Toward Deep Learning Based Access Control

Mohammad Nur Nobi
Institute for Cyber Security (ICS) and Department of Computer Science
University of Texas at San Antonio
Texas, USA
mohammadnur.nobi@utsa.edu

Ram Krishnan
ICS, NSF Center for Security and Privacy Enhanced Cloud Computing (C-SPECC), and Department of Electrical and Computer Engineering
University of Texas at San Antonio
Texas, USA
ram.krishnan@utsa.edu

Yufei Huang
Department of Medicine, University of Pittsburgh, and UPMC Hillman Cancer Center
Pennsylvania, USA
yuh119@pitt.edu

Mehrnoosh Shakarami
ICS, C-SPECC, and Department of Computer Science
University of Texas at San Antonio
Texas, USA
mehrnoosh.shakarami@my.utsa.edu

Ravi Sandhu
ICS, C-SPECC, and Department of Computer Science
University of Texas at San Antonio
Texas, USA
ravi.sandhu@utsa.edu

ABSTRACT

A common trait of current access control approaches is the challenging need to engineer abstract and intuitive access control models. This entails designing access control information in the form of roles (RBAC), attributes (ABAC), or relationships (ReBAC) as the case may be, and subsequently, designing access control rules. This framework has its benefits but has significant limitations in the context of modern systems that are dynamic, complex, and large-scale, due to which it is difficult to maintain an accurate access control state in the system for a human administrator. This paper proposes Deep Learning Based Access Control (DLBAC) by leveraging significant advances in deep learning technology as a potential solution to this problem. We envision that DLBAC could complement and, in the long-term, has the potential to even replace, classical access control models with a neural network that reduces the burden of access control model engineering and updates. Without loss of generality, we conduct a thorough investigation of a candidate DLBAC model, called DLBAC_{α}, using both real-world and synthetic datasets. We demonstrate the feasibility of the proposed approach by addressing issues related to accuracy, generalization, and explainability. We also discuss challenges and future research directions.

CCS CONCEPTS

• **Security and privacy → Access control**; • **Computing methodologies → Machine learning**.

KEYWORDS

Access control; Deep learning; Automation

CODASPY '22, April 24–27, 2022, Baltimore, MD, USA.

ACM ISBN 978-1-4503-9220-4/22/04…$15.00
https://doi.org/10.1145/3508398.3511497

ACM Reference Format:
Mohammad Nur Nobi, Ram Krishnan, Yufei Huang, Mehrnoosh Shakarami, and Ravi Sandhu. 2022. Toward Deep Learning Based Access Control. In *Proceedings of the Twelveth ACM Conference on Data and Application Security and Privacy (CODASPY '22), April 24–27, 2022, Baltimore, MD, USA.* ACM, New York, NY, USA, 12 pages. https://doi.org/10.1145/3508398.3511497

1 INTRODUCTION AND MOTIVATION

Access Control Lists (ACLs) [28], Role-Based Access Control (RBAC) [60], and Attribute Based Access Control (ABAC) [32] are some of the mainstream approaches to determine users' access to resources. Commercial solutions [21] that cater to organizations employ one or more of these classical access control functionalities. While tremendous progress has been made in the realm of classical access control approaches [41], one fundamental issue has remained the same for over forty years. Skilled security administrators needed to engineer and manage accesses as only humans could develop detailed policy insights about individuals' needs within the broader organization. Clearly, this leads to all types of errors and inefficiencies [4]: there remain plenty of users with accesses that should not have those accesses (over-provisioned to ease administrative burden) and plenty of users that lack accesses that should indeed have those accesses (under-provisioned for the sake of tightened security) [22, 65]. Administrators tactfully perform a balancing act to maximize security and minimize costs. This complexity is further exacerbated with the proliferation of cloud-based applications that perform machine-to-machine access through APIs, IoT, BYOD, etc.

In this paper, we propose an automated and dynamic access control mechanism leveraging advances in deep learning technology [62] that could complement or potentially replace the human administrator. This approach, denoted as Deep Learning Based Access Control (DLBAC), addresses three major limitations of classical access control approaches such as RBAC and ABAC. Without loss of generality, we use the term attribute to refer to any form of traditional access control information such as roles and relationships.

1. Attribute Engineering. An organization typically holds a vast number of metadata about its users and resources. However, those metadata are often not meaningful *access control attributes*.

As a first step, using organizational context often inferred from those metadata, administrators engineer access control specific attributes that could be used to express access control rules subsequently. This is at best an art today involving semi-formal design and requirements engineering processes [58].

2. Policy Engineering. After access control relevant attributes are engineered, administrators need to engineer access control rules. This is accomplished through either a manual engineering process akin to attribute engineering above or automated mining techniques that take as input a primitive form of access rules such as ACLs and generate approximate ABAC rules (or user-role assignments, in the case of RBAC) [19, 44]. We will show that, for complex situations, DLBAC generally captures the access control state of the system with more precision than other approaches which are based on policy mining and classical machine learning.

3. Generalization. Most prior approaches [11, 54] that mine access control rules from simpler forms of access control states such as ACLs focus on accurately capturing the access control state as given in those ACLs. Unfortunately, that leads to poor generalization [17, 73]—that is, the ability to make better access control decisions on users and resources with attributes that were not explicitly seen during the mining process. However, this is something machine learning methods, especially deep learning, are better at. They have an innate ability to make quality predictions as long as the test sample at prediction time aligns with the training data *distribution*. We will show that the engineered rules typically make poor access control decisions for user-resource metadata that were not explicitly seen by the mining process.

DLBAC addresses the above issues by exploring a fundamentally different approach to how access control is designed today. As illustrated in Figure 1, DLBAC differs from classical approaches by making decisions based on the *metadata* of users and resources and a trained neural network. (The key distinction between the notions of *metadata* and *attributes* albeit semantic has important practical benefits, which is explained in section 2.1.) It accomplishes this (see Figure 2) by first replacing access control policies with a neural network that instead makes access control decisions. Second, the neural network is trained using raw metadata from the organization instead of laboriously engineered access control attributes.

To summarize, we make the following contributions to the field of access control:

- We propose DLBAC, a new approach of automated and dynamic next generation access control.
- We develop a candidate DLBAC model, DLBAC_α, which outperforms classical policy mining and machine learning techniques in many aspects, including capturing the existing access control state of the system accurately and generalizing well to situations that were not seen during training time.
- As DLBAC is a neural network, we address previously highlighted concerns on the explainability of the black-box nature of neural network-based systems for access control [12]. We apply deep learning interpretation methods to confirm that decision-making in DLBAC can indeed be understood to a large degree (albeit not with 100% accuracy).

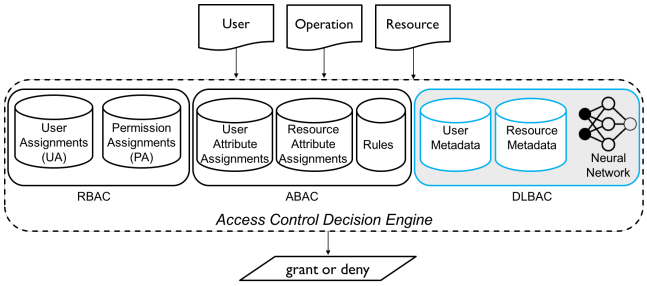

Figure 1: Decision Making in Classical Approaches vs. DLBAC.

- We synthesize several large-scale access control datasets with a varying number of users and resources. We evaluate the performance of DLBAC on those synthetic datasets along with two real-world datasets.

The rest of the paper is organized as follows. Section 2 presents an overview of the DLBAC approach. We discuss related work in Section 3. Section 4 introduces some real-world datasets and presents the synthetic data generation method for DLBAC_α, a candidate DLBAC model. In the same section, we also explain the implementation of DLBAC_α. We conduct performance evaluation of DLBAC_α in Section 5. We present approaches to understand DLBAC_α decisions in Section 6. In Section 7, we discuss future research directions, and conclude in Section 8.

2 DEEP LEARNING BASED ACCESS CONTROL

In this section, we provide a brief overview of DLBAC and explain how it differs from classical approaches.

2.1 Decision Making in Classical Approaches vs. DLBAC

Figure 1 illustrates how DLBAC makes a decision as compared to two classical access control approaches (the discussion applies to other forms of access control approaches such as relationship based access control or ReBAC [14]). In RBAC, an access control decision is simply a cross-reference between user-role and permission-role assignment relations. In the case of ABAC, an access control rule is evaluated for a given operation based on the attributes of the user and resource in question (sometimes attributes of other entities such as "environment" are used as well). In DLBAC, a deep neural network makes an access control decision based on the available *metadata* for the user and resource. For example, metadata could include logs of accesses, employee join date, access time, network access profile, etc. For simplicity, we assume metadata are represented as name-value pairs. While syntactically they appear to be the same as attributes, which are often name-value pairs as well, semantically they are very different. Metadata are primarily different from attributes since they do not go through the access control design and engineering process. A typical organization could host multiple applications such as email, file storage, human resources, benefits, and other cloud services. Each of those applications hold metadata about users and resources in the organization. Metadata are designed inherently as part of the functionality engineering phase of the system instead of during the access control design

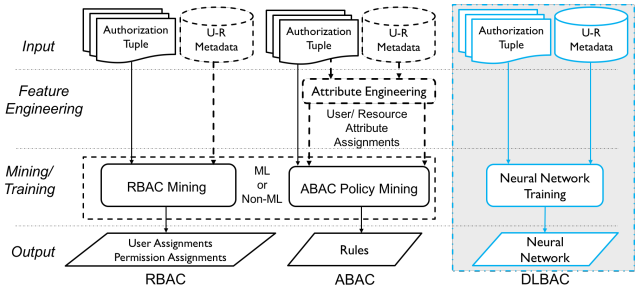

Figure 2: Design Process of Classical Approaches vs. DLBAC.

phase of the system. Therefore, they are immediately available to DLBAC once the system is implemented. For example, 'join_date', 'spending_history' and 'credit_history' could be metadata of customer, whereas an engineered attribute could be 'status' (such as 'status = platinum'), determined based on all of those metadata.

2.2 Policy Engineering in Classical Approaches vs. DLBAC

A conceptual representation of classical access control approaches versus DLBAC is depicted in Figure 2. For simplicity, we assume all methods obtain the current access control state of the system as authorization tuples (e.g., ⟨user, resource, operations⟩), and the metadata of users (e.g., ⟨designation, "employee"⟩) and resources (e.g., ⟨size, "small"⟩) as the input. In ABAC, the first step of *attribute engineering* involves designing users and resources attributes in the system that are selected and properly assigned based on available metadata. Common to ABAC and RBAC, the second stage is *policy mining/engineering*, through which proper policies are developed. Access control mining algorithms, including those using machine learning (ML), are summarized in Section 3. The last element in the conceptual representation of models is the output. For the RBAC approach, the mining process's output is a set of roles, permission assignment to roles (PA), and user assignment to roles (UA). For ABAC, the output includes a policy consisting of a set of access control rules. (Note that most ABAC mining works assume that attributes and attribute assignments are already available [54, 59]). In contrast, DLBAC is an end-to-end access control approach. It does not need attribute engineering since it works directly with the users and resources metadata. The DLBAC approach's output is a trained neural network, which takes user and resource metadata as input and makes access control decisions. We note that the DLBAC is agnostic to any deep neural network architecture.

3 RELATED WORK

There is plenty of work on mining/engineering policies either by analyzing the current access control state in the forms of logs and ACLs or transforming one access control model to another. We review three classes of related work below.

Classical Policy Mining Approaches. A rich body of research on mining classical access control models includes RBAC, ABAC, and ReBAC. Many algorithms were proposed to mine RBAC policies [24, 72], following either top-down [19, 49, 51, 69] or bottom-up [61] methods, while hybrid methods [23] combine the advantages of both approaches. Researchers proposed different criteria to assess the quality of mined policy [51] or satisfy various constraints while mining [38]. Xu and Stoller proposed an ABAC mining algorithm [73], a variant of which was developed to mine ABAC policies from logs [74]. Many approaches have been proposed to mine ABAC policies—for instance, multi-objective evolutionary optimization framework [47], identifying functional dependencies in database tables to mine ABAC policies [67], and algorithms to find both positive and negative policy rules [34]. Bui et. al. [6] utilized XuStoller's algorithm for mining policies in the ReBAC [14] context. Extended versions of this research were presented in [7, 10]. They also proposed a greedy approach for mining ReBAC policy [11]. Iyer et al. [35] proposed a ReBAC mining algorithm in evolving systems for mining graph transitions. The authors later proposed a method for active learning [36] of ReBAC policies from a black-box access control decision engine using authorization and equivalence queries. A universal access control policy mining, called Unicorn, was proposed by Cotrini et al. [18], which builds policies in a class of access control models including RBAC, ABAC, and ReBAC.

Using Machine Learning (ML) for Mining Access Control. Many researchers applied ML algorithms for mining access control policies. A probabilistic model for the role mining problem driven from the logical structure of RBAC was proposed by Frank et al. [22]. Deep learning was used to identify relevant attributes [1] to mine ABAC policies from natural language. Other methods including classification trees [13], deep recurrent neural network (RNN) [53], K-Nearest Neighbor (KNN) [20], Decision Tree [8, 9, 71] and Restricted Boltzmann Machine (RBM) model [50] have also been used to mine ABAC policy. The first unsupervised learning-based ABAC mining method used k-modes clustering to mine rules from historical operation data [40]. Naroui et al. [54] proposed to improve an existing ABAC policy by mining a policy using ML. Cotrini et al. [17] proposed an ABAC policy mining algorithm, named Rhapsody, built upon APRIORI-SD [42] which is an ML algorithm for subgroup discovery. Rhapsody mines a generalized policy based on sparse logs. Jabal et al. [37] proposed a novel framework for learning ABAC policies from data, named Polisma, combining data mining, statistical, and ML techniques. The Polisma mine a set of rules and applies ML techniques to include requests not covered by the mined rules. Karimi et al. [39] proposed an automatic approach for learning ABAC policy by extracting rules containing both positive and negative attributes and relationship filters.

Classical ML for Access Control Decision Making. Classical ML approaches have been widely used for making access decisions. Sanders et al. [59] presented an approach to mine ABAC policies while satisfying the least privilege principle in a large-scale organization. Symbolic and non-symbolic ML algorithms to infer ABAC policies from access logs have been proposed by Cappelletti et al. [12]. Liu et al. [45] proposed a permission decision engine scheme for ABAC based on Random Forest [5]. The method, called EPDE-ML, decoupled the decision engine from the real access control policy by moving policy updates into a separate phase.

Following our discussion in Section 2, DLBAC is fundamentally different from current approaches. It aims to replace traditional rule-based methods with a neural network, leading to better decision accuracy, generalizability, and engineering ease. Comparing to the related works that uses classical machine learning, we will show

that a deep learning based approach provides superior performance while being amenable to usable explainability in practice.

4 DLBAC$_\alpha$: A CANDIDATE DLBAC MODEL

This section presents a prototype of DLBAC, namely DLBAC$_\alpha$, which is an access control model built upon the proposed DLBAC approach in Section 2. As illustrated in Figure 2, DLBAC models e.g., DLBAC$_\alpha$ need to be fed with authorization tuples and user/resource metadata. We apply DLBAC$_\alpha$ to two real-world and eight synthetic datasets. First, we explain synthetic datasets construction and introduce real-world datasets. Then, we discuss how the DLBAC$_\alpha$ neural network is trained, and access decisions are made.

4.1 Synthetic Dataset Generation

An approach to generate synthetic access control datasets was proposed in [73]. We adopt this approach with minor changes to generate multiple synthetic datasets and briefly discuss here. The algorithm first generates a set of attribute names for users and resources randomly. Next, it generates a set of *rules* based on those attributes and then uses these rules to inform user/resource creation and attribute value assignments. Each *rule* is a tuple of the form $\langle UAE; RAE; OP; C\rangle$, where *UAE* is the set of user attribute expression, *RAE* is the set of resource attribute expression, *C* is the set of constraints, and *OP* is a set of operations. For example, ⟨*title=student; type=document; read; department=department*⟩ is a rule where *title=student* represents the UAE, *type=document* is the RAE, *read* is the operation, and *department=department* is the constraint. A user will be authorized to operate on a resource if the user satisfies the UAE, the resource satisfies the RAE, and both the user and resource satisfy the constraint. For each rule, the algorithm generates a set of users that satisfy the rule and then generates resources where for each resource, there is at least one user available to satisfy the rule. The following user and resource are created based on the above rule: *user(student1, title=student, department=cs), resource(document1, department=cs, type=document).* Here, the *student1* and *document1* are the unique ids of a user and a resource, respectively. The user *student1* has two attributes (*title* and *department*), and the resource *document1* has two attributes (department and type). Also, student1 satisfies UAE, as the user has the title *student* which is part of the title in UAE. Similarly, document1 satisfies RAE for the *type* attribute. Both *student1* and *document1* satisfy the constraint as they are from the same department. Thus, according to this rule, *student1* has *read* access to *document1.*

Finally, once the rules are generated, users and resources are created, and attributes are assigned, it is straight-forward to create the authorization tuples. For each user, resource and operation combination that satisfies a rule, an authorization tuple is created or updated with a new operation, as the case may be.

4.1.1 Syntax of Synthetic Dataset. We adapt this data generation approach by creating many metadata instead of attributes. We maintain four operations and various metadata (eight to thirteen) for each user/resource for different datasets. We define the syntax of DLBAC$_\alpha$'s dataset to contain a set of authorization tuples. An authorization tuple could be illustrated of the form $\langle uid|rid|m_1^u : v_1, m_2^u : v_2, ..., m_i^u : v_i|m_1^r : v_1, m_2^r : v_2, ..., m_j^r : v_j|\langle op_1, op_2, op_3, op_4\rangle\rangle$. The uid and rid in the tuple indicates the unique id of a user and a resource, respectively. The next part gives the metadata values of all i metadata of a user and m_1^u indicates the first user metadata name (e.g. umeta0) whereas its value is indicated by v_1. The subsequent part presents the metadata values of all j metadata of a resource, and first resource metadata name (e.g. rmeta0) and its value are represented by m_1^r and v_1, respectively. The last part is a binary sequence with a '1' meaning 'grant' and a '0' meaning 'deny' for that operation. For example, ⟨1011|2021|*30 49 5 26 63 129 3 42 | 43 49 5 16 63 108 3 3* |⟨*1 1 0 1*⟩⟩ is a sample authorization tuple of our dataset where 1011 and 2021 are the user and resource's unique number. The next eight numbers indicate the metadata values of a user, the following eight numbers represent resource's metadata values, and the final four binary digits signify the user has op_1, op_2, op_4 access to the resource. (In the example above, we also skip naming the metadata with the assumption that it could be inferred from the position of the metadata value.) For simplicity, we assume the metadata values in our datasets are categorical, and each metadata value is an *integer* representation of a category. We anticipate that our results will hold even in cases of metadata with *real numbers.*

4.1.2 Dataset Visualization. We use t-SNE plots [68] to visualize the samples in our datasets. A t-SNE plot discovers relationships in the data by identifying analogous clusters of data points with several features and projecting high dimensional features into a low dimensional feature space while retaining essential information. We project each of our samples to a 2-dimensional feature and plot them. (Note that our datasets have varying numbers of features/metadata ranging from 16 to 26 in total.) Each *dot* in the plot (Figure 3) represents an authorization tuple, where multiple tuples of the same color indicate that they have the same access permissions. For example, two tuples with only *read* and *write* access permissions will have the same color. Notably, a dataset with n operations will have tuples with 2^n different access combinations and plotted with 2^n distinct colors. For instance, the authorization tuples of a dataset with two operations (e.g., read and write) can be plotted with four different colors (tuples with the read access, write access, read and write access, and no access).

The position of a tuple in the plot is fixed according to the user and resource metadata values. For instance, two different users with the same metadata values may have access to a resource (or multiple resources with the same metadata values). Therefore, these tuples will have the same position regardless of their access permissions. Figure 3(a) depicts a dataset of 1650 users and 320 resources. The tuples (dots) take the position all over the plot according to their metadata values. This dataset has four different operations, and thereby, there are tuples with 16 distinct colors. However, we observe different tuples with the same access permission (same color) are grouped, and groups are isolated from one another, as shown in the figure with blue circles. Thus, a simple classifier can easily distinguish them without much difficulty (e.g., making one rule for each circle). The access control states in real-world situations might be much more complicated [52]. Figure 3(e) shows the visualization of a dataset from Amazon. Even though this is not a complete access control scenario of the entire Amazon enterprise [17], samples with access (green dots) significantly overlap with other samples without access (red dots) which means tuples with very similar

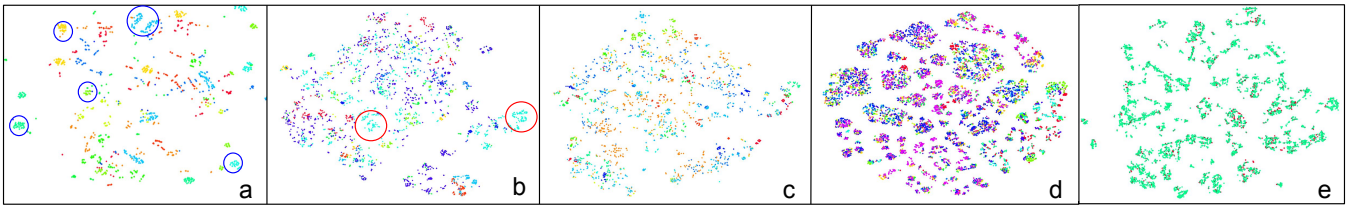

Figure 3: Comparing Complexity of Datasets. (a) A dataset with 1650 users, 320 resources, eight user/resource metadata, (b) A dataset with 1000 users, 639 resources, 11 user/resource metadata, (c) A dataset with 800 users, 656 resources, 11 user/resource metadata, (d) 4500 users, 4500 resources, 11 user/resource metadata, (e) An Amazon dataset [2] (The dataset has more samples with 'grant' access. Therefore, for better visualization, we considered all the tuples with 'deny' access permission and randomly selected a similar number of tuples with 'grant' access permission).

metadata values have entirely different accesses. A simple classifier would create too many rules to model such a dataset.

4.1.3 Introducing Complexity into Synthetic Datasets. Informed by t-SNE visualization of the Amazon dataset, we seek to introduce complexity into our synthetic datatsets to closely reflect real-world situations. We observe that, in practice, the access privileges of users and resources with somewhat similar metadata could vary. It is also expected that not all the metadata of a user/resource would contribute equally to their permissions. To reflect such scenarios in some of our datasets, we determine accesses considering *all* the metadata values of user and resource but *hide* a portion of metadata values from the policy miner (when dealing with mining methods) and model training phase (when dealing with ML methods). However, during rule generation, we ensure that the metadata that we will hide contributes to a lesser extent toward permission decisions by excluding them from being part of the *constraint* of the rules. In a nutshell, hiding metadata attempts to simulate a scenario, where access decisions are made based on access control *attributes* that are not fully informed by the entire metadata set. In a perfect world, access control attributes could succinctly capture the relevant metadata distributed across an organization. However, it is reasonable to hypothesize that this is not a practical assumption.

Figure 3(b) represents a dataset with 11 user and 11 resource metadata. The authorization tuples were created considering all the metadata. To simulate a similar situation while visualizing tuples and understand how it looks from a policy mining (or classification) perspective, we make only the first eight user and the first eight resource metadata values available for visualization. As shown, many dots of other colors now start to mix, indicating two tuples with similar user-resource metadata values may have very different accesses. Indeed, such proximity of the tuples is challenging for clustering or rule creation. The more metadata we hide, the more complicated the dataset for policy mining and ML approaches.

We still notice a few portions in the plot where the same colored dots are clustered together and separable, as shown with red circles in Figure 3(b). This is because the dataset generation algorithm [73] creates the metadata values based on a distribution, where the value range (i.e., number of values for each metadata) is sparse. For example, there are around one hundred different values for specific metadata (e.g., *department*) in a dataset with hundred users. One can easily cluster all the users into hundreds of groups based only on the department. However, there might be the case where metadata values are required to be chosen from a set of a limited number of values (say, ten departments for hundred users). Evidently, it is harder to cluster one hundred users into ten groups than a hundred. To reflect this, for each metadata, we define a fixed and smaller set of values (6 to 20 unique values) following the same distribution used by Xu et al. [73]. We choose each metadata value from the corresponding list during user/resource creation and metadata value assignment. This strategy creates datasets with significantly overlapped samples as depicted in Figure 3(c). We also extended the number of users and resources to simulate a larger organization, which adds more overlaps among samples and higher complexity to the dataset, as shown in Figure 3(d).

Table 1: List of Datasets.

#	Dataset	Type	Users	User Metadata	Resources	Resource Metadata	Authorization Tuples
1	*amazon-kaggle*	Real-world	9560	8	7517	0	32769
2	*amazon-uci*	Real-world	4224	11	7	0	4224
3	*u4k-r4k-auth11k*	Synthetic	4500	8	4500	8	10964
4	*u5k-r5k-auth12k*	Synthetic	5250	8	5250	8	12690
5	*u5k-r5k-auth19k*	Synthetic	5250	10	5250	10	19535
6	*u4k-r4k-auth21k*	Synthetic	4500	11	4500	11	20979
7	*u4k-r7k-auth20k*	Synthetic	4500	11	7194	11	20033
8	*u4k-r4k-auth22k*	Synthetic	4500	13	4500	13	22583
9	*u4k-r6k-auth28k*	Synthetic	4500	13	6738	13	28751
10	*u6k-r6k-auth32k*	Synthetic	6000	10	6000	10	32557

Finally, we synthesized eight different datasets (datasets #3-#10 in Table 1) used for DLBAC$_\alpha$ experimentation and evaluation, with varying numbers of users, resources, user and resource metadata, and authorization tuples, each reflecting a varying degree of complexity. We use the following naming convention for our synthetic datasets, as listed in Table 1: $u\langle$approx. number of users$\rangle - r\langle$approx. number of resources$\rangle$ – $auth\langle$approx. number of authorization tuples$\rangle$. We use 'u', 'r', and 'auth' to indicate users, resources, and authorization tuples, respectively. We also visualize all the synthetic datasets in Figure 4(a-h) using t-SNE plots. As illustrated in the Figure, each plot has dots with one of 16 different colors (for four operations), and those dots mix extensively.

4.2 Real-world Dataset

Amazon published two datasets that contain access control information which is widely used in access control research [12, 17, 39]. We name these datasets as *amazon-kaggle* and *amazon-uci*, and list them in the Table 1 (1-2). The *amazon-kaggle* dataset was released in Kaggle [2] (a platform for predictive modeling competitions) as a challenge to the community to build a machine learning model to determine the employees' accesses. The dataset holds historical access data where employees were manually allowed or denied

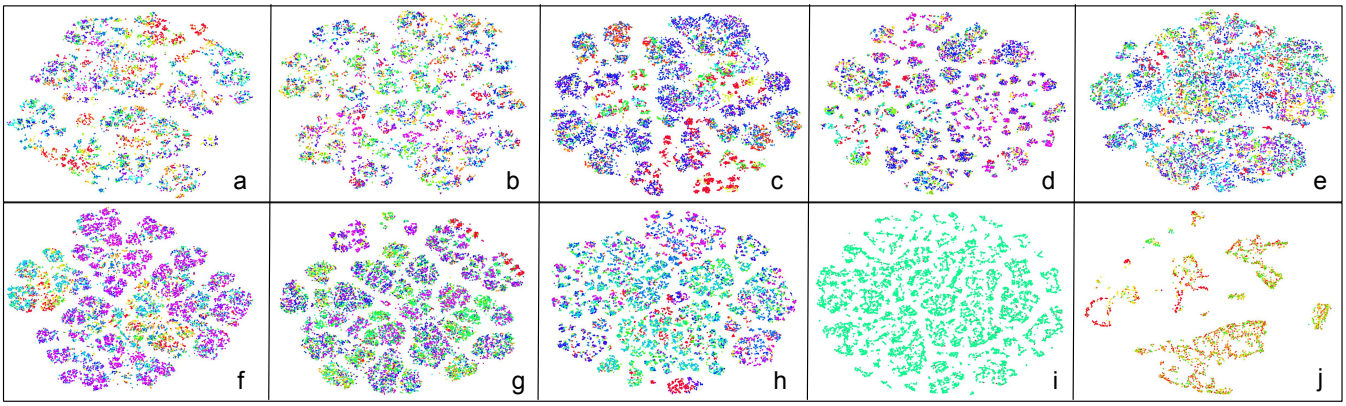

Figure 4: t-SNE Visualization of Synthetic and Real-world Datasets from Table 1. Figures a-h corresponds to synthetic datasets #3–#10 and i-j corresponds to real-world datasets #1–#2 in Table 1, respectively.

access to resources over time. The dataset has about nine thousand users and seven thousand resources with over 32K authorizations tuples. Each tuple specifies eight user metadata that depicts a user's properties, a resource id to identify the resource, and a binary flag to indicate whether the user has access to the resource or not. However, the dataset is highly imbalanced, and about 93% of the tuples are with grant access. We visualize this dataset in Figure 4(i). As we see, there are dots from two colors where green and red correspond to tuples with grant and deny access, each. Notably, a significant number of dots are from grant accesses.

The *amazon-uci* dataset was provided by Amazon in the UCI machine learning repository [3]. This dataset contains access information of more than 36,000 users and 27,000 permissions. For any permission, less than 10% of all users have requested access. The dataset is widely used in access control researches, and in most of the cases, the experiments are confined to only 5 to 8 most requested permissions [12, 17, 39]. Likewise, we took the seven most requested permissions, and for each permission, we list users who have access to the selected permissions. However, this dataset is also imbalanced, around 75% tuples with the deny access. In addition, the dataset is not ABAC in nature, and there are some tuples in the dataset where users with identical attribute values do not have the same access permissions. Because of this, policy mining or classification approaches may suffer while clustering the users for different permissions. We visualize this dataset in Figure 4(j).

4.3 Neural Network Architecture and Training

For DLBAC$_\alpha$, the deep neural network takes user/resource metadata values as input. It includes a classification layer with the number of neurons equal to the number of operations, where each neuron outputs the *probability* of granting the permission for a related operation, *op*. Given a feature vector x of the user and resource metadata, the neural network can be defined as a prediction function f such that $\hat{y} = f(x)$, where $\hat{y}$ is the predicted label or permission (grant (1) or deny (0)) of *op*, obtained from comparing the probability of granting the permission at the output of the network with a *threshold*. Note that for all DLBAC$_\alpha$ experimentation, we consider a threshold of 0.5. To train the neural network f (i.e. determine the network's weights), a set of training authorization tuples $\mathbb{X}$ of size N is collected, where (x_i, y_i) denotes the i-th sample in $\mathbb{X}$,

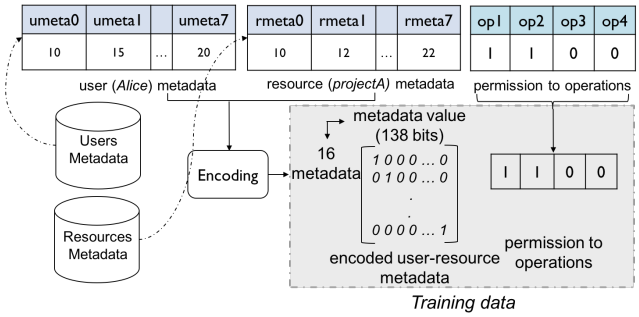

Figure 5: Preparing Training Data for DLBAC$_\alpha$.

where x_i is the feature vector of the user and resource metadata and y_i is the corresponding *op* or the target label. As discussed in Section 4.1.3, for some synthetic datasets, we hide a portion of metadata from the policy mining algorithm to mimic some complex situations. We apply this for the datasets having more than eight user metadata and eight resource metadata. In such a case, x_i represents the feature vector of the first *eight user metadata* and the first *eight resource metadata*. So, for example, for a dataset with 13 user-resource metadata, we hide five metadata from the user metadata and five from the resource metadata.

Since the metadata values in our dataset are categorical, we map them to binary values by utilizing encoding. We encode a tuple's user/resource metadata value using *one-hot encoding* [27] to transform the categorical values into a two-dimensional binary array. The row in the array represents metadata, and the column holds the encoded binary representation of the corresponding metadata value. (Amazon dataset's metadata values are too sparse, and we use *binary encoding* for them considering its memory efficiency in such cases [63]). As each operation is binary, we apply them directly as the target labels without any processing. Figure 5 illustrates the overall training data preparation for DLBAC$_\alpha$. As illustrated, we encode the metadata values of a user *Alice* and a resource *projectA* and apply permissions to the associated operations without further processing. We train DLBAC$_\alpha$ based on the training data, and this *trained* DLBAC$_\alpha$ is used in *Access Control Decision Engine* (discussed below) to produce access decisions for test data.

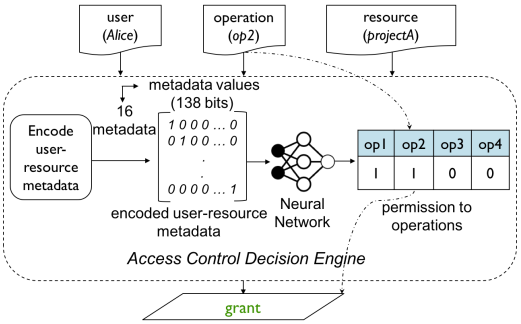

Figure 6: Decision Making Process in DLBAC$_\alpha$.

4.4 Decision Making Process in DLBAC$_\alpha$

Access Control Decision Engine (Decision Engine) is a DLBAC component responsible for receiving and authorizing any access request. In DLBAC$_\alpha$, the *Decision Engine* (Figure 6) takes three inputs (*user, resource, and operation*). The *Decision Engine* retrieves the user and resource metadata from the internal databases and then encodes them to obtain corresponding binary representation. The encoded input is fed into the neural network to predict the corresponding request's access permission. The network outputs access information for all the operations. The decision engine then determines the actual access authorization based on the requested access and the network's output. For the specific example in Figure 6, user *Alice* wants *op2* access on *projectA* resource. The output of the neural network for the *op2* is 1, which indicates that *Alice* has *op2* access on *projectA*. Thus, the *Decision Engine* authorizes this request.

5 EVALUATION

In this section, we experimentally evaluate the performance of DLBAC$_\alpha$ using both synthetic and real-world datasets.

5.1 Evaluation Methodology

We experiment and evaluate the performance for all the datasets listed in Table 1. We consider each dataset to represent an organization with its own unique characteristics. We split each dataset into training (80%) and testing (20%) sets. As the test dataset is entirely *unseen* during training, the evaluation shall adequately measure the generalization of any method.

Instances of DLBAC$_\alpha$. DLBAC is agnostic to deep neural network architecture, and we will show that the deep learning-based model's performance is consistent across datasets. For demonstration, we implement three instances of DLBAC$_\alpha$ using three distinct deep neural network architectures including ResNet [29], DenseNet [33], and Xception [15], and name them as DLBAC$_{\alpha-R}$, DLBAC$_{\alpha-D}$, and DLBAC$_{\alpha-X}$, respectively. For DLBAC$_{\alpha-R}$, we use the ResNet network with depth 8 for the first four datasets in Table 1 whereas, for the rest of the datasets, we use a ResNet with depth 50. For DLBAC$_{\alpha-D}$, we use the DenseNet architecture with [6,12,24,16] layers in the four dense blocks. We adopt the source code from the Keras application for all the model architectures implementation.[1]

The DLBAC$_\alpha$ instances were developed in Python using Keras library with a TensorFlow backend and trained on Google Colab (a 12GB NVIDIA Tesla K80 GPU). We apply Adam optimizer with an initial learning rate of 0.001, scheduled to reduce the learning rate by dividing by ten after every 10 epochs. The epoch and batch size was chosen as 60 and 16, respectively, with an early stop after 5 consecutive epochs without any performance improvement. As the DLBAC$_\alpha$ outputs an access probability between '0' and '1' for each operation, we use binary cross-entropy loss. We have created a repository in GitHub consisting of all the datasets, source code, and trained networks.[2]

Machine learning (ML) algorithms. We compare the performance of DLBAC$_\alpha$ instances with classical machine learning approaches such as Support Vector Machine (SVM) [16] and Random Forests (RF) [5]. We also compare with Multi-Layer Perceptron (MLP) [62] with four hidden layers (a shallow network) to evaluate how significant the performance difference is between a deep and a non-deep neural network. We use the *SVC* and *RandomForestClassifier* class of the Python scikit-learn library [57] for SVM and RF implementation, respectively, with their default configurations. We implement MLP using Keras library.

Policy mining algorithms. There is no other existing *deep learning-based* access control approach to the best of our knowledge, so a direct comparison of our work results is not currently possible. Therefore, we compare DLBAC$_\alpha$ with ABAC policy mining algorithms being one of the flexible and generalized access control approaches. We compare the performance of DLBAC$_\alpha$ instances with the following policy mining algorithms. While a few other works exist as discussed in our related work, a key decision factor in selecting these works was our ability to readily access their source codes and our ability to clearly understand, modify/tweak as needed and compile them.

(1) The policy mining algorithm proposed by Xu and Stoller [73], which we refer to as XuStoller.
(2) Rhapsody [17], a policy mining algorithm built upon an ML algorithm for subgroup discovery named APRIORI-SD [42]. Rhapsody performance has a direct correlation with multiple parameters. We trial with different parameter values and selected the policy with the highest F1 score while maintaining an FPR below 0.05.
(3) EPDE-ML [45], a permission decision engine scheme based on ML where a trained RF model makes the access control decision.

Evaluation metrics. For ML algorithms, we compute the F1 score and compare the performance with DLBAC$_\alpha$ instances and show that deep learning based algorithms generally perform better than traditional ML and MLP techniques. For an extensive comparison against policy mining algorithms, we compute the F1 score, False Positive Rate (FPR), True Positive Rate (TPR), and Precision. We consider the standard definitions [17, 45] of these evaluation metrics. Policies (or models) with a higher F1 score lead to better generalization. They can make more accurate access control decisions on users and resources with attributes not explicitly seen during the mining (or training) process. Also, the higher TPR and Precision are better as these scores indicate how accurately and efficiently the policies (or models) can grant access. On the contrary, the policies (or models) with a lower FPR are better as they are

[1] https://github.com/keras-team/keras-applications

[2] https://github.com/dlbac/DlbacAlpha

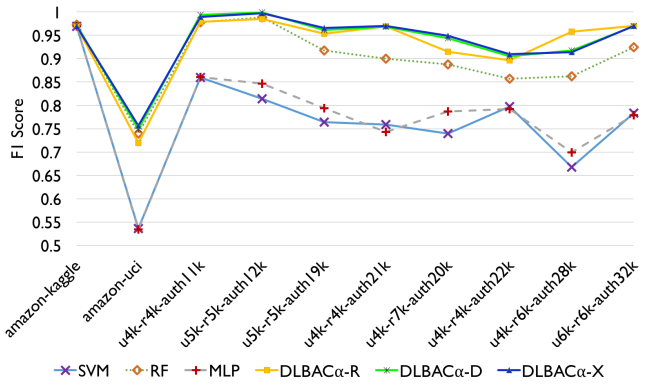

Figure 7: F1 Score Comparison: ML Algorithms vs. DLBAC$_\alpha$ Instances.

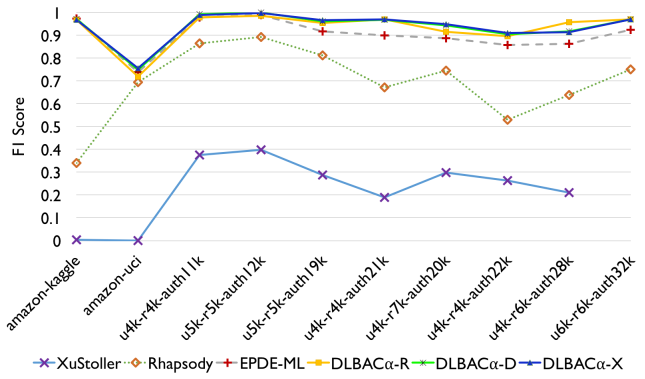

Figure 8: F1 Score Comparison: Policy Mining Algorithms vs. DLBAC$_\alpha$ Instances.

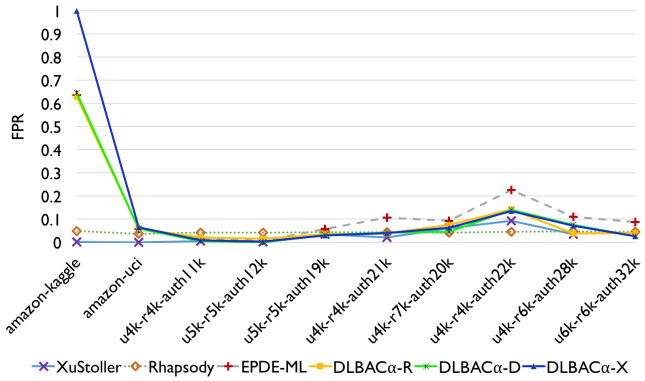

Figure 9: FPR Comparison: Policy Mining Algorithms vs. DLBAC$_\alpha$ Instances.

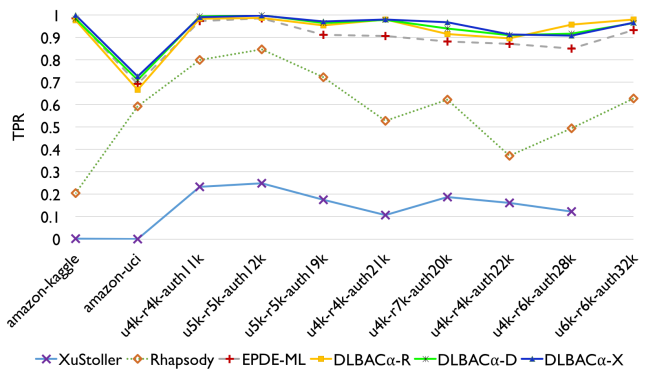

Figure 10: TPR Comparison: Policy Mining Algorithms vs. DLBAC$_\alpha$ Instances.

less likely to give access to requests, those which should be denied according to the ground truth access control policy.

5.2 Results

5.2.1 Performance comparison with ML algorithms. Figure 7 illustrates the overall performance of all ML approaches and DLBAC$_\alpha$ instances for each dataset with respect to F1 score. The performance of all the algorithms is consistent and better for the *amazon-kaggle* dataset, but it is not the case for *amazon-uci* dataset. In this case, SVM and MLP performed significantly lower, and other approaches including DLBAC$_\alpha$ instances could not achieve high performance. Such a result is expected due to the inconsistency in the access permissions in that dataset. The dataset contains tuples where users with identical attribute values do not have the same access permissions, as discussed in Section 4.2. The advantage of the DLBAC approach is more evident if the dataset is processed correctly. We show this using our synthetic datasets. For synthetic datasets, DLBAC$_\alpha$ instances achieved the highest F1 score, while SVM and MLP performed the worst. Also, the instances of DLBAC$_\alpha$'s improvements over RF are significant (for p-value < 0.05; paired T-test (not shown here)) for all the synthetic datasets except $u5k\text{-}r5k\text{-}auth12k$ dataset. The performance advantage of DLBAC$_\alpha$ instances is particularly pronounced in synthetic datasets with a large number of authorization tuples, where DLBAC$_\alpha$ instances report 0.03 to 0.09 improvements over RF as shown in the figure. Notably, the performances of all algorithms vary with the complexity of the datasets. However, DLBAC$_\alpha$ instances show the lowest variation in its performance across the datasets, implying that DLBAC$_\alpha$ is most robust against changes in data characteristics such as number of hidden metadata, number of users, resources, and authorization tuples. Except for the $u4k\text{-}r6k\text{-}auth28k$ dataset, all algorithms' performances drop with the increase of hidden metadata (e.g., the $u4k\text{-}r4k\text{-}auth22k$ dataset with 13 metadata where we hide 5 of the metadata from the feature vector as discussed in Section 4.3), suggesting that an increase in data complexity generally impacts performance. Overall, the experimental results indicate that DLBAC$_\alpha$ is more effective and robust than classical ML approaches, including MLP, for making accurate access decisions.

While the performance advantages of DLBAC$_\alpha$ are not apparent in the Amazon datasets, we emphasize that those datasets are not reflective of the access state complexity of the entire organization but that of a small portion of the company. This is one of the reasons why we synthesized additional datasets for our experimentation.

5.2.2 Performance comparison with policy mining algorithms. Figures 8, 9, 10, and 11 compare the F1 score, FPR, TPR, and Precision, respectively of policy mining algorithms and DLBAC$_\alpha$ instances. We could not experiment XuStoller algorithm for the largest synthetic dataset ($u6k\text{-}r6k\text{-}auth32k$) as it took a very long runtime

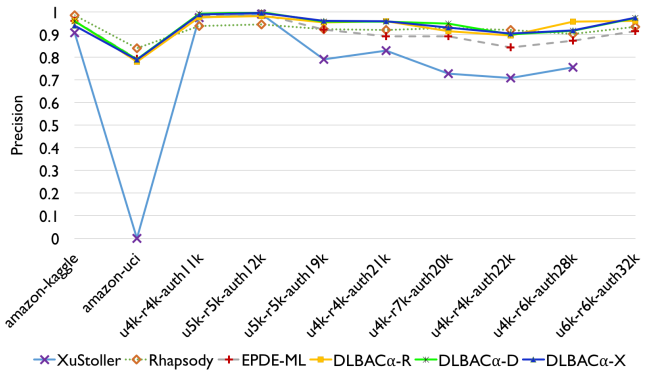

Figure 11: Precision Comparison: Policy Mining Algorithms vs. DLBAC$_\alpha$ Instances.

without any output. We can make the following observations based on all these experimental results.

- *A deep learning based approach can make more accurate access control decisions and generalize better.* The F1 score of EPDE-ML and DLBAC$_\alpha$ instances are significantly better than the rule-based approaches such as XuStoller and Rhapsody. That signifies, in general, machine learning-based approaches can make better generalization and accurate access control decisions. As we see in Figure 8, the performance improvements of DLBAC$_\alpha$ instances over EPDE-ML, which is statistically significant for most datasets, suggest that deep learning based algorithms make the more accurate decision and have even better generalization capability than classical ML-based policy mining approaches.
- *A deep learning based approach can properly balance both over-provision and under-provision.* XuStoller and Rhapsody achieved the best FPR as shown in Figure 9, indicating they are unlikely to give access to requests that should be denied according to the actual access control policy. However, that is not the case for denying access. As we see in Figure 10, the TPR for Rhapsody is between 0.2 to 0.85, and for XuStoller, it is from 0 to 0.25. Such a lower TPR indicates that these algorithms are pretty inefficient while denying access (under-provisioned) even though the requests deserve grant access according to the actual access control policy. On the other hand, the EPDE-ML performed lowest in terms of FPR (over-provisioned) across synthetic datasets ranging from 0.06 to 0.23. Their average TPR and Precision are below 0.9. Such higher FPR and comparably lower TPR and Precision suggest that EPDE-ML could not achieve desirable performance in terms of over-provision and under-provision. The DLBAC$_\alpha$ instances obtained a much higher TPR and Precision as illustrated in Figure 10 and 11. Also, the DLBAC$_\alpha$ instances reached an FPR which is comparable to XuStoller and Rhapsody, as shown in Figure 9. This suggests that deep learning based approach can balance better between over- and under-provisioning.
- *An imbalanced dataset may affect some performance that could be calibrated.* The FPR result of DLBAC$_\alpha$ instances for *amazon-kaggle* dataset is high, and the TPR for *amazon-uci* dataset is relatively low, arising due to the characteristics of the dataset [70]. As discussed in Section 4.2, these datasets are imbalanced, one has unreasonably more samples from the grant class, and the other has more from the deny class. We argue that this is a typical machine learning problem, and EPDE-ML has a similar performance for these datasets. For balanced datasets, the FPR and TPR of DLBAC$_\alpha$ instances are consistent, and FPR is below 0.05 for most of the datasets while TPR is above 0.95, as demonstrated in Figure 9 and 10. Evidently, these metrics could be calibrated based on the tolerance to over vs. under-provisioning of an application context. Specifically, one could favor a particular metric in DLBAC$_\alpha$ by modifying the loss function to increase the weight of the minority class [31], adjusting the threshold for granting permissions as described in Section 4.3, etc. We note that an improvement in one of the metrics will likely negatively impact one or more of the others and that every work is prone to this issue.

Overall, the DLBAC$_\alpha$ instances achieved better or comparable performance for all the metrics across datasets, suggesting that deep learning based approaches generalized better and made more accurate access control decisions than rule based and classical ML based approaches. These results demonstrate the effectiveness of using DLBAC as an access control system.

6 UNDERSTANDING DLBAC DECISIONS

As the core of a DLBAC system is a *neural network*, a major challenge is providing insights into why and how DLBAC makes certain decisions. That is, explainability is a key challenge for DLBAC. For instance, in Figure 6, the *Decision Engine* received a request that user *Alice* wishes *op2* access on *projectA* resource. Based on the result of the *neural network*, the decision engine approved the request. However, it is not quite obvious why the *neural network* made that prediction for this request. Such a justification is generally straightforward in traditional access control systems as the decisions are made based on written policies. But, it is challenging for DLBAC due to the *black-box* nature of a neural network [64]. As the decisions are made based on provided *user/resource metadata*, it is essential to understand why a decision is made and which metadata influenced that decision. Many techniques have been introduced to help gain insights into a neural network's internal details. In this section, we investigate two state-of-the-art approaches for this purpose: *Integrated Gradients* [66] and *Knowledge Transferring* [25]. We experiment on DLBAC$_{\alpha-R}$ instance (as introduced in Section 5.1), and for brevity, we refer it to DLBAC$_\alpha$ in the following discussion. The related source code is uploaded to GitHub.[3]

6.1 Integrated Gradients

Integrated Gradients is an effective interpretation technique that focuses on attributing the decisions of a neural network to the input features of the prediction samples. It attributes a network's decision to its input features in terms of gradient, which specifies the most effective elements for a decision. To understand the *decision* of DLBAC$_\alpha$ for a *tuple*, it is required to provide the *user/resource metadata values*, the *decision*, and the *neural network* as input to the *Integrated Gradients*. Then, *Integrated Gradients* outputs the *attribution scores* of the input metadata that we normalize in the scale of 0 to 1, denoting the degree of impact on the decision. Such an interpretation is known as *local interpretation* which helps to better understand each decision of a neural network. It is also helpful to have a *global interpretation* [26] of a network to understand the

[3] https://github.com/dlbac/DlbacAlpha/tree/main/understanding_dlbac_alpha

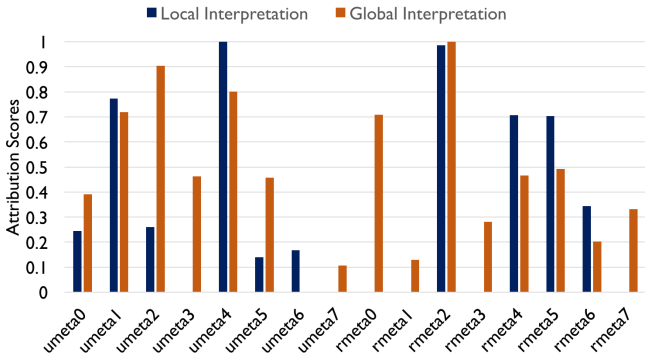

Figure 12: Local and Global Interpretation (blue: local interpretation of a decision in DLBAC$_\alpha$. orange: global interpretation of DLBAC$_\alpha$ for the grant access of an operation).

network's overall knowledge. Generally, it takes a set of decisions to generate a global interpretation of a network.

For DLBAC$_\alpha$, we investigate explainability for both specific access control decision and the overall knowledge learned by the network. For this purpose, we train DLBAC$_\alpha$ on our *u4k-r4k-auth11k* dataset. Then, we request *op1* operation access to *projectD* resource for a user *Dave* and the Decision Engine grants the request. To learn the reason behind this decision, we perform local interpretation with metadata values of *Dave* and *projectD*, the decision (*grant* access on op1 operation), and the *DLBAC$_\alpha$ network*. As depicted in Figure 12 (blue bars), for this particular request, user's *umeta4* and resource's *rmeta2* metadata are the most important and influential.

To achieve global interpretation for the *grant* access to *op1* operation, we take the *op1* access information for a set of fifty random samples with grant access from the *u4k-r4k-auth11k* dataset. (Note that the more samples we use, the more precise the result we get. However, Integrated Gradients is memory inefficient, so we could not test more samples using our workstation with 16 GB of memory). We provide each user/resource metadata and their respective decisions for the *op1* operation to *Integrated Gradients* to obtain the global interpretation. Figure 12 (orange bars) depicts the global interpretation of DLBAC$_\alpha$ for the *u4k-r4k-auth11k* dataset for the *grant* access to *op1* operation. The result identifies resource's *rmeta2* as the most influential metadata, user's *umeta2* as second most influential metadata, and the *attribution scores* of other metadata.

6.1.1 Application of Integrated Gradients based Understanding. Improved explainability could be utilized to achieve other benefits. For instance, developers can utilize interpretation techniques to debug [55] incorrect decisions from the *neural network* in *decision engine*. We show that *Integrated Gradients* based interpretation can be used to grant/deny access permissions by modifying proper metadata. As shown in Figure 13 (tuple2), the user *Carol* doesn't have op1 access on *projectC* resource. Applying local interpretation on other tuple with op1 access (e.g., Dave has the op1 access to projectD resource as shown in Figure 13 (tuple1)) revealed the *attribution scores* of different metadata for op1 operation. As circled in tuple1, *Dave*'s 'umeta1' and 'umeta4' are the most dominant metadata for this specific access. To grant Carol's op1 access on projectC resource, we utilize *attribution scores* of tuple1. Our result shows that replacing Carol's 'umeta1' and 'umeta4' metadata value with Dave's metadata

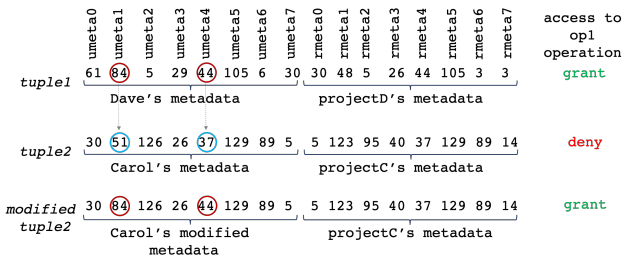

Figure 13: Applying Integrated Gradients to Grant Access.

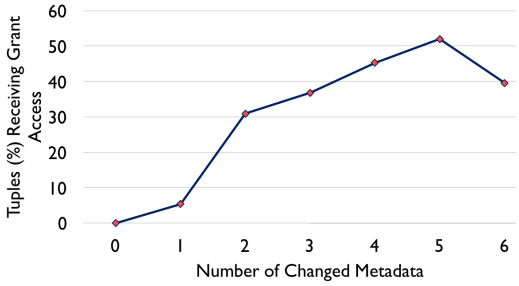

Figure 14: Modifying Metadata Value Based on Integrated Gradients Across Tuples (4581 tuples) with Deny Access.

value enables Carol's op1 access to projectC resource. As reported, modifying one or two significant metadata changes corresponding access. The opposite holds for least-significant metadata where modifying multiple metadata could not alter related access.

We also experiment with such a metadata value modification across all the samples of a specific decision (e.g., tuples with 'deny' access on op1) in the same dataset to see the impact of a global interpretation. The idea is to alter the metadata values of influential metadata for a specific decision. For instance, according to Figure 12, the *rmeta2* has the most influence on denying access. If we alter the *rmeta2* value of any tuple, say *tupleA*, with a rmeta2 value from a known tuple with *grant* access, say *tupleB*, the chance of denying access for *tupleA* might reduce and increases the chance of getting access. Also, altering the value of multiple influential metadata of the same tuple may eventually help to get access.

To investigate that we alter different metadata values, one by one, of all the tuples with *deny* access (4581 such tuples) on *op1* in order of their significance level in the global interpretation. For example, we first change the value of rmeta2 metadata, next umeta2 metadata, and so on. We utilize the user/resource metadata values from tuple1 in Figure 13 as a known tuple with the 'grant' access for *op1* operation. As described in Figure 14, initially, with no change, no tuple has been granted access. However, with the change of first metadata (rmeta2), around 5% of the tuples receive grant access. By changing the second metadata value, around one-third of the tuples get grant access. A similar surge continues to obtain grant access for over half of all the tuples before decreasing for sixth metadata (rmeta5) modification, reaching below 40%. It indicates that some tuples with such a big number of metadata modifications fall under a different distribution for which they have the 'deny' access. Overall, using this technique, a system admin can estimate the impact of metadata value vs. dependent accesses.

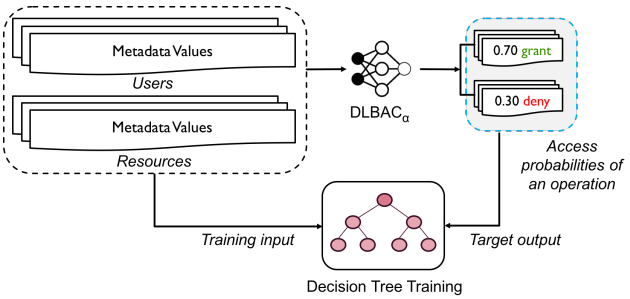

Figure 15: Knowledge Transferring Technique.

6.2 Knowledge Transferring

Although *Integrated Gradients* determines the *attribution scores* of each metadata for a decision, it does not establish the relationship among metadata or the network's logic [66]. *Knowledge Transferring* could help to identify such relationships. With *Knowledge Transferring*, we can extract a decision tree to approximately understand the decision of DLBAC$_\alpha$ in the form of traditional rules. While accurate reconstruction of the representation details is infeasible, the generated decision tree will give an approximate explanation of the underlying logic of DLBAC$_\alpha$. Though the decision tree makes classification decisions understandable [57], it does not generalize as well as deep neural networks. Interestingly, a neural network's generalization ability is likely to be transferred to a decision tree through a method called *distillation* [30], which has been widely used in ML literature and we adopt in this paper. As explained in Section 4.3, DLBAC$_\alpha$ outputs whether a user has access to any resource by giving the *probability* of granting the access instead of a direct *yes/no* answer. Therefore, we can determine those *probability outputs* for all the *tuples* in a dataset and train a decision tree, as shown in Figure 15. These probabilities represent the *knowledge* of the neural network, and we aim to transfer it to a decision tree.

To explore *Knowledge Transferring* technique, we train DLBAC$_\alpha$ for the *u4k-r4k-auth11k* dataset that we used for the Integrated Gradients experiment. We take *op1* access *probabilities* for all the samples. Then, we train a decision tree (DT) on the same dataset. However, instead of giving corresponding ground-truth permissions from the dataset, we provide the *probability* outputs of DLBAC$_\alpha$. Eventually, we construct a DT with a maximum depth of eight that facilitates retrieving underlying rules for any specific decision. While the tree serves the global interpretation, a rule for any specific decision represents the local interpretation. We apply the metadata values of *Dave* and *projectD* that we described in previous section, and retrieve the access rules from the DT for *op1* operation. We observe that *Dave* obtained *grant* access to *projectD* for *op1* based on following rule: $\Big\langle \langle \langle$ umeta0 $> 31 \wedge$ umeta0 $< 63 \rangle \wedge \langle$ umeta2 $< 20 \rangle \wedge \langle$ umeta6 $< 50 \rangle \rangle \wedge \Big\langle \langle \langle$ rmeta0 $< 72 \rangle \wedge \langle$ rmeta2 $< 18 \rangle \wedge \langle$ rmeta5 $< 111 \rangle \rangle \Big\rangle$. It is worth mentioning that the decision tree should be generated with unlimited depth to obtain more precise rules. Note that the Integrated Gradients and Knowledge Transferring techniques are orthogonal in terms of insights they each provide into the neural network and do not substitute each other. For better insights, one could use both methods.

7 FUTURE RESEARCH DIRECTIONS

In this section, we discuss some of the challenges for DLBAC and explore some ideas of how those could be addressed.

Access Control Administration. A major task in access control is policy administration, i.e. updating the system's rules to affect a particular change. It is a significant challenge for DLBAC as a policy change amounts to adjusting the current neural network's weights. This can, of course, be obtained by retraining the network. However, that is neither ideal nor cost-effective. Fine-tuning is one of the common approaches that helps a neural network to learn new changes by updating the current network's weight [43]. This allows one to implement policy changes by making minor changes to the network. However, the network should handle the issue of catastrophic forgetting, which is a common pitfall in fine-tuning. We believe these issues can be effectively mitigated by developing methods based on life-long learning [56] for DLBAC.

Adversarial Attacks. Adversarial attacks are a common concern for any machine learning based system, deep learning networks in particular. An adversary can obtain unauthorized access by fooling the network with modified samples that are indistinguishable from natural ones by human [75]. However, such attacks could be mitigated by applying adversarial training [46]. In the context of DLBAC, an adversarial attack is to acquire access permissions based on modified/perturbed user and resource metadata. In access control, the datasets and the adversarial attack profile are somewhat interesting and different from traditional image domains. One typically expects a mix of categorical and continuous metadata. Moreover, since some metadata are more trustworthy than others, an adversary does not have the complete flexibility to change an entire sample imperceptibly. These observations could be leveraged to develop better defenses against adversarial attacks in DLBAC. Another related aspect needs to investigate whether DLBAC can efficiently handle an access request if some of its user/resource metadata are deleted, a.k.a attribute hiding attack.

Bias and Fairness. As DLBAC learns based on metadata distributed across various parts of an enterprise, there might be different types of human biases or errors in training data. As such, the DLBAC network trained based on such data can inadvertently be biased, favoring some decisions. For example, as observed in the *amazon-kaggle* dataset, most of the authorization tuples were with 'grant' decision, and DLBAC$_\alpha$ instances were biased towards the same decision. Certain metadata could be influenced by various factors including ethnographic. Therefore, to obtain a fair and trustworthy DLBAC system, it is critical to audit training data, evaluate decisions for fairness, and establish a proper feedback loop [48].

DLBAC in Tandem with Traditional Access Control. Evidently, in practice, we do not foresee (nor advocate) that DLBAC will simply substitute traditional forms of access control immediately. One research challenge is how DLBAC could be effectively integrated to operate in tandem with traditional access controls such as RBAC or ABAC. One of the issues that will arise is conflicting decisions between, say, RBAC and DLBAC. If that conflict were to be resolved in favor of RBAC, that decision could be used to fine-tune the DLBAC network. DLBAC could also be used in the background for monitoring or auditing purposes.

8 CONCLUSION

We proposed DLBAC, a deep learning based access control approach, to deal with issues in classical access control approaches. As DLBAC learns based on metadata, it obviates the need for attribute/role engineering, policy engineering, etc. We implemented DLBAC_α, a prototype of DLBAC, using both real-world and synthetic datasets. We demonstrated DLBAC_α's effectiveness as well as its generalizability. As the core of DLBAC is a neural network, we applied two different state-of-the-art techniques to understand DLBAC decisions in human terms. We also discussed some future directions to build new models upon DLBAC and address current challenges, including access control administration issues.

ACKNOWLEDGMENTS

We would like to thank the CREST Center For Security And Privacy Enhanced Cloud Computing (C-SPECC) through the National Science Foundation (NSF) (Grant Award #1736209) and the NSF Division of Computer and Network Systems (CNS) (Grant Award #1553696) for their support and contributions to this research.

REFERENCES

[1] Manar Alohaly, Hassan Takabi, and Eduardo Blanco. 2018. A deep learning approach for extracting attributes of ABAC policies. In *SACMAT*. ACM.
[2] Kaggle Amazon. 2013. Amazon Employee Access Challenge in Kaggle. https://www.kaggle.com/c/amazon-employee-access-challenge/
[3] UCI Amazon. 2011. Amazon Access Samples Data Set. http://archive.ics.uci.edu/ml/datasets/Amazon+Access+Samples
[4] L. Bauer, L. F Cranor, et al. 2009. Real life challenges in access-control management. In *Conference on Human Factors in Computing Systems*.
[5] Leo Breiman. 2001. Random forests. *Machine learning* (2001).
[6] T. Bui et al. 2017. Mining relationship-based access control policies. In *SACMAT*.
[7] T. Bui et al. 2018. Mining Relationship-Based Access Control Policies from Incomplete and Noisy Data. In *Symposium on Foundations and Practice of Security*.
[8] Thang Bui and Scott D Stoller. 2020. A decision tree learning approach for mining relationship-based access control policies. In *SACMAT*. ACM.
[9] T. Bui and S. D Stoller. 2020. Learning Attribute-Based and Relationship-Based Access Control Policies with Unknown Values. In *Information Systems Security*.
[10] Thang Bui, Scott D Stoller, and Hieu Le. 2019. Efficient and Extensible Policy Mining for Relationship-Based Access Control. In *SACMAT*. ACM.
[11] T. Bui, S. D Stoller, and J. Li. 2019. Greedy and evolutionary algorithms for mining relationship-based access control policies. *Computers & Security* (2019).
[12] Luca Cappelletti et al. 2019. On the quality of classification models for inferring abac policies from access logs. In *Big Data*. IEEE.
[13] Suresh N Chari and Ian M Molloy. 2016. Generation of attribute based access control policy from existing authorization system. US Patent 9,264,451.
[14] Y. Cheng, K. Bijon, and R. Sandhu. 2016. Extended ReBAC administrative models with cascading revocation and provenance support. In *SACMAT*. ACM.
[15] François Chollet. 2017. Xception: Deep learning with depthwise separable convolutions. In *IEEE CVPR*.
[16] C. Cortes and V. Vapnik. 1995. Support-vector networks. *Machine learning* (1995).
[17] C. Cotrini et al. 2018. Mining ABAC rules from sparse logs. In *Euro S&P*.
[18] Carlos Cotrini, Luca Corinzia, Thilo Weghorn, and David Basin. 2019. The next 700 policy miners: A universal method for building policy miners. In *CCS*. ACM.
[19] S. Das, B. Mitra, V. Atluri, J. Vaidya, and S. Sural. 2018. Policy Engineering in RBAC and ABAC. In *From Database to Cyber Security*. Springer.
[20] M. A El Hadj, Y. Benkaouz, B. Freisleben, and M. Erradi. 2017. ABAC rule reduction via similarity computation. In *Intl. Conference on Networked Systems*.
[21] David F Ferraiolo, Dennis M Gilbert, and Nickilyn Lynch. 1995. An examination of federal and commercial access control policy needs. In *NIST-NCSC*.
[22] M. Frank et al. 2008. A class of probabilistic models for role engineering. In *CCS*.
[23] M. Frank et al. 2009. A probabilistic approach to hybrid role mining. In *CCS*.
[24] M. Frank et al. 2013. Role mining with probabilistic models. *TISSEC* (2013).
[25] S. Fukui et al. 2019. Distilling Knowledge for Non-Neural Networks. In *APSIPA*.
[26] Jindong Gu and Volker Tresp. 2019. Semantics for global and local interpretation of deep neural networks. *arXiv:1910.09085* (2019).
[27] J. Hancock et al. 2020. Survey on categorical data for neural networks. *Big Data*.
[28] M. A Harrison et al. 1976. Protection in operating systems. *Commun. ACM*.
[29] Kaiming He et al. 2016. Deep residual learning for image recognition. In *CVPR*.
[30] G. Hinton et al. 2015. Distilling the knowledge in a neural network. *arXiv*.
[31] Yaoshiang Ho and Samuel Wookey. 2019. The real-world-weight cross-entropy loss function: Modeling the costs of mislabeling. *IEEE Access* 8 (2019), 4806–4813.
[32] V. C Hu et al. 2013. Guide to attribute based access control (ABAC) definition and considerations (draft). *NIST special publication* (2013).
[33] Gao Huang et al. 2017. Densely connected convolutional networks. In *CVPR*.
[34] Padmavathi Iyer and Amirreza Masoumzadeh. 2018. Mining positive and negative attribute-based access control policy rules. In *SACMAT*. ACM.
[35] Padmavathi Iyer and Amirreza Masoumzadeh. 2019. Generalized Mining of Relationship-Based Access Control Policies in Evolving Systems. In *SACMAT*.
[36] Padmavathi Iyer and Amirreza Masoumzadeh. 2020. Active learning of relationship-based access control policies. In *SACMAT*.
[37] Amani Abu Jabal, Elisa Bertino, et al. 2020. Polisma-a framework for learning attribute-based access control policies. In *ESORICS*.
[38] J. H Jafarian, H. Takabi, et al. 2015. Towards a general framework for optimal role mining: A constraint satisfaction approach. In *SACMAT*.
[39] L. Karimi, M. Aldairi, J. Joshi, and M. Abdelhakim. 2021. An automatic attribute based access control policy extraction from access logs. *IEEE TDSC* (2021).
[40] Leila Karimi and James Joshi. 2018. An unsupervised learning based approach for mining attribute based access control policies. In *Big Data*. IEEE.
[41] Alan H Karp, Harry Haury, and Michael H Davis. 2010. From ABAC to ZBAC: the evolution of access control models. *Journal of Information Warfare* (2010).
[42] Branko Kavšek and Nada Lavrač. 2006. APRIORI-SD: Adapting association rule learning to subgroup discovery. *Applied Artificial Intelligence* (2006).
[43] A. Kaya et al. 2019. Analysis of transfer learning for deep neural network based plant classification models. *Computers and electronics in agriculture* (2019).
[44] L. Krautsevich et al. 2013. Towards attribute-based access control policy engineering using risk. In *Intl. Workshop on Risk Assessment and Risk-driven Testing*.
[45] A. Liu, X. Du, and N. Wang. 2021. Efficient Access Control Permission Decision Engine Based on Machine Learning. *Security & Communication Networks* (2021).
[46] A. Madry, A. Makelov, L. Schmidt, D. Tsipras, and A. Vladu. 2017. Towards deep learning models resistant to adversarial attacks. *arXiv:1706.06083* (2017).
[47] Eric Medvet, Alberto Bartoli, Barbara Carminati, and Elena Ferrari. 2015. Evolutionary inference of attribute-based access control policies. In *EMO*. Springer.
[48] N. Mehrabi et al. 2019. A survey on bias and fairness in machine learning. *arXiv*.
[49] B. Mitra et al. 2016. A survey of role mining. *Computing Surveys (CSUR)* (2016).
[50] D. Mocanu, F. Turkmen, and A. Liotta. 2015. Towards ABAC policy mining from logs with deep learning. In *International Multiconference (Intelligent Systems)*.
[51] Ian Molloy et al. 2009. Evaluating role mining algorithms. In *SACMAT*. ACM.
[52] I. Molloy et al. 2011. Adversaries' holy grail: access control analytics. In *Workshop on Building Analysis Datasets and Gathering Experience Returns for Security*.
[53] M. Narouei, H. Khanpour, H. Takabi, et al. 2017. Towards a top-down policy engineering framework for attribute-based access control. In *SACMAT*. ACM.
[54] Masoud Narouei and Hassan Takabi. 2019. A Nature-Inspired Framework for Optimal Mining of Attribute-Based Access Control Policies. In *ICSPCS*. Springer.
[55] A. Nguyen, J. Yosinski, and J. Clune. 2015. Deep neural networks are easily fooled: High confidence predictions for unrecognizable images. In *CVPR*. IEEE.
[56] G. I Parisi, R. Kemker, J. L Part, C. Kanan, and S. Wermter. 2019. Continual lifelong learning with neural networks: A review. *Neural Networks* (2019).
[57] F. Pedregosa et al. 2011. Scikit-learn: Machine Learning in Python. *JMLR*.
[58] Zhang Sainan and Zheng Changyou. 2019. Research and Application of Rigorous Access Control Mechanism in Distributed Objects System. In *ITNEC*. IEEE.
[59] Matthew W Sanders and Chuan Yue. 2019. Mining least privilege attribute based access control policies. In *ACSAC*.
[60] Ravi Sandhu et al. 1996. Role-based access control models. *Computer* (1996).
[61] Gerhard Schimpf. 2000. Role-engineering critical success factors for enterprise security administration. In *ACSAC*. ACM.
[62] Jürgen Schmidhuber. 2015. Deep learning in neural networks: An overview. *Neural networks* 61 (2015).
[63] Cedric Seger. 2018. An investigation of categorical variable encoding techniques in machine learning: binary versus one-hot and feature hashing.
[64] Avanti Shrikumar, Peyton Greenside, and Anshul Kundaje. 2017. Learning important features through propagating activation differences. In *ICML*. PMLR.
[65] Sara Sinclair and Sean W Smith. 2008. Preventative directions for insider threat mitigation via access control. In *Insider Attack and Cyber Security*. Springer.
[66] M. Sundararajan et al. 2017. Axiomatic attribution for deep networks. In *ICML*.
[67] Tanay Talukdar et al. 2017. Efficient bottom-up mining of attribute based access control policies. In *Intl. Conference on Collaboration and Internet Computing*. IEEE.
[68] L. V d Maaten and G. Hinton. 2008. Visualizing data using t-SNE. *JMLR* (2008).
[69] J. Vaidya et al. 2010. The role mining problem: A formal perspective. *TISSEC*.
[70] Cheng G Weng and Josiah Poon. 2008. A new evaluation measure for imbalanced datasets. In *7th Australasian Data Mining Conference-Volume 87*.
[71] Chengcheng Xiang, Yudong Wu, et al. 2019. Towards Continuous Access Control Validation and Forensics. In *CCS*. ACM.
[72] Z. Xu and S. D Stoller. 2012. Algorithms for mining meaningful roles. In *SACMAT*.
[73] Zhongyuan Xu and Scott D Stoller. 2014. Mining attribute-based access control policies. *TDSC* (2014).
[74] Zhongyuan Xu and Scott D Stoller. 2014. Mining attribute-based access control policies from logs. In *DBSec*. Springer.
[75] T. Zheng et al. 2019. Distributionally adversarial attack. In *AAAI Conf. on AI*.