Security Policy Generation for Cloud-Based Security Services using Large Language Model

Mikel Larrarte Rodriguez*, Jorge Alcorta Berasategui†, and Jaehoon (Paul) Jeong‡
*Faculty of Informatics, University of the Basque Country, San Sebastian, Spain
†Faculty of Engineering, University of Deusto, Bilbao, Spain
‡Department of Computer Science & Engineering, Sungkyunkwan University, Suwon, Republic of Korea
Email: {2024319650, 2024319370}@g.skku.edu, pauljeong@skku.edu

***Abstract*—This paper proposes an intelligent Security Policy Translator (SPT) to automate security policy generation using the GPT-4o-mini language model. This SPT focuses on translating natural language descriptions into XML policies compliant with the Consumer-Facing Interface (CFI) in the framework for Interface to Network Security Functions (I2NSF). The SPT simplifies the creation of policies for non-technical users, enabling seamless configuration of network security rules. By leveraging OpenAI's API, the model interprets a user input and outputs structured, machine-readable policies that are aligned with IETF I2NSF standards. The paper demonstrates the feasibility of integrating Large Language Models (LLM) into cybersecurity workflows and highlights opportunities to improve scalability, context awareness, and interoperability within the I2NSF framework.**

***Index Terms*—Large Language Model, I2NSF, Network Security Function, Consumer-Facing Interface, Security Policy Translator.**

I. Introduction

Defining and enforcing security policies is a fundamental aspect of modern cybersecurity. These policies govern how network traffic is managed to ensure compliance with security policies and protect sensitive data. However, creating accurate, machine-readable policies often requires technical expertise, making it challenging for non-expert users.

Interface to Network Security Functions (I2NSF) [1], which is standardized by the Internet Engineering Task Force (IETF), provides a standardized framework for defining security policies for cloud-based security services. These security policies are used to configure Network Security Functions (NSF) for the required security services in either a cloud system or an edge system. Its Consumer-Facing Interface (CFI) specifies an XML schema for policy representation, ensuring consistency and interoperability. Despite its advantages, the manual creation of such policies is time-consuming and prone to errors.

This paper addresses these challenges by leveraging GPT-4o-mini [2], which is a Large Language Model (LLM), is optimized for structured data generation, and supports powerful prompting. By carefully designing prompts, the system can translate natural language descriptions into I2NSF-compliant XML policies, more specifically into the Consumer-Facing Interface [3] XML policies. Prompting enables the model to effectively interpret a user intent and generate a highly accurate output, demonstrating the advantages of this approach for structured data tasks.

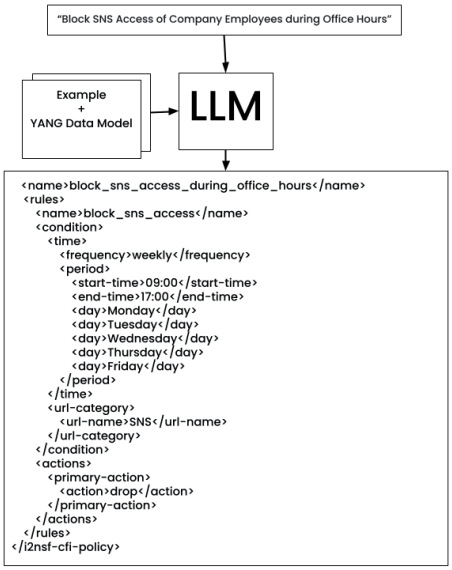

Fig. 1. Diagram of security policy translation with LLM.

This work highlights the potential of prompting as a tool for bridging the gap between a human intent and a machine-executable configuration, making policy creation be very accessible, efficient, and reliable.

The remainder of this paper is organized as follows. Section II provides related work in security policy enforcement in the I2NSF framework. Section III outlines the methodology used in designing our system to generate I2NSF security policies. Section IV presents the implementation details and the obtained results. Finally, Section VI concludes this paper along with future work.

II. Related Work

First of all, it is essential to briefly explain the Consumer-Facing Interface (CFI) [3]. This CFI in the I2NSF framework allows a security administrator to define high-level security policies that are subsequently translated into low-level configurations for enforcement by Network Security Functions (NSFs) such as firewall and web filter. Using an Event-Condition-Action (ECA) policy model, the CFI structures policies around specific events (e.g., system alarms), conditions (e.g., network traffic attributes and time-based restrictions), and actions (e.g.,

allowing, dropping, and rate-limiting traffic). The CFI also supports the definition of Endpoint Groups, which mean group network entities such as users, devices, and locations, simplifying the application of the policy. Integration with threat prevention allows for dynamic updates based on external risk data. The CFI's YANG data model ensures a standardized, machine-readable approach for defining and translating policies. It facilitates seamless communication between an I2NSF user and an I2NSF security controller in the I2NSF framework, and also reduces the complexity of security policy management.

III. Methodology

The core of this paper's methodology lies in the effective understanding of natural language descriptions and the careful design of prompts to translate these inputs into machine-readable structured XML policies compliant with I2NSF [1].

A. Natural Language Understanding

The I2NSF system employs GPT-4o-mini [2] as a language model optimized for structured data tasks to interpret and process a user input. This step is critical to bridge the gap between human-readable descriptions and machine-executable configurations.

The model is designed to extract the user's intent from natural language descriptions, identifying key components such as events, conditions, and actions. Conditions may include factors like specific time periods, days of the week, or URL categories, while actions involve tasks like allowing or blocking particular types of network traffic. This ensures that the system captures both the "what" (actions to be taken) and the "when" (conditions under which the actions should apply). Using GPT-4o-mini's advanced contextual understanding, the system accurately dissects complex policy descriptions.

B. Prompt Engineering

Prompting [4] is the cornerstone of this system, as carefully crafted prompts allow the model to generate the desired output without the need of fine-tuning it and consequentially and also without a large amount of data. In this case, the prompt is designed to guide the model's attention to relevant details such as time constraints, URL categories, and actions, thereby ensuring compliance with the I2NSF [1] schema. The prompt structure is intentionally designed to get specific elements of the security policy from the I2NSF user's input.

1) Prompt Selector: To further enhance the functionality of the prompting mechanism, we implemented a prompt-selector system. This mechanism leverages the *LangChain* [5] framework to dynamically integrate few-shot examples to improve the performance of the LLM. We detail the core components and their functionality within the prompt-selector mechanism as follows.

An essential element of this mechanism is the *PromptTemplate*, which ensures that prompts maintain a consistent format. The defined template uses the following structure:

```
Question: {input}
{output}
```

This template serves to pair a natural language input with the corresponding structured output, enabling seamless interaction with the language model. Predefined examples are provided as a key input to the system. One example consists of the following:

- **Input**: A natural language query, such as *"Restrict access to adult content from all devices."*
- **Output**: A structured XML file detailing a required security policy, adhering to a specific schema. Listing 1 shows this XML file for the required security policy.

```
<i2nsf-cfi-policy
    xmlns="urn:ietf:params:xml:ns:yang:ietf-i2nsf-cfi-policy">
    <name>restrict_adult_content_policy</name>
    <rules>
        <name>restrict_adult</name>
        <condition>
            <url-condition>
                <url-name>adult_content</url-name>
            </url-condition>
        </condition>
        <actions>
            <primary-action>
                <action>drop</action>
            </primary-action>
        </actions>
    </rules>
</i2nsf-cfi-policy>
```

Listing 1. A security policy for restricting adult content

The mechanism employs a *SemanticSimilarityExampleSelector*, which uses techniques such as vector embeddings to measure semantic similarity [6] between a user's query and predefined examples. Semantic similarity evaluates how closely the meanings of two pieces of text align with each other, even if the wording differs. By converting text into numerical representations (i.e., embeddings) in a high-dimensional space, the selector calculates the proximity between the query and predefined examples, identifying those most relevant to the query's intent. This ensures the selection of the most contextually relevant examples for the task. These examples are then integrated into a cohesive prompt using a *FewShotPromptTemplate*, enhancing the model's task-specific understanding. The enriched prompt is processed by the language model to generate an output, often in XML, which can be validated against schemas or business rules for domain alignment.

IV. Implementation

This section describes the design process of the prompt and shows a generated CFI security policy [3]. The general overview of the implementation is shown in Fig. 1. The source code of our implementation is available at the GitHub repository of https://github.com/jaehoonpauljeong/Data-Modeling-Group-2-Project. The demonstration video clip is available at the YouTube of https://youtu.be/yrNMPQB2R64.

When calling the model to translate the input into a security policy XML file, we designed a specific prompt to condition the model. In addition to making use of few-shot learning [7] providing the model with some examples of the objective task, our system includes a detailed YANG data model of the CFI to serve as a guideline for the model.

After defining the prompt, we generated some examples with different inputs to test the performance of the model.

Due to a lack of security policies, a quantitative evaluation of the implementation was not possible. Therefore, we evaluated the generated output taking its format, syntax, and contents into account.

```
%\begin{lstlisting}[language=XML, label={lst:example2}, caption={Input: "Block SNS
      access during office hours with a weekly frequency (9am-5pm;
      Monday-Friday)"}]
<?xml version="1.0" encoding="UTF-8" ?>
<i2nsf-cfi-policy
    xmlns="urn:ietf:params:xml:ns:yang:ietf-i2nsf-cfi-policy">
    <name>block_sns_access_during_office_hours</name>
    <rules>
        <name>block_sns_access</name>
        <condition>
            <time>
                <frequency>weekly</frequency>
                <period>
                    <start-time>09:00</start-time>
                    <end-time>17:00</end-time>
                    <day>Monday</day>
                    <day>Tuesday</day>
                    <day>Wednesday</day>
                    <day>Thursday</day>
                    <day>Friday</day>
                </period>
            </time>
            <url-category>
                <url-name>SNS</url-name>
            </url-category>
        </condition>
        <actions>
            <primary-action>
                <action>drop</action>
            </primary-action>
        </actions>
    </rules>
</i2nsf-cfi-policy>
```

Listing 2. A security policy for blocking SNS access during office hours

In the above example, we used the following input: *Block SNS access during office hours with a weekly frequency (9am-5pm; Monday-Friday)*. As shown in Listing 2, the result looks promising since the format of the XML scheme follows the provided guideline. The syntax is appropriate enough to capture the required contents to construct a security policy from the input text accordingly.

V. Execution Flow

The generation of XML policies has the following six steps.

1) **Define Examples with Input-Output Pairs:** Create a list of examples where each example consists of:
   - An *Input* in a natural language, such as "Block malicious websites for my son's computer."
   - An *Output* in the XML file for the corresponding security policy.

   This list serves as references for generating new outputs.
2) **Initialize Semantic Similarity Example Selector:** Use an embedding model to convert all example inputs into numerical vector embeddings. These embeddings are stored in a vector database to efficiently retrieve examples most relevant to a given query.
3) **Create Few-Shot Prompt Template:** Construct a prompt that combines the following components:
   - *Contextual instructions*: High-level guidance for generating XML security policies.
   - *Selected examples*: Examples retrieved based on their semantic similarity to the query, which are retrieved using semantic search algorithms.
   - *User input query*: The specific natural language query, which is an input in a high-level language, requiring an XML policy.
4) **Configure the Large Language Model (LLM):** Pass the constructed prompt to a language model, such as GPT-4. The model processes the input and generates the corresponding XML security policy.
5) **Process New Queries:** For a new input query:
   a) Search the vector database for semantically similar examples.
   b) Integrate the retrieved examples into the few-shot prompt template.
   c) Pass the final prompt to the LLM for output generation.
6) **Output:** Return the generated XML file as a security policy for the input text.

VI. Conclusion

This paper demonstrates the potential of Large Language Models (LLM) for a Security Policy Generator (SPT) in the I2NSF framework. The SPT with LLM automates the generation of an I2NSF security policy from a natural language description. By using prompt engineering and the structured YANG data model of the Consumer-Facing Interface, the proposed SPT simplifies policy creation for non-technical users, ensuring efficiency and compliance with the I2NSF framework. While qualitative evaluation shows promising results in terms of format and content alignment, future work will focus on expanding policy datasets for quantitative assessment, improving model scalability, and enhancing context awareness to address diverse cybersecurity scenarios. Moreover, the integration of our implementation with the I2NSF system will be considered for intelligent security provisioning.

Acknowledgments

This work was supported by the Institute of Information & Communications Technology Planning & Evaluation (IITP) grant funded by the Ministry of Science and ICT (MSIT), South Korea (No. RS-2024-00398199 and IITP-2025-RS-2023-00254129). Note that Jaehoon (Paul) Jeong is the corresponding author.

References

[1] P. Lingga, J. P. Jeong, and L. Dunbar, "Icsc: Intent-based closed-loop security control system for cloud-based security services," *IEEE Communications Magazine*, pp. 1–7, 2024.
[2] OpenAI. [Online]. Available: https://openai.com/index/gpt-4o-mini-advancing-cost-efficient-intelligence
[3] J. P. Jeong, C. Chung, T.-J. Ahn, R. Kumar, and S. Hares, "I2NSF Consumer-Facing Interface YANG Data Model," Internet Engineering Task Force, Internet-Draft draft-ietf-i2nsf-consumer-facing-interface-dm-31, May 2023, work in Progress. [Online]. Available: https://datatracker.ietf.org/doc/draft-ietf-i2nsf-consumer-facing-interface-dm/31/
[4] J. White, Q. Fu, S. Hays, M. Sandborn, C. Olea, H. Gilbert, A. Elnashar, J. Spencer-Smith, and D. C. Schmidt, "A prompt pattern catalog to enhance prompt engineering with chatgpt," 2023. [Online]. Available: https://arxiv.org/abs/2302.11382
[5] H. Chase, "LangChain," Oct. 2022. [Online]. Available: https://github.com/langchain-ai/langchain
[6] J. J. Jiang and D. W. Conrath, "Semantic similarity based on corpus statistics and lexical taxonomy," 1997. [Online]. Available: https://arxiv.org/abs/cmp-lg/9709008
[7] A. Parnami and M. Lee, "Learning from few examples: A summary of approaches to few-shot learning," 2022. [Online]. Available: https://arxiv.org/abs/2203.04291