Vision: “AccessFormer”: Feedback-Driven Access Control Policy Generation Framework

Sakuna Harinda Jayasundara
University of Auckland
sjay950@aucklanduni.ac.nz

Nalin Asanka Gamagedara Arachchilage
University of Auckland
nalin.arachchilage@auckland.ac.nz

Giovanni Russello
University of Auckland
g.russello@auckland.ac.nz

***Abstract*—Access control failures can cause data breaches, putting entire organizations at risk of financial loss and reputation damage. One of the main reasons for such failures is the mistakes made by system administrators when they manually generate low-level access control policies directly from high-level requirement specifications. Therefore, to help administrators in that policy generation process, previous research proposed graphical policy authoring tools and automated policy generation frameworks. However, in reality, those tools and frameworks are neither usable nor reliable enough to help administrators generate access control policies accurately while avoiding access control failures. Therefore, as a solution, in this paper, we present “AccessFormer”, a novel policy generation framework that improves both the usability and reliability of access control policy generation. Through the proposed framework, on the one hand, we improve the *reliability* of policy generation by utilizing Language Models (LMs) to generate, verify, and refine access control policies by incorporating the system’s as well as administrator’s feedback. On the other hand, we also improve the *usability* of the policy generation by proposing a usable policy authoring interface designed to help administrators understand policy generation mistakes and accurately provide feedback.**

I. Introduction

In June 2023, Microsoft AI (Artificial I ntelligence) researchers published a GitHub repository, allowing users to download open-source AI models and code for image recognition using an Azure storage URL (Uniform Resource Locator) [1]. However, instead of allowing users only to download the source code and AI models in a specific s torage bucket, the storage account administrator (the administrator can be a researcher or a system administrator) has accidentally given “full access” to the entire storage account through that URL [1]. As a result, 38 terabytes of sensitive information, including usernames and passwords to Microsoft services stored in the storage account, were leaked to the general public [1].

To avoid such incidents due to mistakes by system administrators, previous research first p roposed g raphical policy authoring tools [2]–[7] to guide administrators write access control policies manually even without knowledge of access control models, languages, or syntax [8]. However, the lack of usability of the existing graphical policy authoring tools makes it challenging for administrators to write policies accurately, resulting in access control failures [9], [10]. For example, in their user study, Brostoff et al. found that inexperienced administrators make mistakes when writing policies using their policy authoring tool, because administrators did not understand its features described in technical language [10].

Nevertheless, even if the usability of those tools is improved, administrators still have to extract policies from the high-level requirement specification documents and write them in the policy authoring tool manually to generate low-level access control configurations [11]. While doing that, they might make mistakes due to misinterpretations of those high-level policies [10]–[12], resulting in incorrect access control configurations that could lead to data breaches [1]. Therefore, as a solution, previous research then proposed fully automated policy generation frameworks attempting to remove administrator almost entirely from access control policy generation [11], [13]–[19]. Those frameworks process high-level natural language (NL) requirement specification documents [11] and translate containing sentences into machine-executable policies using machine learning (ML)/natural language processing (NLP) techniques without human feedback [20]. However, the existing automated policy generation frameworks are far from being 100% accurate to use without sufficient human feedback [21], [22] (i.e., lack of reliability). For example, the highest reported F1 score (the harmonic mean of the precision and recall [23]) in automated access control policy generation so far is 0.72 [13], indicating that the existing automated policy generation frameworks also make incorrect predictions, resulting in incorrect access control policies.

Therefore, to help administrators generate access control policies more accurately by improving both the usability and reliability of the access control policy generation process (i.e., by addressing the usability-security trade-off), in this paper, we introduce “AccessFormer”, a feedback-driven access control policy generation framework. To improve the reliability of access control policy generation, we first propose a design for a novel policy generation pipeline (Section III-A), leveraging recent advancements in transformer-based Language Models (LMs) and Large LMs (LLMs) [24]–[27]. It generates policies from high-level requirement specifications accurately by incorporating the system’s and administrator’s feedback based on an automatic policy verification technique [28], [29], which is lacking in existing literature [18], [30]. Then, allowing the administrator to accurately understand the system feedback and provide their feedback while improving the usability of policy generation, we provide design guidelines for developing a usable policy authoring interface (Section III-B).

Symposium on Usable Security and Privacy (USEC) 2024
26 February 2024, San Diego, CA, USA
ISBN 979-8-9894372-5-2
https://dx.doi.org/10.14722/usec.2024.23067
www.ndss-symposium.org, https://www.usablesecurity.net/USEC/

The rest of this paper is organized as follows. Section II presents a brief overview of related works, highlighting the research gap. To address that research gap, in Section III, we describe our proposed framework and how its elements can be incorporated to design a usable policy authoring tool. Finally, in Section IV, we summarize the work presented and conclude with an outlook on future works.

II. Related Works

Access control failures due to accidental mistakes made by system administrators can result in data breaches [12], [31]. Therefore, to avoid such access control failures, previous research has proposed tools and frameworks to help the administrator correctly generate access control policies from high-level requirement specifications by following two main approaches. They are **graphical policy authoring** [2], [6], [10], [32]–[34], and **automated policy generation** [11], [13], [14], [16], [17], [20], [35]–[38].

Graphical policy authoring tools provide a graphical user interface (GUI) to guide administrators in writing policies manually via text editors [8], [32], templates [33], [39] or access matrices [2], [34]. However, the **lack of usability** of those tools makes it difficult for system administrators to write accurate access control policies without human errors [9]. For example, most of the existing graphical policy authoring interfaces are not easily explainable to administrators [9], [10]. As a result, administrators often misinterpret their functionalities and write incorrect access control policies, causing access control failures that lead to data breaches [2], [3], [10].

On the other hand, automated policy generation frameworks use ML/NLP techniques to generate policies from high-level requirement specifications automatically with almost no human involvement [11], [16], [17], [19], according to 4 main steps: (1) pre-processing [16], [17], [35], (2) natural language access control policy (NLACP) identification [13], [16], [20], (3) policy component extraction [13], [16], [18], [19], [40], and (4) policy transformation [41]–[43]. However, the techniques used to perform the above steps are often less accurate due to lack of domain adaptation [44] and due to their inability to handle complex and ambiguous NLACPs [15], [41], [42], [45], resulting in incorrect access control policies in the end (i.e., **lack of reliability**). When such incorrect policies are applied to the system without even verifying or refining them, access control failures can occur, leading to data breaches [18].

Therefore, to improve the usability and reliability of the access control policy generation process, in this paper, we propose "AccessFormer", a novel access control policy generation framework that generates, verifies, and refines access control policies by incorporating feedback mechanisms.

III. Framework Overview

The proposed framework aims to improve the **usability** and **reliability** of the access control policy generation process, by allowing administrators to generate access control policies from high-level access requirements accurately. To do that, we propose a novel policy generation pipeline and a policy authoring interface, as highlighted in Fig. 1. According to Fig. 1, the policy generation process starts when the administrator provides a high-level NL requirement specification document

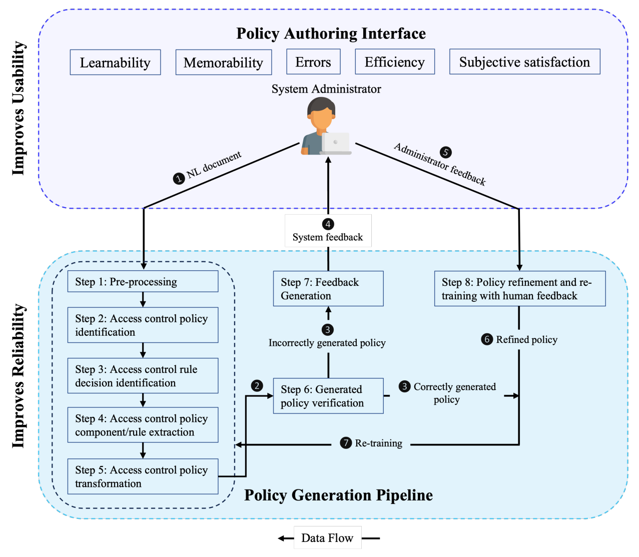

Policy Authoring Interface
Learnability
Memorability
Errors
Efficiency
Subjective satisfaction
System Administrator
Improves Usability
1 NL document
5 Administrator feedback
4 System feedback
Step 1: Pre-processing
Step 2: Access control policy identification
Step 3: Access control rule decision identification
Step 4: Access control policy component/rule extraction
Step 5: Access control policy transformation
Step 7: Feedback Generation
Step 8: Policy refinement and re-training with human feedback
3 Incorrectly generated policy
6 Refined policy
2
Step 6: Generated policy verification
3 Correctly generated policy
7 Re-training
Policy Generation Pipeline
Improves Reliability
Data Flow

Fig. 1: Proposed access control policy generation framework

as input to the policy generation pipeline through the interface. Once the document is provided, it will first be pre-processed. Then, NLACPs from the pre-processed document will be identified in Step 2, and the rules of identified NLACPs will be classified according to their access decision (i.e., allow or deny) in Step 3. After the above classification steps, we propose extracting necessary information such as policy components (i.e., users, actions, resources, purposes, and conditions)/access control rules from the NLACPs in Step 4, and transforming them into machine-executable format in Step 5.

However, the automated policy generation might not be accurate every time [18], [21]. It can generate incorrect policies, either due to the inherent complexities and ambiguities of unconstrained NLACPs [14], [43] or due to the false positives and false negatives of the NLP/ML models used to generate policies [14], [18], [22]. Therefore, we propose identifying such incorrectly generated policies (i.e., policies that do not match the high-level requirements) via an automatic verification mechanism [28], [29] in Step 6. Then, identified incorrect policies will be returned to the administrator with automatically generated feedback in Step 7, expecting the administrator's expertise to refine them. In that case, since administrators are also involved in ensuring the correctness of the policies by providing feedback, it will improve the reliability of the generated policies [18].

However, that feedback should be presented to system administrators in an easily explainable way by incorporating explainable security (XSec) concepts [46]. At the same time, administrators should be able to provide their feedback to refine incorrect policies without making human errors. Therefore, while improving the usability of policy generation, we propose designing a usable graphical policy authoring interface according to Nielsen's five usability quality components: memorability, efficiency, learnability, errors, and subjective satisfaction [47]. Finally, we suggest using the administrator's feedback collected through the interface to refine the incorrectly generated policy and to re-train the policy generation pipeline in Step 8 to improve the reliability and, in turn, security of access control policy generation further.

A. Policy Generation Pipeline

The policy generation pipeline is designed to generate machine-executable access control policies from high-level requirement specifications written in unconstrained natural language, according to eight steps shown in Fig. 1.

1) Step 1: Pre-processing
2) Step 2: Access control policy identification
3) Step 3: Access control rule decision identification
4) Step 4: Access control component/rule extraction
5) Step 5: Access control policy transformation
6) Step 6: Generated policy verification
7) Step 7: Feedback Generation
8) Step 8: Policy refinement and domain adaptation

*1) **Step 1: Pre-processing**:* The input to the policy generation framework (i.e., high-level requirement specification documents) often provides details on how information access is manipulated within the organization and who, under what circumstances, can access what resource in NL [14]. Those documents are often unstructured, ambiguous, and contain unwanted sentences [11]. Therefore, first, we propose removing such unwanted sentences, such as titles, using concise grammar rules, as advised by Slankas et al. [16], [17]. Secondly, we propose resolving co-references of the retained sentences as the next stage in the pre-processing step, which is rarely conducted in previous literature [14]. By resolving co-references, the meaning of the NLACP can be improved. For example, consider a NLACP saying that, *"Nurses are allowed to read the prescription, but they are not allowed to change it."*. After resolving co-references, the above NLACP will be modified as *"Nurses are allowed to read the prescription, but nurses are not allowed to change the prescription."*, resulting in more meaningful and exclusive rules (i.e., *"Nurses are allowed to read the prescription"* and *"nurses are not allowed to change the prescription"*) [15]. Then, we suggest performing subword tokenization, which breaks each sentence into sub-words and represents them with integers [48] using a technique such as Byte-Pair Encoding [49]. Ultimately, the NL sentences in high-level requirement specification documents will be represented by a set of integer sequences to feed into the LMs for further processing.

*2) **Step 2: Access control policy identification**:* After the pre-processing, in this step, the tokenized input sentences will be classified as NLACPs or sentences that do not contain access control policies (non-NLACPs). Among many techniques used in previous literature, such as Support Vector Machine (SVM) [17] and k-Nearest Neighbours (k-NN) [16], a transformer-based LM (BERT [27]), has achieved the highest policy identification F1-score of 0.92 [13]. Therefore, we suggest using transformer-based LMs, such as BERT [27], that are proven effective in text classification tasks [50] to classify sentences as NLACP and non-NLACP with high reliability.

*3) **Step 3: Access control rule decision identification**:* After identifying NLACPs, we propose classifying the access control rules of NLACPs based on their rule decision (i.e., rules that allow users to access a resource and rules that prevent users from accessing a resource) [13]. This step was rarely conducted in the existing automated policy generation frameworks due to the lack of domain-related data to train ML/NLP models to identify allow and deny access control rules separately [13], [18]. Nonetheless, we propose performing this step, as it helps decide whether or not the generated access control rule should be applied to the authorization system based on the default rules. For example, if the authorization system follows the default-deny principle, deny access control rules do not need to be added to the system to avoid redundant non-functional rules [10]. Therefore, similar to step 2, we propose using transformer-based bi-directional LMs such as BERT [27] to classify each access control rule of NLACPs into allow or deny rules [13]. After classifying the input sentences in Steps 2 and 3, access control rules/policy components from the NLACPs will be extracted in Step 4.

*4) **Step 4: Access control component/rule extraction**:* The high-level requirement specification documents often contain NLACPs written in unconstrained natural language that are sometimes ambiguous and complex in structure [15], [43]. Therefore, to extract their access control policy components/rules, deep learning-based information extraction techniques were often used in previous literature, such as neural network-based semantic role labeling (SRL) [13], [44]. By doing so, Xia et al. [13] were able to achieve the highest F1 score so far in access control rule extraction, which is 0.72, using a transformer-based SRL model [51]. However, almost all the deep learning-based access control policy component/rule extraction techniques used in previous literature were not fine-tuned using domain-related datasets [13]–[15], [44]. As a result, not only those extraction techniques have failed to extract access control policy components/rules with higher accuracy, but also they were not able to extract complex policy components such as purposes and conditions accurately [13], [14]. Therefore, we suggest utilizing transformer-based LLMs such as Falcon [26] and LLaMa [52] that show superior performances in similar tasks, including code generation from NL [53] to extract access control components/rules from NLACP, via supervised fine-tuning. To do that, we utilize the dataset introduced by Slankas et al. [16], which contains data from real-world high-level requirement specification documents. Then, we improve the dataset by generating synthetic data via augmentation techniques (e.g., back translation [13]) and annotating them manually. Using the improved dataset, LLMs can be adapted to the access control domain by improving the reliability of policy generation [44].

*5) **Step 5: Access control policy transformation**:* Even if the access control policy components are extracted from NLACPs, they cannot be directly applied to the authorization system, as they are not in a machine-executable format. Therefore, as the fifth step, we propose transforming the above information into a machine-executable policy in an access control language such as XACML (eXtensible Access Control Markup Language) [41], [42], [45] or in an intermediate representation format such as JSON (JavaScript Object Notation) [54], [55].

*6) **Step 6: Generated policy verification**:* Fully automated policy generation is not always successful as the ML/NLP techniques used for policy generation are often prone to false positives and false negatives (due to lack of domain adaptation [15]) [22]. As a result, the frameworks developed for automated policy generation might generate incorrect access control policies from NLACPs, leading to access control failures [56]. Furthermore, such incorrect policies will also be generated due to the complexity or [14], ambiguity [43] of NLACPs. Therefore, in contrast to existing fully automated policy generation pipelines [13], [14], [16], [18]–[20], we propose utilizing the administrator's feedback on refining such incorrectly generated policies before adding them to the authorization system to avoid access control failures. To do that, in Step 6, we suggest identifying incorrectly generated policies first, through an automatic verification mechanism using a trained verifier [28], [29]. Based on the verification result (i.e., the probability of the generated policy being correct given the NLACP), incorrectly generated policies will be identified using a verification probability threshold [28].

*7) **Step 7: Feedback Generation**:* Once a generated policy was identified as incorrect, we then suggest generating automatic system feedback in Step 7 using techniques such as reasoning (e.g., Chain-of-Thought [57]), mentioning the identified mistakes in the written NLACP and suggestions to avoid them [30]. By doing so, we attempt to make administrators more cautious when writing NLACPs, leading them to write more complete and unambiguous NLACPs that can be used to generate correct access control policies accurately.

*8) **Step 8: Policy refinement and domain adaptation**:* As the last step, we propose allowing administrators to provide feedback on the incorrectly generated policy based on the system feedback. Administrator's feedback can be a NL sentence describing why the generated policy was incorrect, such as missing rules or policy components. It will be used as part of the input to LLM (i.e., prompt) to generate more accurate policy iteratively [58] and to re-train the policy generation pipeline with the unique access requirements of the organization [18], using techniques such as Reinforcement Learning with Human Feedback (RLHF) [25] or Direct Preference Optimization (DPO) [59]. Consequently, the reliability of the policy generation will improve further [25] as it minimizes the data drift [60]. For instance, recent research utilized human feedback via RLHF [25] as well as automatic feedback via iterative prompting [58] to improve the reliability of domain-specific LLMs. Therefore, we suggest adopting a similar technique to fine-tune the policy generation pipeline using the organization-specific policies with the administrator's feedback.

Using the proposed policy generation pipeline, we attempt to improve the reliability of the overall access control policy generation process: (1) by making the administrator more cautious when writing NLACPs via system feedback (step 7) and (2) by refining generated policies and improving the policy generation pipeline via administrator's feedback (step 8). However, even if the system feedback is automatically generated, it should be presented to administrators in an easily explainable way to understand the feedback without misinterpreting it [47]. At the same time, administrators should also be allowed to provide feedback on the incorrectly generated policies to refine them before adding them to the authorization system. To do that, we propose designing a usable policy authoring interface, improving the usability of access control policy generation.

B. Policy Authoring Interface

We propose designing the policy authoring interface according to Nielsen's five usability quality components: memorability, efficiency, learnability, errors, and subjective satisfaction [47] as shown in Fig. 1.

*1) **Memorability**:* Memorability measures how easily administrators can reestablish their proficiency in using the policy authoring tool even after not using it for some time [47]. To achieve that, we propose a minimalistic interface with features, presented according to XSec [46] in easily understandable language and familiar concepts that match with real-world [61], [62]. For example, as shown in Fig. 2(e) in Appendix A, to navigate through incorrectly generated policies, we used left (go backward) and right (go forward) arrows as a metaphor that users might be familiar with [62], [63]. Such familiar metaphors, as well as the simple and descriptive language used to describe the functionality of a feature (using tooltips as in Fig. 2(i)), will help administrators recall how that feature works, even after not using the tool for some time [47], [62]. We then propose iteratively improving the initial interface via user studies by utilizing techniques such as Conceptual Design [64] as advised by Brostoff et al. [10]. Finally, to check the administrator's knowledge retention on how the interface works, we suggest conducting a longitudinal study [47], [65], which has not been conducted in the existing related literature.

*2) **Efficiency**:* Manual policy authoring [8], [10], [33] is a repetitive and time-consuming process, as administrators have to repetitively extract NLACPs from high-level requirement specification documents and write them manually using the policy authoring tool [11]. Therefore, we propose allowing administrators to input high-level requirement specification documents directly instead of engaging in the mentioned time-consuming process using a file browser, as shown in Fig. 2(a). As a result, the time administrators spend reading those documents and extracting policy sentences manually will be saved, increasing their efficiency [11]. Furthermore, administrators spend a significant amount of time searching for permissions of existing users before adding new permissions to them [34], which are often difficult to find in most policy authoring tools [2], [34]. Therefore, as a solution, we propose visualizing such information using a visualization technique (e.g., access matrix [66]) as shown in Fig. 2(h). Consequently, administrators can quickly find information about existing permissions without navigating through multiple windows or many lines of code, improving their efficiency [2], [34]. Following the same approach, Reeder et al. reduced the average policy configuration task completion time by 35.3s (40%) compared to the traditional Windows XPFP interface [2].

*3) **Learnability**:* According to Nielsen's usability quality components, "Learnability" measures how easy it is to learn and perform a given policy authoring task for the first time using the policy authoring tool [47]. Therefore, to improve the learnability of the policy authoring interface, we first propose making its features easily explainable by using label names and tooltips provided in a simple and descriptive way as shown in Fig, 2(i) in Appendix A. It will guide administrators step-by-step through the policy generation process, allowing them to learn the process easily [10], [62]. Brostoff et al. utilized the mentioned learnability improvement technique to improve their policy authoring interface by simplifying its label names used to define its policy configuration features, resulting in higher learnability [10]. Furthermore, according to Nielsen, allowing users to understand the current status of a system is another way of improving the learnability of the system as it bridges the gulf of evaluation [62], [64]. Therefore, to improve the learnability of the policy authoring interface as well as the underlying policy generation process, we then propose utilizing techniques such as progress bars and a summary of the current system status [62] to inform users about the ongoing policy generation as highlighted in Fig. 2(c) and Fig. 2(d).

*4) **Errors**:* In the proposed policy generation framework, human errors can occur either when adding new access control policies, such as goal errors [34], or when providing feedback on the incorrectly generated policies without understanding their errors [30]. For example, suppose administrators attempt to allow a group in an organization to access a resource without knowing about its permissions as well as the permissions of its users. In that case, they might even accidentally allow a group member to access the resource (i.e., a goal error [34]), even though that member is already restricted from accessing it, causing access control failures [3]. Therefore, to avoid such errors, we propose providing information about existing permissions in the system saliently and accurately within the interface using a visualization technique such as access matrices [2], [34] as shown in Fig. 2(h). By doing so, since the information required to write new policies is readily available, policy authoring mistakes due to goal errors will be reduced [34]. By following the same approach, Reeder et al. were able to achieve a 27.1% increment in policy configuration accuracy compared to the Windows XPFP interface [2]. However, even though some errors can be reduced by providing information about the existing policies, still the underlying policy generation pipeline can make errors when translating high-level requirements, as we pointed out in Section III-A. In such scenarios, the administrator should be able to clearly understand those errors and provide accurate feedback to refine the incorrectly generated policies [30]. To do that, we then propose presenting the system feedback in an easily explainable way, according to the XSec concepts [46] as highlighted in Fig. 2(f), to help administrators correctly understand the error and provide feedback [47].

*5) **Subjective satisfaction**:* Subjective satisfaction refers to how pleasant the interface is to use by the system administrators [47]. Therefore, to make the interface pleasant, we first propose organizing the features of the interface so that similar features are clustered together [67], as shown in Fig. 2. For example, we divide the interface shown in Fig. 2 into two portions: the left portion for providing inputs and setting configurations (highlighted in green) and the right portion for displaying results (highlighted in yellow). Furthermore, we present an incorrectly generated policy in a table, clearly categorizing similar policy components together as highlighted in Fig. 2(g), improving its readability. As a result, the interface will become more organized, making it easy for administrators to find necessary information, leading to higher pleasantness and higher subjective satisfaction [67]. Secondly, we propose allowing administrators to provide policies in unconstrained natural language in contrast to existing policy authoring tools [10], [41], [42], [45], to improve subjective satisfaction further. By doing so, since the "naturalness" of the language used to write policies is improved, administrators will easily be able to transfer their mental plans into the policy authoring tool, improving their satisfaction with it [8], [43]. For instance, as Shi et al. observed, after utilizing a Controlled Natural Language instead of syntactically strict PERMIS language to write policies, user study participants preferred the proposed tool over the traditional PERMIS policy authoring tool [42].

IV. Conclusion and Future Works

When system administrators attempt to write machine-executable access control policies directly from high-level requirement specifications written by security experts, administrators sometimes make mistakes [12]. Even though there are numerous tools and frameworks proposed to help administrators avoid such mistakes, they either lack usability [2], [41], [42] or lack reliability [11], [18], [21]. Therefore, to help administrators correctly generate access control policies from high-level requirement specifications by improving both the reliability and usability of traditional policy generation approaches [13], [16], [20], [41], in this paper, we propose a novel policy generation framework.

Through the framework, we introduced a novel policy generation pipeline that was designed to translate high-level requirement specifications into low-level access control configurations reliably using feedback mechanisms (i.e., system and administrator). Those feedback mechanisms also improve the adaptability of the policy generation framework to different and unique access requirements of different types of organizations, which is lacking in the current policy generation frameworks [15], [18]. Secondly, we explained how the elements of the policy generation pipeline shown in Fig. 1 can be incorporated to design a usable policy authoring interface shown in Fig. 2 according to Nielsen's usability quality components [47]. For example, by presenting the automatically generated system feedback (step 7 in Fig. 1) in an easily explainable way through XSec, administrators will be able to easily understand errors and recover from them [30], [62]. Similarly, we explained how to improve the learnability, efficiency, memorability, and subjective satisfaction of the policy generation tool and, in turn, improve its usability.

We designed and developed the initial prototype of the proposed policy generation framework. As future works, first, we develop the policy generation pipeline discussed in Section III-A using transformer-based LMs/LLMs to improve the reliability of the policy generation compared to the existing frameworks [13] in terms of F1 score [13]. Secondly, we iteratively improve the policy authoring interface by involving system administrators through participatory design [68]. After implementing the entire framework (reliable policy generation pipeline and usable interface according to Nielsen's five usability quality components [47] as discussed in Sections III-A and III-B) through the prototype, its reliability and usability will be evaluated empirically through a lab study [2], [41], involving system administrators. Once administrators interacted with the developed prototype, we will evaluate their satisfaction with the prototype in the end, according to standard evaluation instruments (e.g., SUS (System Usability Scale) [69] and PSSUQ (Post-Study System Usability Questionnaire) [70]), and think aloud data will be used to evaluate the overall reliability (i.e., security) of the policy generation.

Acknowledgment

The authors thank MBIE (Ministry of Business, Innovation, and Employment), New Zealand for funding this research.

REFERENCES

[1] C. Page, “Microsoft ai researchers accidentally exposed terabytes of internal sensitive data,” Sep 2023. [Online]. Available: https://techcrunch.com/2023/09/18/microsoft-ai-researchers-accidentally-exposed-terabytes-of-internal-sensitive-data/

[2] R. W. Reeder, L. Bauer, L. F. Cranor, M. K. Reiter, K. Bacon, K. How, and H. Strong, “Expandable grids for visualizing and authoring computer security policies,” in *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*, 2008, pp. 1473–1482.

[3] R. W. Reeder, L. Bauer, L. F. Cranor, M. K. Reiter, and K. Vaniea, “More than skin deep: measuring effects of the underlying model on access-control system usability,” in *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*, 2011, pp. 2065–2074.

[4] C. Morisset and D. Sanchez, “On building a visualisation tool for access control policies,” in *International Conference on Information Systems Security and Privacy*. Springer, 2018, pp. 215–239.

[5] A. Bertard and J.-K. Kopp, “Using sugiyama-styled graphs to directly manipulate role-based access control configurations,” in *International Conference on Human-Computer Interaction*. Springer, 2020, pp. 405–412.

[6] K. Vaniea, Q. Ni, L. Cranor, and E. Bertino, “Access control policy analysis and visualization tools for security professionals,” in *SOUPS Workshop (USM)*, 2008, pp. 7–15.

[7] C. Morisset and D. Sanchez, “Visabac: A tool for visualising abac policies.” in *ICISSP*, 2018, pp. 117–126.

[8] B. Stepien, A. Felty, and S. Matwin, “A non-technical user-oriented display notation for xacml conditions,” in *International Conference on E-Technologies*. Springer, 2009, pp. 53–64.

[9] R. W. Reeder, C.-M. Karat, J. Karat, and C. Brodie, “Usability challenges in security and privacy policy-authoring interfaces,” in *IFIP Conference on Human-Computer Interaction*. Springer, 2007, pp. 141–155.

[10] S. Brostoff, M. A. Sasse, D. Chadwick, J. Cunningham, U. Mbanaso, and S. Otenko, “‘r-what?’development of a role-based access control policy-writing tool for e-scientists,” *Software: Practice and Experience*, vol. 35, no. 9, pp. 835–856, 2005.

[11] M. Narouei, H. Khanpour, and H. Takabi, “Identification of access control policy sentences from natural language policy documents,” in *IFIP Annual Conference on Data and Applications Security and Privacy*. Springer, 2017, pp. 82–100.

[12] L. Bauer, L. F. Cranor, R. W. Reeder, M. K. Reiter, and K. Vaniea, “Real life challenges in access-control management,” in *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*, 2009, pp. 899–908.

[13] Y. Xia, S. Zhai, Q. Wang, H. Hou, Z. Wu, and Q. Shen, “Automated extraction of abac policies from natural-language documents in healthcare systems,” in *2022 IEEE International Conference on Bioinformatics and Biomedicine (BIBM)*. IEEE, 2022, pp. 1289–1296.

[14] M. Narouei and H. Takabi, “Automatic top-down role engineering framework using natural language processing techniques,” in *IFIP International Conference on Information Security Theory and Practice*. Springer, 2015, pp. 137–152.

[15] ——, “Towards an automatic top-down role engineering approach using natural language processing techniques,” in *Proceedings of the 20th ACM Symposium on Access Control Models and Technologies*, 2015, pp. 157–160.

[16] J. Slankas, X. Xiao, L. Williams, and T. Xie, “Relation extraction for inferring access control rules from natural language artifacts,” in *Proceedings of the 30th annual computer security applications conference*, 2014, pp. 366–375.

[17] J. Slankas and L. Williams, “Access control policy extraction from unconstrained natural language text,” in *2013 International Conference on Social Computing*. IEEE, 2013, pp. 435–440.

[18] J. Heaps, R. Krishnan, Y. Huang, J. Niu, and R. Sandhu, “Access control policy generation from user stories using machine learning,” in *IFIP Annual Conference on Data and Applications Security and Privacy*. Springer, 2021, pp. 171–188.

[19] M. Alohaly, H. Takabi, and E. Blanco, “Automated extraction of attributes from natural language attribute-based access control (abac) policies,” *Cybersecurity*, vol. 2, no. 1, pp. 1–25, 2019.

[20] X. Xiao, A. Paradkar, S. Thummalapenta, and T. 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*, 2012, pp. 1–11.

[21] M. Kaur, M. van Eeten, M. Janssen, K. Borgolte, and T. Fiebig, “Human factors in security research: Lessons learned from 2008-2018,” *arXiv preprint arXiv:2103.13287*, 2021.

[22] J. M. Del Alamo, D. S. Guaman, B. García, and A. Diez, “A systematic mapping study on automated analysis of privacy policies,” *Computing*, pp. 1–24, 2022.

[23] Z. C. Lipton, C. Elkan, and B. Naryanaswamy, “Optimal thresholding of classifiers to maximize f1 measure,” in *Machine Learning and Knowledge Discovery in Databases: European Conference, ECML PKDD 2014, Nancy, France, September 15-19, 2014. Proceedings, Part II 14*. Springer, 2014, pp. 225–239.

[24] A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, Ł. Kaiser, and I. Polosukhin, “Attention is all you need,” *Advances in neural information processing systems*, vol. 30, 2017.

[25] L. Ouyang, J. Wu, X. Jiang, D. Almeida, C. L. Wainwright, P. Mishkin, C. Zhang, S. Agarwal, K. Slama, A. Ray *et al.*, “Training language models to follow instructions with human feedback,” *arXiv preprint arXiv:2203.02155*, 2022.

[26] G. Penedo, Q. Malartic, D. Hesslow, R. Cojocaru, A. Cappelli, H. Alobeidli, B. Pannier, E. Almazrouei, and J. Launay, “The refined-web dataset for falcon llm: outperforming curated corpora with web data, and web data only,” *arXiv preprint arXiv:2306.01116*, 2023.

[27] J. Devlin, M.-W. Chang, K. Lee, and K. Toutanova, “Bert: Pre-training of deep bidirectional transformers for language understanding,” *arXiv preprint arXiv:1810.04805*, 2018.

[28] A. Ni, S. Iyer, D. Radev, V. Stoyanov, W.-t. Yih, S. Wang, and X. V. Lin, “Lever: Learning to verify language-to-code generation with execution,” in *International Conference on Machine Learning*. PMLR, 2023, pp. 26 106–26 128.

[29] J. Shen, Y. Yin, L. Li, L. Shang, X. Jiang, M. Zhang, and Q. Liu, “Generate & rank: A multi-task framework for math word problems,” *arXiv preprint arXiv:2109.03034*, 2021.

[30] T. Xu, H. M. Naing, L. Lu, and Y. Zhou, “How do system administrators resolve access-denied issues in the real world?” in *Proceedings of the 2017 CHI Conference on Human Factors in Computing Systems*, 2017, pp. 348–361.

[31] M. X. Heiligenstein, “Facebook data breaches: Full timeline through 2023,” May 2023. [Online]. Available: https://firewalltimes.com/facebook-data-breach-timeline

[32] B. Stepien, A. Felty, and S. Matwin, “A non-technical xacml target editor for dynamic access control systems,” in *2014 International conference on collaboration technologies and systems (CTS)*. IEEE, 2014, pp. 150–157.

[33] M. Johnson, J. Karat, C.-M. Karat, and K. Grueneberg, “Optimizing a policy authoring framework for security and privacy policies,” in *Proceedings of the Sixth Symposium on Usable Privacy and Security*, 2010, pp. 1–9.

[34] R. A. Maxion and R. W. Reeder, “Improving user-interface dependability through mitigation of human error,” *International Journal of human-computer studies*, vol. 63, no. 1-2, pp. 25–50, 2005.

[35] J. Slankas and L. Williams, “Classifying natural language sentences for policy,” in *2012 IEEE International Symposium on Policies for Distributed Systems and Networks*. IEEE, 2012, pp. 33–36.

[36] ——, “Access control policy identification and extraction from project documentation,” *SCIENCE*, vol. 2, no. 3, pp. 145–159, 2013.

[37] M. Alohaly and H. Takabi, “Better privacy indicators: a new approach to quantification of privacy policies,” in *Twelfth Symposium on Usable Privacy and Security (SOUPS 2016)*, 2016.

[38] M. Alohaly, H. Takabi, and E. Blanco, “Towards an automated extraction of abac constraints from natural language policies,” in *IFIP International Conference on ICT Systems Security and Privacy Protection*. Springer, 2019, pp. 105–119.

[39] M. Johnson, J. Karat, C.-M. Karat, and K. Grueneberg, "Usable policy template authoring for iterative policy refinement," in *2010 IEEE International Symposium on Policies for Distributed Systems and Networks*. IEEE, 2010, pp. 18–21.

[40] M. Narouei, H. Khanpour, H. Takabi, N. Parde, and R. Nielsen, "Towards a top-down policy engineering framework for attribute-based access control," in *Proceedings of the 22nd ACM on Symposium on Access Control Models and Technologies*, 2017, pp. 103–114.

[41] C. A. Brodie, C.-M. Karat, and J. Karat, "An empirical study of natural language parsing of privacy policy rules using the sparcle policy workbench," in *Proceedings of the second symposium on Usable privacy and security*, 2006, pp. 8–19.

[42] L. Shi and D. W. Chadwick, "A controlled natural language interface for authoring access control policies," in *proceedings of the 2011 ACM Symposium on Applied Computing*, 2011, pp. 1524–1530.

[43] P. Inglesant, M. A. Sasse, D. Chadwick, and L. L. Shi, "Expressions of expertness: the virtuous circle of natural language for access control policy specification," in *Proceedings of the 4th symposium on Usable privacy and security*, 2008, pp. 77–88.

[44] M. Narouei, H. Takabi, and R. Nielsen, "Automatic extraction of access control policies from natural language documents," *IEEE Transactions on Dependable and Secure Computing*, vol. 17, no. 3, pp. 506–517, 2018.

[45] C. Brodie, C.-M. Karat, J. Karat, and J. Feng, "Usable security and privacy: a case study of developing privacy management tools," in *Proceedings of the 2005 symposium on Usable privacy and security*, 2005, pp. 35–43.

[46] L. Vigano and D. Magazzeni, "Explainable security," in *2020 IEEE European Symposium on Security and Privacy Workshops (EuroS&PW)*. IEEE, 2020, pp. 293–300.

[47] J. Nielsen, *Usability engineering*. Morgan Kaufmann, 1994.

[48] S. J. Mielke, Z. Alyafeai, E. Salesky, C. Raffel, M. Dey, M. Gallé, A. Raja, C. Si, W. Y. Lee, B. Sagot *et al.*, "Between words and characters: a brief history of open-vocabulary modeling and tokenization in nlp," *arXiv preprint arXiv:2112.10508*, 2021.

[49] R. Sennrich, B. Haddow, and A. Birch, "Neural machine translation of rare words with subword units," *arXiv preprint arXiv:1508.07909*, 2015.

[50] M. Hoang, O. A. Bihorac, and J. Rouces, "Aspect-based sentiment analysis using bert," in *Proceedings of the 22nd nordic conference on computational linguistics*, 2019, pp. 187–196.

[51] P. Shi and J. Lin, "Simple bert models for relation extraction and semantic role labeling," *arXiv preprint arXiv:1904.05255*, 2019.

[52] H. Touvron, T. Lavril, G. Izacard, X. Martinet, M.-A. Lachaux, T. Lacroix, B. Rozière, N. Goyal, E. Hambro, F. Azhar *et al.*, "Llama: Open and efficient foundation language models," *arXiv preprint arXiv:2302.13971*, 2023.

[53] D. Zan, B. Chen, F. Zhang, D. Lu, B. Wu, B. Guan, W. Yongji, and J.-G. Lou, "Large language models meet nl2code: A survey," in *Proceedings of the 61st Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, 2023, pp. 7443–7464.

[54] F. Liu, S. Wilson, P. Story, S. Zimmeck, and N. Sadeh, "Towards automatic classification of privacy policy text," *School of Computer Science Carnegie Mellon University*, 2018.

[55] M. Rosa, J. P. Barraca, A. Zuquete, and N. P. Rocha, "A parser to support the definition of access control policies and rules using natural languages," *Journal of Medical Systems*, vol. 44, no. 2, pp. 1–12, 2020.

[56] I. F. del Amo, J. A. Erkoyuncu, R. Roy, R. Palmarini, and D. Onoufriou, "A systematic review of augmented reality content-related techniques for knowledge transfer in maintenance applications," *Computers in Industry*, vol. 103, pp. 47–71, 2018.

[57] J. Wei, X. Wang, D. Schuurmans, M. Bosma, F. Xia, E. Chi, Q. V. Le, D. Zhou *et al.*, "Chain-of-thought prompting elicits reasoning in large language models," *Advances in Neural Information Processing Systems*, vol. 35, pp. 24 824–24 837, 2022.

[58] N. Yoshikawa, M. Skreta, K. Darvish, S. Arellano-Rubach, Z. Ji, L. Bjørn Kristensen, A. Z. Li, Y. Zhao, H. Xu, A. Kuramshin *et al.*, "Large language models for chemistry robotics," *Autonomous Robots*, pp. 1–30, 2023.

[59] R. Rafailov, A. Sharma, E. Mitchell, S. Ermon, C. D. Manning, and C. Finn, "Direct preference optimization: Your language model is secretly a reward model," *arXiv preprint arXiv:2305.18290*, 2023.

[60] A. Mallick, K. Hsieh, B. Arzani, and G. Joshi, "Matchmaker: Data drift mitigation in machine learning for large-scale systems," *Proceedings of Machine Learning and Systems*, vol. 4, pp. 77–94, 2022.

[61] B. Saket, A. Endert, and J. Stasko, "Beyond usability and performance: A review of user experience-focused evaluations in visualization," in *Proceedings of the Sixth Workshop on Beyond Time and Errors on Novel Evaluation Methods for Visualization*, 2016, pp. 133–142.

[62] J. Nielsen, "Ten usability heuristics," 2005.

[63] G. Lakoff and M. Johnson, *Metaphors we live by*. University of Chicago press, 2008.

[64] D. A. Norman, "Some observations on mental models," in *Mental models*. Psychology Press, 2014, pp. 15–22.

[65] J. Kjeldskov, M. B. Skov, and J. Stage, "Does time heal?: a longitudinal study of usability," in *Proceedings of the Australian Computer-Human Interaction Conference 2005 (OzCHI'05)*. Association for Computing Machinery, 2005.

[66] B. W. Lampson, "Protection," *ACM SIGOPS Operating Systems Review*, vol. 8, no. 1, pp. 18–24, 1974.

[67] P. Balatsoukas, A. Morris, and A. O'Brien, "Designing metadata surrogates for search result interfaces of learning object repositories: Linear versus clustered metadata design." in *ELPUB*, 2007, pp. 415–424.

[68] D. Schuler and A. Namioka, *Participatory design: Principles and practices*. CRC Press, 1993.

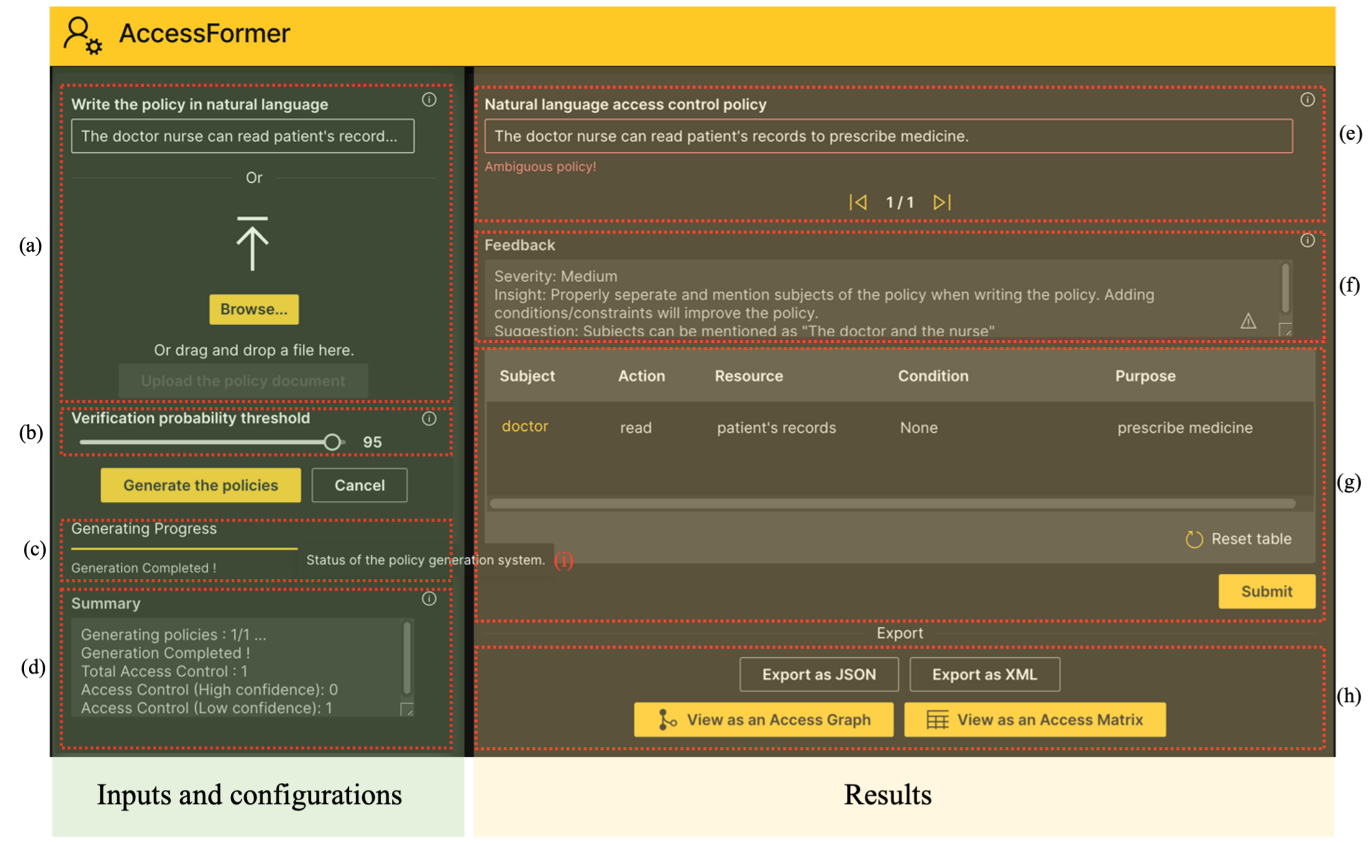
[69] J. Brooke, "Sus: a "quick and dirty'usability," *Usability evaluation in industry*, vol. 189, no. 3, pp. 189–194, 1996.

[70] A. Fruhling and S. Lee, "Assessing the reliability, validity and adaptability of pssuq," *AMCIS 2005 proceedings*, p. 378, 2005.

[71] [Online]. Available: https://wave.h2o.ai/

Appendix A
Policy Authoring Interface

The designed policy authoring interface according to the guidelines described in Section III-B can be shown as follows.



The doctor nurse can read patient's record...

Generation Completed !

Status of the policy generation system.

Generating policies : 1/1 ...
Generation Completed !
Total Access Control : 1
Access Control (High confidence): 0
Access Control (Low confidence): 1

The doctor nurse can read patient's records to prescribe medicine.

Ambiguous policy!

Severity: Medium
Insight: Properly seperate and mention subjects of the policy when writing the policy. Adding conditions/constraints will improve the policy.
Suggestion: Subjects can be mentioned as "The doctor and the nurse"

Subject	Action	Resource	Condition	Purpose
doctor	read	patient's records	None	prescribe medicine

Fig. 2: Proposed policy authoring interface created using H2O Wave Framework [71]. (a) High-level requirement specification document upload or manual policy input. (b) Configurable policy verification threshold that decides when to query the administrator. (c) Progress bar indicating the current policy generation progress. (d) System status that indicates what is happening behind the interface. (e) Poorly written access control policy with the error message in red. (f) Feedback from the system as insights on how to improve the NLACPs related to low-confident machine-executable policies. (g) Area to provide administrator's feedback on the incorrectly generated policies identified using the verification probability threshold. (h) Visualize and export all generated policies in different formats. (i) Tool-tip provided to describe the features.