the control and its purpose.

C. Scoring Mechanism

The overall score for a Gherkin script is calculated using the following formula:

$$score = (S1 + S2 + S3 + S4 + S5) \times (R1 + R2)/2 \quad (2)$$

This formula integrates the evaluations from both the scenario and rule criteria, ensuring a comprehensive assessment of the Gherkin script's quality.