2024 IEEE 6th International Conference on Trust, Privacy and Security in Intelligent Systems, and Applications (TPS-ISA) | 979-8-3503-8674-5/24/$31.00 ©2024 IEEE | DOI: 10.1109/TPS-ISA62245.2024.00042

SR2ACM: A Methodical Approach for Translating Natural Language Security Requirements to Access Control Model

Saja Alqurashi[1], Indrakshi Ray[1], Mahmoud Abdelgawad[1], and Hosein Shirazi[2]
[1]Department of Computer Science, Colorado State University, Fort Collins, Colorado, USA
[2]Management Information Systems, San Diego State University, San Diego, California, USA
[1]{saja.alqurashi, indrakshi.ray, m.abdelgawad}@colostate.edu, [2]hshirazi@sdsu.edu

***Abstract*—Access control policies (ACPs), embedded in the security requirements of an enterprise, are typically expressed in natural languages. Security administrators manually extract ACPs, interpret them, and construct a formal access control model which is later enforced by security mechanisms. The manual process is tedious, complex, expensive, labor-intensive, and error-prone. To address these limitations, we introduce a Natural Language Processing (NLP) pipeline called Security Requirements to Access Control Model (*SR2ACM*). This pipeline is designed to extract ACPs from statements in natural language automatically, convert these ACPs into the Next Generation Access Control (NGAC) model that we represent in the form of a graph, propose a set of properties that can be tested on the graph to assess the quality of the NGAC model so derived. SR2ACM performs downstream NLP tasks that include identifying ACP sentences, identifying NGAC relations, and identifying NGAC user and object attributes. The experimental results are promising as we achieved, on average, F1-score of 93% when identifying ACP sentences, F1-score of 97% when extracting NGAC relations between attributes, and F1-score of 96% when extracting user attributes and 89% for object attributes from natural language access control policies that we obtained from real-world applications. We utilize six diverse datasets representing access control policies of various domains to ensure a comprehensive evaluation. To assess the *correctness* of the NGAC policies generated, we propose a set of formal properties, namely, *completeness* (checks no ACPs have been omitted), *well-formedness* (checks the ACP construct), *minimality* (checks for absences of redundancies), and *consistency* (checks for absence of conflicts). Our formal analysis on our dataset reveals a completeness rate of 95% on average, a perfect rate of 100% of well-formedness, non-redundancy in 98% of NGAC ACPs, and an absence of inconsistencies. Our research is unique as it combines machine learning with software testing to assure the quality of the extracted model.**

***Index Terms*—Access Control Policies (ACPs), Next Generation Access Control (NGAC), Bidirectional Encoder Representations from Transformers (BERT), Graph Analysis**

I. Introduction

Organizations specify their security requirements, including Access Control Policies (ACPs), using natural language statements. ACPs define who is authorized for what resources and the conditions for such access. Formal access control models, such as Role-Based Access Control (RBAC) and Attribute-Based Access Control (ABAC) are then developed from these ACPs and implemented via security mechanisms [1], [2]. The conversion of access control requirements, articulated through natural language, into formal access control models, represents a task that is both labor-intensive and susceptible to inaccuracies. This challenge emerges from the requirement to precisely interpret both explicit and implicit access directives embedded within unstructured text descriptions. Consider, for example, the policy statement: “Access to project-specific documentation after standard operational hours is granted exclusively to individuals holding the position of project manager.” This necessitates the identification of ‘project managers’ as a distinct user role, ‘project-specific documentation’ as the designated resource, and ‘after standard operational hours’ as the conditional parameter for access. Policy statements may be more complex involving a multifaceted condition: “Employees, possessing a minimum of five years of tenure and affiliated with the Research and Development department, are authorized to retrieve confidential project data via secured terminals.” Herein, multiple attributes and constraints necessitate meticulous encoding into an access control model. These instances underscore the complexity and potential error margin inherent in the manual extraction of policy statements. Consequently, we need methodologies for the automated generation of access control models from security requirements and analyze the correctness of these models.

Most earlier research efforts [3]–[5] have investigated the development of RBAC [1] and XACML ABAC [2] from natural language policy statements. We focus on NIST Next Generation Access Control (NGAC) model [6] as it is more expressive than RBAC and it supports dynamic, event-triggered policies that can model the access control needs of most applications including situation-monitoring ones. NGAC model allows and denies access through *associations* and *prohibitions* respectively using *attributes* of users, and objects; it uses *obligations* to change the access control model; it has *assignment* relations for mapping users/objects to their respective attributes, and also mapping attribute hierarchy. Constructing an NGAC model from an ACP statement is complex because it involves extracting these relations and attributes, and the set of attributes used in a given application is not pre-defined.

Abdelgawad et al. [7] focused on developing the NGAC model by utilizing spaCy, an NLP library to extract entities and

979-8-3503-8674-5/24/$31.00 ©2024 IEEE
DOI 10.1109/TPS-ISA62245.2024.00042

relationships from ACP narratives. spaCy is good for extracting ACPs from sentences having a well-defined structure. However, ACPs, authored by humans, can be highly complex and often incorporate words such as "except," "unless," "prohibit," and "deny," which carry special connotations in access control contexts. Unfortunately, spaCy struggles to recognize these words as negations that indicate prohibitions within access control policies. Thus, more sophisticated techniques are needed for access control model extraction, such as, for those examples presented earlier.

Our approach offers a comprehensive method for deriving the NGAC model directly from natural language security requirements. We leverage advanced Natural Language Processing (NLP) techniques such as Bidirectional Encoder Representations from Transformers (BERT), distilling BERT (DistilBERT), Robustly Optimized BERT (RoBERTa), and eXtreme Language understanding NETwork (XLNET), to understand more complex sentence structure associated with real-world access control statements. The access control model so derived are then converted into a graph database Neo4j representation for analyzing its quality. Its quality is evaluated through properties that we propose.

Specifically, our proposed approach, Security Requirements to Access Control Model (SR2ACM), involves five tasks to structure and analyze ACPs adhering to the NGAC model. We first classify sentences that qualify as ACP using BERT and its variations. We then identify the types of NGAC relations within the ACP sentences by multi-label classification using BERT and its family models. Subsequently, we identify the NGAC entity types present in the ACP sentences using the NGAC-Entity Recognition (NGAC-ER) technique that we develop. After identifying the entities and relations, we generate a graph visually representing access control policies. Finally, the generated graph is evaluated against established criteria to ensure the derived policies' integrity and validity, concluding the NGAC model's formation.

To assess the effectiveness of our proposed method, we established metrics to evaluate the quality of the derived access control model alongside the performance of the machine learning algorithms employed. These metrics include:
(i) **Completeness.** This metric verifies that every ACP statement derived from security requirements is represented within the NGAC graph. Ensuring completeness is critical to confirm that no policy statement is omitted during translation.
(ii) **Well-Formedness.** This criterion evaluates the logical structure and validity of the ACP constructs generated, ensuring that the ACPs are correctly structured and adhere to the expected format and rules of the NGAC model.
(iii) **Minimality.** Minimality ensures the absence of redundancies. It guarantees that each user attribute to the object attribute path, providing the same access rights, is unique, eliminating future inconsistencies when some policy gets updated.
(iv) **Consistency.** This metric examines the model for any conflicting policies, such as instances where a user is both granted and denied access to the same resource, ensuring the consistent behavior of the access control model.
(v) **Machine Learning Accuracy.** In addition to the graph-specific metrics that evaluate the approach, the machine learning accuracy metric (F1-score) is utilized to measure the machine learning algorithms' performance in identifying and classifying ACP statements, relation types, and entities in these intermediate tasks. High accuracy indicates the effectiveness of the machine learning algorithms in correctly interpreting and processing the natural language ACPs.

To demonstrate our approach, we selected two critical domains – education and healthcare – as areas where clear policies are not only regulated at different levels (like organizations, state, or federal levels) but also rigorously enforced, necessitating precise and accurate access control models. To construct a robust dataset within these domains, we engaged in a comprehensive data collection process, gathering relevant access control policies and statements. Recognizing the imperative of high-quality annotated data for effective machine learning applications, we embarked on a manual annotation process, aimed to label the data and establish a gold-standard dataset that serves as a benchmark for training and evaluating our machine learning models. Acknowledging the challenges posed by the limited size of manually collected datasets, we leveraged the capabilities of the GPT-3 algorithm to augment our dataset.

The structure of this paper is summarized as follows: Section II provides a review of related work. Section III presents an overview of our proposed approach. Section IV elaborates on the structure and statistics of datasets. Section V provides details on ACP identification, NGAC relation and attributes identification. Sections VI, and VII elaborate the NGAC graph generation and evaluates the accuracy of the NGAC model derived respectively. Section VIII concludes the paper summarizing the contributions and directions for future research.

II. Literature Review

ACP Extraction from Natural Language Statements

Xiao et al.'s Text2Policy [4], Slankas et al.'s ACRE [8], and Narouei et al.'s method [3] represent foundational efforts in extracting ACP elements from natural language using linguistic analysis, dependency parse graphs, and Semantic Role Labeling (SRL), respectively. Despite achieving notable precision, these approaches are constrained by reliance on predefined patterns or limited in automatically learning new ACP patterns, often resulting in incomplete policy extraction. Alohaly et al. [5] develop a deep learning-based framework to automate ABAC attribute extraction. Utilizing NLP, relation extraction, and Convolution Neural Networks (CNNs), they achieve high F1-scores in extracting attribute values. However, the framework's accuracy is dependent on the Named Entity tagger's performance, which can lead to potential inaccuracies in attribute data type determination. Abu Jabal et al.'s *Polisma* framework [9] leverages data mining to learn ABAC policies from diverse sources, demonstrating the utility of machine learning in policy extraction. Xia et al. [10] utilize SRL integrated with BERT for identifying ABAC rule structures, yielding promising results. Despite this, the generalized architecture of SRL poses challenges in accurately labeling attributes related to security policies. Our proposed approach seeks to address these limitations by enhancing the precision and completeness of ACP extraction and analysis.

Access Control Policies Analysis

Different researchers studied the formal analysis of access control policies through the lens of three distinct approaches. Ray et al. [11] introduce a formal spatio-temporal RBAC model that integrates time and location dimensions, utilizing the Alloy analyzer for model verification. Fernández et al. [12] propose a structured method for specifying and analyzing ABAC policies using a category-based meta-model (CBAC) expressed in first-order logic. This approach provides a granular appraoch for policy specification and verification, focusing on ensuring policy correctness through rigorous formal analysis. Chen et al. [13] address the quality assurance of NGAC policies through mutation analysis, assessing the robustness of testing techniques against potential policy faults. By generating and testing policy mutants, this method aims to uncover faults in policy configurations and obligations. All the above approaches encounter limitations in scalability. These studies collectively underscore the importance of formal analysis in the evolution of access control models while highlighting significant obstacles in complexity, and scalability that our proposed approach aims to overcome.

III. Overview of Our Approach

NGAC model is structured around users, processes, objects, operations, and various attributes, all interconnected through specific relations like assignment, association, prohibition, and obligation. Assignment links users and objects to their respective attributes and policy classes, while association determines permitted access rights between user and object attributes. Prohibition negates permissions in contrast to association, and obligation reacts to specific events with predefined actions, playing a crucial role in establishing a dynamic security framework [6], [14].

We developed an NLP pipeline (namely *SR2ACM*) designed to automatically extract ACPs from statements in natural language and convert them into the NGAC model. Figure 1 illustrates the five tasks in our pipeline, described below, and outlines the architecture of our proposed approach. The first three tasks represented as downstream NLP tasks, include identifying ACP statements, discerning the types of relations within these statements, and capturing relevant entities. The last two tasks involve generating a graphical representation of the ACP and conducting an in-depth analysis of the generated graph to evaluate the integrity of the derived policies. This graph-based representation facilitates a more intuitive analysis and enables a rigorous assessment of the derived access control.

University Name	**Sentences**
Colorado State University	66
University of Colorado Denver	165
University of Colorado Boulder	35
University of Northern Colorado	350
Harvard University	165
Georgia State University	25
University of Arizona	36
New York University	139
Rochester Institute of Technology	27
University of California	26
University of Massachusetts	23
Tennessee State University	34
Total	1091

TABLE I: The Collected Security Sentences

IV. Data Collection and Preparation

Our supervised NLP algorithms for the initial three tasks in our pipeline require access to accurately labeled data. We identified five labeled datasets, namely, iTrust-v1 (DS1) [15], iTrust-v2 (DS2) [15], IBM (DS3) [8], CyberChair (DS4) [16], and Collected-ACPs (DS5) [8], in the domains of healthcare and education. However, most of the datasets have been labeled to determine whether a sentence constitutes an ACP and to identify the requisite entities for NGAC without annotating the NGAC relations. To address this gap, we extended our data collection efforts to include content from 12 US university websites, subsequently engaging in a meticulous manual annotation process to generate a ground-truth dataset tailored to our specific needs. We, thus, created a novel dataset from the educational domain, which we then enriched through manual annotation to secure reliable ground truth data. Table I shows the name and number of statements we have collected from each university.

A. Data Annotation

The annotation process encompassed consists of: (i) Labeling sentences as either *ACPs* or *non-ACPs* for Task 1, (ii) Identifying sentences with one or more NGAC relation types: *association* (AS), *assignment (AA)*, *prohibition (PR)*, or *obligation (OB)* for Task 2, (iii) Annotating the entities in ACP sentences as *user (U)*, *action (A)*, *object (O)*, *user attributes (UA)*, or *object attributes (OA)* for Task 3. The annotation criteria were based on three questions. Question (i) demanded binary yes/no responses to determine if the sentence is an ACP. Question (ii) required the annotators to select from among four potential NGAC relation types (*AS, AA, PR, OB*), and Question (iii) involved choosing from five NGAC entity types (*U, A, O, UA, OA*).

Our annotation team consisted of three Computer Science graduate students proficient in English and trained in NGAC. An expert in access control fields oversaw the operation as a supervisor. For each annotation task, the three annotators independently labeled the data. Then, the supervising expert

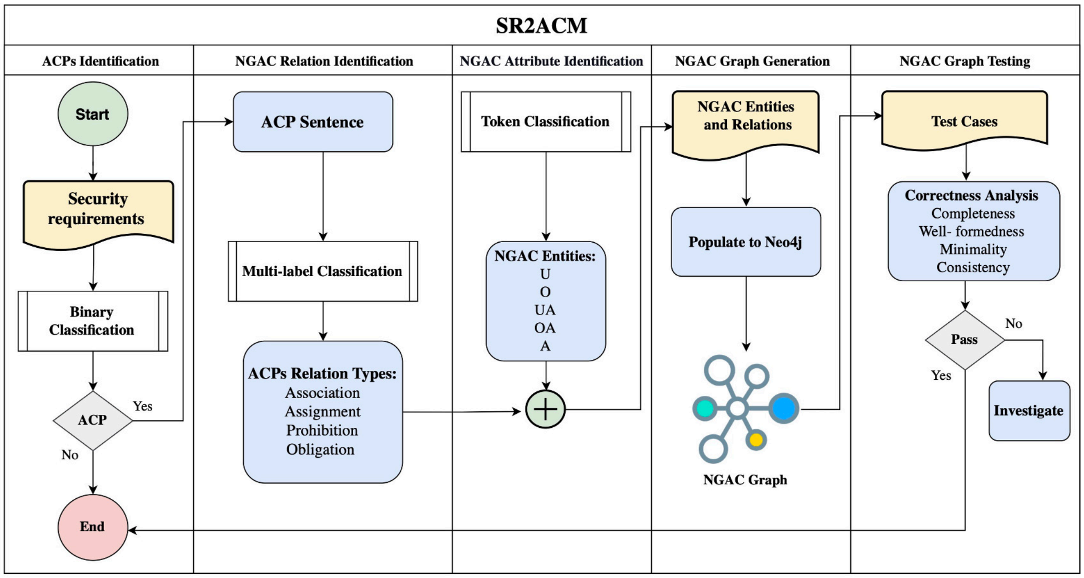

Fig. 1. A Cross-Functional Flowchart Presents SR2ACM Approach

reviewed their annotations, validated them, and resolved any discrepancies between the annotators. For the previously constructed online datasets, the first type of annotation (identifying ACPs) had been conducted by [3], [4], [16]. We manually annotated the relations and entities in these datasets.

B. Data Augmentation

Data augmentation mitigates model overfitting by generating modified versions of the existing data [17]. We employed a suite of augmentation techniques, notably back translation, integration of the datasets, and leveraging the capabilities of the GPT-3 Model Generator, to enrich our annotated data.

Back-Translation. We utilized the back translation to semantically retain the essence of the original ACP sentences while introducing syntactical variations. This was achieved through the development of a back translation script, utilizing the Google Translator API[1]. The back translation process involves three critical steps: (i) First, we performed a *Temporary Translation*, where ACP sentences were initially translated from English into a series of random languages sequentially, undergoing this translation cycle ten times to ensure a diverse linguistic transformation. (ii) Subsequently, a *Back Translation* step was implemented, wherein these transformed sentences were translated back to English. This process was iteratively conducted 100 times for each sentence across all datasets to maximize the generation of syntactically varied yet semantically consistent sentences. (iii) Finally, we conducted a *Duplicate Removal* phase, aimed at eliminating redundant back-translated sentences to maintain a unique set of augmented data. The following is an example of a back translation.

Original ACP: The user provides their MID and password to gain role-based access to the iTrust Medical Records system or request a password change.
Back Translation: The user provides his own MID and password to gain access based on his roles in the iTrust Medical Records system or to request a password change.

Data Integration. To accurately identify NGAC relations, a dataset having all four types of NGAC relations (*association*, *assignment*, *prohibition*, and *obligation*) is essential for model training. However, an examination of the available datasets, as detailed in Table II, reveals an uneven distribution of these relation types, with a notable scarcity of samples for *prohibition* and *obligation*. This imbalance stems from the prevalent inclusion of *association* and *assignment* relations in security requirements, where prohibitions are often implied, and obligations are explicitly mentioned under specific conditions. To address the shortfall in *prohibition* and *obligation* samples, we first integrated all datasets (D1-DS6). Subsequently, we employed data augmentation (GPT-3 Generator) to generate additional instances of these less common relation types. By duplicating each type twice, we ensured a more balanced representation of all NGAC relation types in our training dataset for the relation identification task outlined in Section V-B.

GPT-3 Generator. For both NGAC relations and entities identification tasks, we utilized the GPT-3 model, following a method akin to [18]. This method leverages GPT-3's capacity to produce synthetic yet hyper-realistic sentences, enhancing the diversity of our dataset. The augmentation process involved embedding a selection of ACP sentences from our training dataset into the GPT-3 prompt, thereby generating a variety of

[1] pypi.org/project/googletrans/

synthetic ACP sentences. For instance, the list below depicts five synthetic ACP sentences generated by GPT-3.

1) Only administrators, not data owners, make changes to the security label of a resource.
2) LHCP cannot cancel old appointments.
3) Inactive patients cannot be changed or logged into the system and can only be reactivated by the administrator.
4) The professor can change a student's grade at any time.
5) If a teacher is signed up to teach the course in the current stream, the system will not delete the course.

This technique resulted in the creation of a thousand new sentences, ranging from simple to complex structures.

C. Dataset Summary

Table II provides details of the five online datasets (DS1-DS5) in addition to our own one (DS6).

1) **DS1:** Sourced from iTrust [15], this dataset includes aspects like patients' medical histories, caregiver identification, communications with doctors, and sharing of results. Utilized in [4].
2) **DS2:** An expanded iteration of DS1 with over a thousand sentences, employed in [3].
3) **DS3:** Originating from IBM's course registration system, this dataset from the education sector is referenced in [8].
4) **DS4:** Cyberchair dataset, from the conference management domain, includes diverse conference and workshop policies [16].
5) **DS5:** A compilation of 114 ACP sentences from 18 different sources, including publications and websites, cited in [8].
6) **DS6:** Our collection comprises IT access control policies from U.S. universities.

V. Downstream NLP tasks

The NLP tasks to construct NGAC model include identifying ACP statements, distinguishing the types of relations embedded within these statements, and capturing pertinent entities.

A. ACP Identification

This task aims to extract the ACPs from security requirements. To do this task, we fine-tune the BERT and BERT family models to classify a sentence as ACPs or non-ACPs.

To leverage BERT and other BERT family models (including RoBERTa, DistilBERT, and XLNET), for a classification task, a new layer is appended atop the pre-trained model, specifically tailored for classification objectives. Initially, the text is segmented into constituent sentences, then is tokenized into a sequence of tokens. By inputting these structured tokens into BERT, the model generates an embedding vector for each token, typically of size 128. For the purpose of binary text classification, distinguishing between ACP statements and non-ACP statements, the embedding vector corresponding to a special token, called [CLS] token, which encapsulates the sentence's overall contextual representation is utilized. This vector serves as input to a newly added layer atop BERT, which employs a softmax function to classify the input into the relevant categories: ACP and non-ACP. In our experimental setup, we trained various models including BERT-base-cased, RoBERTa-base, DistilBERT-base-cased, DistilRoBERTa-base, and XLNET-base-cased. The training was carried out under specific hyperparameters: text sequence length set at 128, training epochs to 10, training batch size to 128, and evaluation batch size to 64. This method ensures precise, context-aware classification by effectively leveraging BERT's deep learning capabilities enhanced with a softmax-based classification layer.

Experimental Results.

For evaluation, we use the average of the F1-score across the resultant of binary classification that represents ACP or non-ACP in the testdata. Table III (ACP) shows the average F1-score in the resultant of the BERT text classification that represents ACP sentences in test datasets. We found that the model performance improved after data augmentation, proving the importance of data size and quality in NLP models.

Furthermore, this paper examines other BERT-based models, such as DistillBERT, RoBERTa, and XLNET. The goal is to compare the performance of different BERT models in the ACP identification task. The experimental results show the performance of various models on the test set of datasets. Overall, the XLNET shows significantly better performance in terms F1-score because XLNET operates in an autoregressive manner. In other words, it generates tokens sequentially while considering the preceding tokens' context. In contrast, BERT is a masked language model (MLM) in which random tokens are masked, and the model is assigned the task of predicting those masked tokens [19], [20]. Furthermore, our proposed ACP identification model overall outperforms the prior work Tex2Policy [4], ACRE [8], SENNA [3], and BERT based SRL [10] for ACP identification in terms of F1-score as shown in Table III. The results also indicate that the BERT-based methods, including our approach, performed better than Text2Policy [4], ACRE [8], SENNA [3]. In particular, the self-attention mechanism within transformers enables BERT models to focus on relevant parts of the input text, effectively capturing long-range dependencies and contextual information essential for understanding ACP sentences [20].

Our method achieves an average 93% F1-score, compared to BERT-based SRL [10] that achieves 85%. Xia et al. [10] utilizes SRL as a sub-module for ACP identification to extract the predicate-argument structure of a sentence.

B. NGAC Relations Identification

This task employs multi-label classification [21] to detect various NGAC relation types in an ACP, acknowledging that

Dataset		Data Size				Types			
Name	Ref.	Stmt	ACP	ACPs	Aug.	Association	Assignment	Prohibition	Obligation
DS1	[15]	471	419	89.0%	8572	408	269	6	0
DS2	[15]	1171	528	45.1%	15642	484	317	4	12
DS3	[8]	402	163	40.5%	4677	124	105	1	1
DS4	[16]	303	124	40.9%	4259	177	51	0	28
DS5	[8]	142	115	81.0%	2511	89	76	8	10
DS6	Our work	937	828	88.4%	13210	474	68	28	47
Total	-	**3426**	**2077**	**N/A**	**40871**	**1756**	**886**	**47**	**98**

TABLE II: Comprehensive analysis across datasets highlighting security statement counts with number and percentage of statements that contain access control policies, augmentation outcomes, and NGAC policy type distribution.

Experiment		**[4]**	**[8]**	**[3]**	**[10]**	**BERT**	**DistilBERT**	**RoBERTa**	**XLNET**
ACP									
	DS1	87%	98%	80%	95%	97%	97%	98%	98%
	DS2		88%	58%	82%	79%	74%	70%	87%
	DS3	97%	87%	89%	82%	98%	97%	99%	99%
	DS4		64%	77%	89%	85%	86%	86%	88%
	DS5		89%	78%	77%	96%	75%	95%	95%
	DS6					90%	89%	89%	89%
Relation Type									
	Association					97%	95%	95%	95%
	Assignment					86%	82%	83%	82%
	Prohibition					93%	80%	85%	85%
	Obligation					88%	81%	82%	87%
Attribute									
	User					70%	68%	70%	79%
	User Attribute					96%	91%	90%	91%
	Object					62%	58%	59%	54%
	Object Attribute					89%	51%	43%	72%
	Action					81%	81%	78%	57%

TABLE III: Comparative analysis of 3 tasks: ACP Identification, Relation Types Identification, and Attribute Identification conducted on 4 different algorithms of BERT, DistilBERT, RoBERTa, and XLNET. For the first task, ACP identification, we also report results from existing references with different algorithms on 5 manually annotated datasets in the literature (DS1 to DS5) and our manually annotated dataset (DS6).

an ACP may encapsulate multiple relation types simultaneously. Utilizing the BERT base model and other BERT family models (including RoBERTa, DistilBERT, and XLNET) for thier efficacy in generating contextual vector representations, we extend it with a linear layer comprising four outputs to correspond with the NGAC relation types. This approach allows for the nuanced differentiation between association, assignment, prohibition, and obligation, addressing the challenge of multi-label classification. To enhance the model's performance, particularly for the underrepresented classes of prohibition and obligation, we augment our dataset using the GPT-3 model generator [18], ensuring a balanced and comprehensive training dataset. In training the models, we maintained consistent hyperparameters: a text sequence length of 128, 10 training epochs, a train batch size of 64, and an evaluation batch size of 66.

Experimental Results. We use F1-score across the resultant of multi-label classification that represents NGAC relations types in our test data. Overall BERT shows significantly better performance in terms of F1-score. Our findings are presented in Table III (Relation Type). We achieved a 97% F1-score, for identifying association relation types. Regarding the types of assignment relation, our results yielded an 86% F1-score. For prohibition relation types, we achieved a 93% F1-score. Lastly, for obligation relation types, we obtained an 88% F1-score. Our experimental results demonstrate the performance with an overall average of accuracy 91% on the test set of ACP sentences containing different types of NGAC relations. To the best of our knowledge, no previous work has investigated how to automatically identify all the relation types in NGAC.

C. *NGAC Entities Identification*

We developed the NGAC Entity Recognition (NGAC-ER) model, designed to tackle the sequence labeling classification challenge [22] in identifying NGAC entities in ACP sentences. Utilizing the BERT base model and other BERT family models (including RoBERTa, DistilBERT, and XLNET), we fine-tune them to accurately categorize each token as User (U), User Attributes (UA), Object (O), Object Attributes (OA), and Action (A), enabling precise entity recognition in ACP contexts. For this task, we used the BERT models, optimized with a text sequence length of 75, 10 training epochs, and batch sizes set to 32 for both training and evaluation phases. Addressing the challenge of diverse sentence structures in NGAC-ER, we expanded our training dataset through data augmentation, leveraging the GPT-3 model to generate an additional 1000 unique ACP sentences. This augmentation significantly enhances the model's training, enabling more robust and nuanced entity recognition capabilities.

Experimental Results. We use F1-score for the token classification experiments that identify NGAC entities. NGAC-ER model shows promising results for predicting each token in an ACP sentence. Overall BERT shows significantly better performance for (UA) and (OA) which achieved 96% and 89% F1-score respectively, while (U) and (O) achieved 70% and 62% F1-score respectively as demonstrated in Table III. Our experimental results demonstrate that BERT significantly outperforms other models in terms of F1-score. Specifically, it achieves an overall average accuracy of 93% on the test set of ACP sentences containing various types of NGAC entities. However, predicting Object (O) and User (U) is not high enough because of the nature of security requirements documents written in general without specifying a user such as John or an object such as DeviceX. However, adding lists of subjects and objects as input documents with the model outputs will resolve the low performance, and we will have all the attributes to generate the NGAC model.

NGAC-ER model significantly outperforms that proposed in [5] by obtaining higher F1-score for user attributes and object attributes identification as shown in Table IV. Moreover, the NGAC-ER model outperforms the model in [23] that aims to identify actor, action, and resource in user stories by obtaining 87% accuracy while the NGAC-ER model obtains 93% accuracy. BERT transformer employs attention mechanisms, enabling it to focus on different segments of the input sequence during prediction. This attention mechanism allows the model to weigh the importance of different words in the input sequence, providing an understanding of the relationships between words [20]. This contributes to the superior performance of the BERT transformer model over deep learning approaches in the task of extracting attributes in ACPs.

VI. NGAC Model Generation

The NGAC model generation comprises two primary steps: (i) Constructing the NGAC graph to formulate the model, and (ii) Graph testing to evaluate and analyze the resultant model.

We use Neo4j [24], [25] to generate the NGAC graph from the entities and relations extracted using the models outlined in Sections V-A, V-B, and V-C. Utilizing Neo4j, we populate the extracted NGAC entities and relations and represent them in the form of an NGAC graph. Nodes correspond to users, objects, user attributes, and object attributes. Edges correspond to assignment relations and actions. Assignment edges are added between users and user attributes as well as between object attributes and objects. Action edges are from user attributes to object attributes. The current representation of NGAC model does not represent assignment relations between pairs of object attributes or pairs of user attributes. Figure 2 illustrates an instance of a Neo4j graph constructed from a subset of policies in dataset DS6. As shown in Table V, we have a director of network services and an access control administrator who approves the devices that connect to the university network. At the same, the access control administrator maintains the logs for access requests. Additionally, students are allowed to use their files while they are not permitted to use monitoring software. To demonstrate users accessing objects, we present four policy sentences (1-4) from the dataset DS6, shown in Table V. Sentences 5-7 assign three users (User.X, User.Y, User.Z) as a student, access control administrator, and a director of network services. We added four sentences (8-11) to demonstrate objects (File.X, Log.X, SW.X, and Device.X) that are assigned to object attributes (file, access request (AR) log, monitoring software (M.SW), and connected network (Conn.N) device).

VII. NGAC Graph Testing

We evaluate the correctness of our approach through graph-based testing, focusing on four main properties: *completeness*, *well-formedness*, *minimality*, and *consistency*. We generate paths that cover each user assigned to user attributes associated with (or prohibited from) access rights to object attributes assigned to objects which are used to form test cases. We use two datasets to demonstrate our approach. One in the education domain (DS6) and the other in the healthcare domain (DS1).

The testing procedure illustrated in Figure 3 is initiated by loading the paths derived from the NGAC graph for the ACPs. Subsequently, a test scripter transforms these paths into Python code test cases. In each path, the test scripter retrieves the user from the source node, the access right from an intermediary node, and the object from the sink node. These elements are then translated into Regular Expressions.

We use *Neo4j* Cypher path matching, using *MATCH* statement. Therefore, we can generate paths from every U user node to every O object node.

```
MATCH paths=(u:U)-[:Assignment| Prohibition|
Association * 1..*]-(o:O)
RETURN distinct(paths) ORDER BY length(paths);
```

This *MATCH* statement generates a set of paths that covers the NGAC graph. These paths are a sequence of nodes and relations representing how a user node U reaches an object node O through $assignment$, $association$, and $prohibtion$ relations. We use these paths to script test cases that will be executed against each ACP sentence's content. For instance, when the test cases are executed against the ACP 1 (Student should use only their assigned file to access the information system), at least one test case has to pass successfully.

This test case matches the word (student) as a user attribute node, (use) as an access right node (intermediate node), and (file) as an object attribute node. If all test cases fail to match these three phrases, this ACP sentence has not been extracted. The scripter will fetch the (student), (use), and (file) words and then form a regular expression as follows: `.*student.* (use).*file .*`. The dot refers to any character; the star means zero or many. As a result, this regular expression verifies that from the beginning of the sentence, ignore a series of characters until reaching the (student) word, ignore another series of characters until reaching (use), and another string of characters until getting the word (file). Similarly,

Access Control Policy Attribute	Alohaly [5]	NGAC-ER
User Attributes	85%	96%
Object Attribute	71%	89%

TABLE IV: Attribute Identification Performance Compared with Other Work

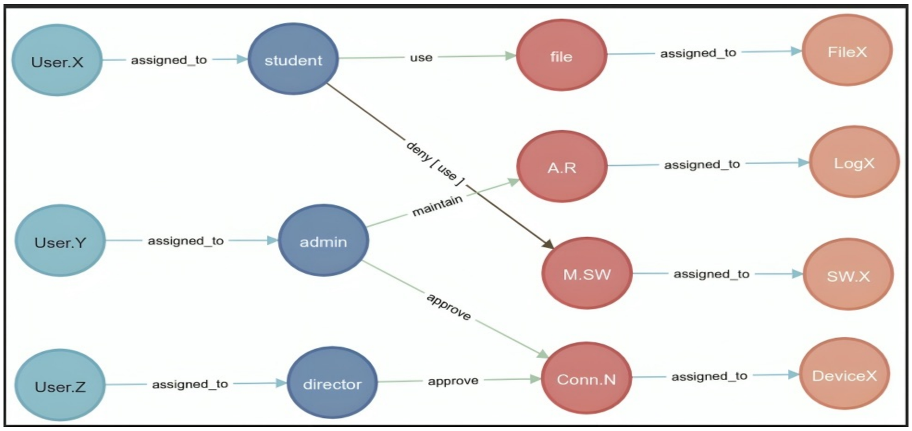

Fig. 2. The Generated NGAC Graph Model

ACPs	
1	Student should use only their assigned file to access the information system.
2	Student is prohibited to use monitoring software in the university network.
3	The director of network services and access control administrator must approve the connected network devices.
4	The access control administrator will maintain all access request logs for creations, modifications, and deletions.
User	
5	UserX is a student.
6	UserY is an access control administrator.
7	UserZ is a director of network services.
Object	
8	File.X is an object.
9	Log.X is an object.
10	SW.X is an object.
11	Device.X is an object.

TABLE V: A Set of ACPs and Users and Objects

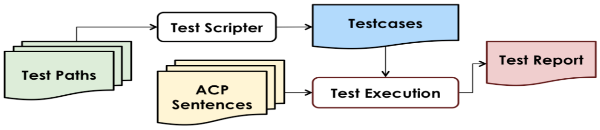

Fig. 3. Testing Process

the scripter converts all paths into regular expressions. Each regular expression is passed as an argument to a match function (*search(pattern, ACP_Sentence)*) that searches for a match over a string variable that represents an ACP sentence. The case sensitivity of letters is ignored. The result of this function is asserted to be not equal to *None*. The match function and the assertion are wrapped up in a test case defined with a unique name. The test cases generated are then encased into a test-suite. The following is a *Python* snippet code that shows one of the test cases resulting from the scripter when the path from Table VI is passed into it.

```
class TestSuite(unittest.TestCase):
 ACP_Sentence = ''
 def testcase01(self):
   pattern ='.*student.*use.*file .*'
   found = re.search(pattern, self.ACP_Sentence + '|$',
   flags=re.IGNORECASE)
    self.assertNotEqual(found, None)
```

The test execution loads the ACP sentences. It then passes each ACP sentence to the test-suite, executes it, and reports the test result into a file. Subsequent paragraphs illustrate how we use graph testing to validate the correctness of the real-world dataset in the education and healthcare domains. To ensure the robustness of our approach, we meticulously assessed its correctness through the lenses of *completeness*, *well-formedness*, *minimality*, and *consistency*.

Completeness ensures that the generated NGAC graph covers all the ACP sentences. We checked the completeness by testing the paths generated from the NGAC graph against the ACP sentences. Table VII shows the test result when this test-suite is executed against the ACP sentence (Student should use only their assigned file to access the information system.) It reports that the first test case passed successfully while the others failed. The test result is analyzed based on the following test

Path: Sequence of nodes and relations
[{"name":"User.X"},{"assign":"assigned_to"},{"name":"student"},{"name":"student"}, {"ars":"[use]"}, {"name":"use"},{"name":"name":"file"},{"assign":"assigned_to"}, {"name":"File.X"}]

TABLE VI: A Path that Generated from the NGAC Graph Model

********** ACP ************************
Testing: Student should use only their assigned file to access the information system.
testcase01 (ACP_TestSuite.TestSuite) ... ok
testcase02 (ACP_TestSuite.TestSuite) ... FAIL
testcase03 (ACP_TestSuite.TestSuite) ... FAIL

TABLE VII: Result of Test Suite Executed against ACP1

criterion: *"for each ACP sentence, at least one testcase should pass"* Thus, this ACP sentence has been extracted as NGAC policy.

Well-formedness ensures the inclusion of all NGAC entities in each ACP sentence. It starts with involving the user node and concludes with the object node, where the edge connecting the user attribute, and object attribute signifies either an association or prohibition relation. To verify well-formedness, distinct paths for each ACP sentence were generated, ensuring that the path initiated from the user U and concluded with the object O and that the relation between UA and OA was either association or prohibition. The generation of distinct paths was facilitated using the `DISTINCT` keyword in *Neo4j* Cypher. Well-formedness is verified by confirming the presence of user (U), user attribute (UA), access rights (ars), object attribute (OA), and object (O) in the path using the following two regular expressions that verify whether a path is formed as a user assigned to a user attribute that is associated with access rights or prohibited from access rights to an object attribute assigned to an object:

```
{.*user.*user-attribute.*ars.*object-attribute.*object.*}
{.*user.*user-attribute.*deny.*object-attribute.*object.*}
```

Minimality or non-redundancy refers to the fact that an access right given to a user attribute to access an object attribute should not exist for two different paths in the NGAC graph. We checked the minimality by generating two groups of paths; one group has all paths of ACPs where the path starts from user attributes (UA) to object attributes (OA) without distinguishing them. The other group involves distinct paths. We used the keyword `DISTINCT` in the *Neo4j* Cypher to generate the distinct paths. The difference between these two groups of paths results in redundant paths, which reflect ACP sentences' redundancy. The paths of all ACPs are retrieved through the following *MATCH* statement:

```
MATCH allPaths=(user:U)-[:Assignment|Prohibition|
Association * 1..n]-(object:O)
WHERE none(node in nodes(allPaths) WHERE node:PC)
RETURN allPaths ORDER BY length(allPaths);
```

The distinct ACP paths are obtained by the following *MATCH* statement that is written as follows:

```
MATCH dsPaths=(user:U)-[:Assignment|Prohibition|
Association * 1..n]-(object:O)
WHERE none(node in nodes(dsPaths) WHERE node:PC)
RETURN distinct dsPaths ORDER BY length(dsPaths);
```

Consistency refers if no two ACPs conflict with each other. We checked the consistency by detecting the *paths-conflict* that occurs between association relation and prohibition relation, giving an access right to a user attribute to access an object in one place and denying it from accessing the same object in another. After we generate paths for ACPs, we separate the paths into two groups; one group carries association relations paths, and the other group consists of prohibition relations. We then checked the consistency by exercising that for every user attribute, matching the access rights of paths with association relations to the denied access rights of paths with prohibition relations to which the same user attribute and object attribute are assigned. We used *regular expression* for matching values of the access rights and denied access rights. To ensure the verification process performs correctly, we added two ACP sentences to each dataset that causes *paths-conflict*, *(Student is prohibited to use monitoring software in the university network.)* and *(Student can use monitoring software in the university network)*. The verification process detected this *paths-conflict* successfully. Removing this *paths-conflict* and rerunning the verification process over the generated NGAC models, we found no *paths-conflict* that may lead to inconsistency between the NGAC ACPs.

We analyzed policies from the healthcare and education domains. For education domain policies, the level of *completeness* averages at 95%, 46 out of 48 test cases were successful. *Well-formedness* level averages at 100%. The analysis also indicates that 98% of NGAC ACPs are non-redundant. The remaining 2% of NGAC ACPs are redundant because many policies in educational institutions have applicability across diverse contexts. For example, when a university defines conditions for an event, the associated policy is expected to be enforced, which might also be relevant to other scenarios. Moreover, based on our analysis, there is no evidence of conflict between any two policies.

From healthcare domain policies, the *completeness* level averages 97%, with 83 out of 86 test cases passing successfully. Our analysis indicates a *well-formedness* level averaging at 100%. Additionally, our assessment highlights that 31% of NGAC ACPs are redundant. This redundancy occurs due to repeated role-specific policies, highlighting shortcomings in policy specification. Furthermore, our analysis did not uncover any evidence of *conflicts* between policies. No path conflicts were identified that could potentially lead to inconsistencies within the NGAC model.

SR2ACM demonstrates its effectiveness in extracting the formal NGAC model compared to machine learning methods like [3], [4], [8], and deep learning approaches such as [5].

The graph testing method is unique compared to existing approaches [11]–[13].

VIII. Conclusion

Security requirements are often written in natural language. Extracting access control models from these requirements is non-trivial. Towards this end, we developed the *SR2ACM* that automatically extracts ACPs from statements in natural language and converts these ACPs into a formal NGAC model expressed as a graph. The correctness of the extraction NLP pipeline has been ensured by validating four properties: *completeness*, *well-formedness*, *minimality*, and *consistency*. The results indicate that the *SR2ACM* shows potential in efficiently extracting the NGAC access control models from available security requirements. Future research will explore how we can adapt *SR2ACM* for efficiently handling updates to security requirements. Finally, we will investigate how we can develop formal privacy models from privacy documents such as GDPR using this approach.

Acknowledgements

This work was supported in part by funding from NIST under Award Number 60NANB23D152 and NSF under Award Numbers DMS 2123761, CNS 1822118, CNS 2335687, ARL, Statnett, AMI, NewPush, and Cyber Risk Research.

References

[1] R. S. Sandhu, "Role-based access control," in *Advances in computers*. Elsevier, 1998, vol. 46, pp. 237–286.

[2] V. C. Hu, D. R. Kuhn, D. F. Ferraiolo, and J. Voas, "Attribute-based access control," *Computer*, vol. 48, no. 2, pp. 85–88, 2015.

[3] 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.

[4] 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*. Cary, NC, USA: ACM, 2012, pp. 1–11.

[5] 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.

[6] D. F. Ferraiolo, S. I. Gavrila, and W. Jansen, "Policy machine: features, architecture, and specification," *NIST Interagency/Internal Report (NISTIR)*, vol. 1, no. 2, pp. 100–120, 2015.

[7] M. Abdelgawad, I. Ray, S. Alqurashi, V. Venkatesha, and H. Shirazi, "Synthesizing and analyzing attribute-based access control model generated from natural language policy statements," in *Proceedings of the 28th ACM Symposium on Access Control Models and Technologies*. Trento,Italy: ACM, 2023, pp. 91–98.

[8] 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*. New Orleans, LA, USA: ACM, 2014, pp. 366–375.

[9] A. Abu Jabal, E. Bertino, J. Lobo, M. Law, A. Russo, S. Calo, and D. Verma, "Polisma a framework for learning attribute-based access control policies," in *Proceedings of 25th European Symposium on Research in Computer Security*. Guildford, UK: Springer, 2020, pp. 523–544.

[10] 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 *Proceedings of the IEEE International Conference on Bioinformatics and Biomedicine*. Las Vegas, NV, USA: IEEE, 2022, pp. 1289–1296.

[11] I. Ray and M. Toahchoodee, "A spatio-temporal role-based access control model," in *IFIP Annual Conference on Data and Applications Security and Privacy*. Berlin Heidelberg: Springer, 2007, pp. 211–226.

[12] M. Fernández, I. Mackie, and B. Thuraisingham, "Specification and analysis of ABAC policies via the category-based metamodel," in *Proceedings of the 9th ACM conference on data and application security and privacy*. Richardson, TX, USA: Association for Computing Machinery, 2019, pp. 173–184.

[13] E. Chen, V. Dubrovenski, and D. Xu, "Mutation analysis of NGAC policies," in *Proceedings of the 26th ACM Symposium on Access Control Models and Technologies*. Virtual Event, Spain: Association for Computing Machinery, 2021, p. 71–82.

[14] D. Ferraiolo, R. Chandramouli, R. Kuhn, and V. Hu, "Extensible access control markup language (XACML) and next generation access control (NGAC)," in *Proceedings of the ACM International Workshop on Attribute Based Access Control*. New Orleans,: ACM, 2016, pp. 13–24.

[15] A. Meneely, B. Smith, and L. Williams, "Appendix b: itrust electronic health care system case study," *Software and Systems Traceability*, vol. 1, no. 3, pp. 409–425, 2012.

[16] R. Van De Stadt, "Cyberchair: A web-based groupware application to facilitate the paper reviewing process," *CoRR*, vol. abs/1206.1833, p. 7, 2012. [Online]. Available: http://arxiv.org/abs/1206.1833

[17] S. C. Wong, A. Gatt, V. Stamatescu, and M. D. McDonnell, "Understanding data augmentation for classification: when to warp?" in *Proceedings of the IEEE International Conference on Digital Image Computing: Techniques and Applications*. Gold Coast, Queensland: IEEE, 2016, pp. 1–6.

[18] K. M. Yoo, D. Park, J. Kang, S.-W. Lee, and W. Park, "GPT3Mix: Leveraging large-scale language models for text augmentation," in *Findings of the Association for Computational Linguistics*. Punta Cana, Dominican Republic: Association for Computational Linguistics, 2021, pp. 2225–2239.

[19] P. Liu, W. Yuan, J. Fu, Z. Jiang, H. Hayashi, and G. Neubig, "Pre-train, prompt, and predict: A systematic survey of prompting methods in natural language processing," *ACM Computing Surveys*, vol. 55, no. 9, pp. 1–35, 2023.

[20] J. D. M.-W. C. Kenton and L. K. Toutanova, "BERT: Pre-training of deep bidirectional transformers for language understanding," in *Proceedings of NAACL-HLT*. Minneapolis, Minnesota: Association for Computational Linguistics, 2019, p. 4171–4186.

[21] A. C. de Carvalho and A. A. Freitas, "A tutorial on multi-label classification techniques," *Foundations of Computational Intelligence*, vol. 5, pp. 177–195, 2009.

[22] M. Ehrmann, A. Hamdi, E. L. Pontes, M. Romanello, and A. Doucet, "Named entity recognition and classification in historical documents: A survey," *ACM Computing Surveys*, vol. 56, no. 2, pp. 1–47, 2023.

[23] J. Heaps, R. Krishnan, Y. Huang, J. Niu, and R. Sandhu, "Access control policy generation from user stories using machine learning," in *Proceedings of Data and Applications Security and Privacy*. Calgary, Canada: Springer, 2021, pp. 171–188.

[24] J. Webber, "A programmatic introduction to neo4j," in *Proceedings of the 3rd annual conference on Systems, programming, and applications: software for humanity*. Arizona, USA: ACM, 2012, pp. 217–218.

[25] D. Fernandes, J. Bernardino *et al.*, "Graph databases comparison: AllegroGraph, ArangoDB, InfiniteGraph, Neo4J, and OrientDB," in *Proceedings of DATA*. Portugal: ACM, 2018, pp. 373–380.