Administration of Machine Learning Based Access Control

Mohammad Nur Nobi[1](✉), Ram Krishnan[2](✉), Yufei Huang[3], and Ravi Sandhu[4]

[1] Department of Computer Science, Institute for Cyber Security (ICS), The University of Texas at San Antonio (UTSA), San Antonio, TX 78249, USA
mohammadnur.nobi@my.utsa.edu

[2] Department of Electrical and Computer Engineering, ICS, NSF Center for Security and Privacy Enhanced Cloud Computing (C-SPECC), UTSA, San Antonio, TX 78249, USA
ram.krishnan@utsa.edu

[3] Department of Medicine, University of Pittsburgh, and UPMC Hillman Cancer Center, Pittsburgh, PA 15260, USA
yuh119@pitt.edu

[4] Department of Computer Science, ICS, C-SPECC, UTSA, San Antonio, TX 78249, USA
ravi.sandhu@utsa.edu

Abstract. When the access control state of a system is complex, Machine learning (ML)-based access control decision engines have demonstrated advantages of accuracy and generalizability. This field is emerging with multiple efforts where an ML model is trained based on either existing access control state or historic access logs; the trained model then makes access control decisions. This paper explores ML-based access control's administration problem, focusing on capturing changes in the access control state. We investigate this problem in a simulated system with Random Forest (RF) method from a symbolic ML class and the ResNet method from a non-symbolic one. Both classes have their respective advantages and disadvantages for issues such as insufficient learning of new changes and forgetting existing access information while updating the ML model. Our experimental results show that the non-symbolic approaches perform better than the symbolic ones while adjusting for continual (or incremental) changes in the access control state.

Keywords: Access control · Administration · Machine learning

1 Introduction

Machine Learning (ML) is used in the field of access control for different purposes such as policy mining [1], attribute engineering [3] and role mining [26]. In traditional access control systems such as RBAC [31] and ABAC [15], the access control decision engine decides accesses based on a written policy (or role assignments, in the case of RBAC). In recent years, researchers have proposed utilizing a trained

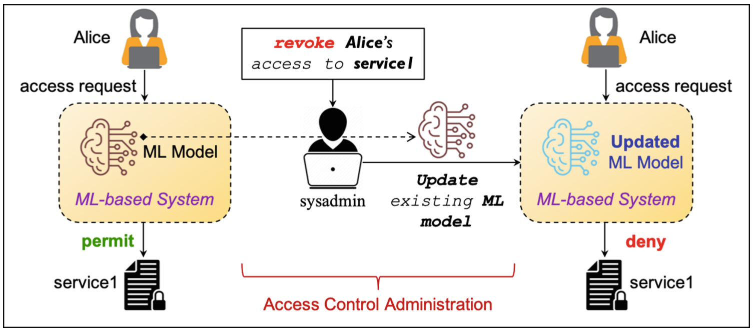


Fig. 1. Administration problem in ML-based access control system.

ML model to make access control decisions, possibly supplementing or even eventually replacing rule-based access control systems. We refer to such systems as machine learning based access control (MLBAC) [9,10,19,22,28,35]. We briefly discuss these methods in related work Sect. 2.2. These works have demonstrated that, for a given ground truth of an access control state represented in the form of authorization tuples, MLBAC models could capture that access control state with significantly higher accuracy than the traditional access control models such as ABAC. In addition, some works also demonstrate that MLBAC generalizes better than traditional approaches [9,10,19,28]. (Note that generalization is the ability of a model to make accurate decisions on users and resources not explicitly seen during policy mining or ML model training.) Even if MLBAC does not replace traditional forms of access control in practice, it could serve as an effective approach for access control monitoring/auditing or operate in tandem with traditional systems [24,28,38].

However, access control systems are not static—changes in access control state are inevitable. A user may be granted new permissions, or some of her current permissions could get revoked. As shown in Fig. 1, 'Alice' has access to the 'service1' resource. To revoke her access, the learned access control state in MLBAC will need to be correspondingly updated such that it can react accordingly to the applied change. This problem is referred to as *access control administration* in the access control domain [30]. Evidently, administration problems have been thoroughly investigated for traditional approaches [18,30,36], but the issue remains entirely unexplored for MLBAC.

Administration challenges could vary from model to model, but the problem's importance remains unchanged. In the case of RBAC, administration activities include assigning/removing permission to/from a role, creating a new role, and managing role hierarchy [30]. For ABAC, administration activities include updating user/resource attributes and policy modification [33]. In such traditional approaches, the changes are accomplished by modifying existing configurations such as written access control policies, and attribute and role assignments. However, in MLBAC, there is no notion of a human-readable written policy to update. If an access control state change is to be made, it requires modification of existing model. Such a modification is complicated as, in most cases, an ML model is a highly complex function, a tree, or even a black-box that a human user can not directly access and modify. Often, to capture changes, one must go through a process similar to the initial training process.

In this paper, we investigate the administration problem of MLBAC. In particular, we consider the situations where a trained ML could be either from *symbolic* (e.g., RF) or *non-symbolic* (e.g., neural network) types. The symbolic ML methods represent knowledge in the form of logic or a tree that distinguishes them from non-symbolic ones, either statistical or neural [9]. To the best of our knowledge, our proposed method is the first work towards administration in an ML-based access control system. We summarize our contributions as follows.

- We define MLBAC administration problem and propose a methodology to automate and systematize the MLBAC administration process.
- We develop two prototypes of administration in a system where access control decisions are made based on either symbolic or non-symbolic ML approaches.
- We thoroughly evaluate both prototypes for the efficacy of symbolic and non-symbolic ML approaches from an access control administration perspective.
- We demonstrate that administration in an MLBAC poses additional challenges and propose different techniques to overcome them.

The rest of the paper is organized as follows. Section 2 discusses related work. Section 3 presents an overview of the MLBAC administration, its requirements, and the administration methodology. In Sect. 4, we implement two MLBAC administration prototypes in a simulated system and evaluate those prototypes in Sect. 5. Finally, Sect. 6 concludes the paper.

2 Related Work

This section discusses works where ML algorithms are proposed in the context of an access control decision engine. One body of work in this category apply ML for the policy administration in traditional access control system [2,6,12]. Another body of work proposes an ML model (MLBAC) in place of traditional access control policies [9,10,19,22,28,35].

2.1 ML for Administration of Policy-Based Access Control

Researchers exploit the power of ML to administer changes in traditional policy-based access control systems. The authors in [2] develop an adaptive access control framework for the IoT domain using RF and Neural Networks. The proposed framework dynamically refines the access policies based on the access behaviors of the IoT devices. Argento et al. [6] propose an ML-based approach that identifies policy misconfigurations and adjusts policies at run-time. Gumma et al. [12] propose PAMMELA, an ML-based ABAC policy administration method that creates new rules for the proposed changes and extends existing policy.

2.2 MLBAC

In MLBAC, an ML model makes an access control prediction, which is then interpreted into a permit or deny decision. We briefly discuss them below.

Cappelletti et al. [9] train ML models based on user/resource attribute values and access request logs that subsequently decide access permissions. Specifically, the authors build a Decision Tree [29] and RF [8] classifier from the symbolic ML class and Support Vector Machines (SVM) [11] and Multi-Layer Perceptron (MLP) [32] from non-symbolic ones. The empirical results suggest that if the underlying access control state is complex, a symbolic ML approach could perform better. The authors highlight that a system is *complex* if the data (e.g., access logs) are not easily separable according to PCA and t-SNE visualization.

Chang et al. [10] propose an ML-based time-constraint access control where access policies are associated with the time (e.g., a user may only have access to a resource during office hours). The authors train an SVM using each user's login time and a password. For any access request, the trained SVM classifies the users into their respective groups (e.g., department) and provides desired security access right during that period.

Karimi et al. [19] develop an ABAC-RL framework to map between access requests and the access decisions (permit or deny). For deciding accesses, the ABAC-RL trains a reinforcement learning (RL) *agent* that adapts an ABAC policy via a feedback control loop by interacting with users and administrators.

Liu et al. [22] propose EPDE-ML transforming the access control 'permission decision' problem into an 'ML classification' problem that allows or denies accesses. EPDE-ML uses an RF to construct a vector decision classifier to establish a permission decision engine for making access decisions.

Our prior work [28] develops DLBAC using user/resource metadata and the existing access control state as authorization tuples. We build multiple deep neural networks, including ResNet [14], DenseNet [17], etc., that make more generalized and accurate access decisions than ABAC and other ML-based methods.

Srivastava et al. [35] develop risk adaptive access control (RAdAC) for dynamic access decisions (changes in accesses during run-time). For any access request, the RAdAC determines the genuineness of the user, measures the risk, and then provides access accordingly. The framework considers many dynamic attributes such as access time, location, user history, resource sensitivity, etc., and experiments using a neural network and an RF algorithm.

While developing MLBAC using either symbolic [9,22,35] or non-symbolic approaches [9,10,19,28,35], the administration problem is not investigated. In this work, we develop an administrative framework for MLBAC.

3 MLBAC Administration

Figure 2 illustrates the overview of MLBAC administration. As depicted in the figure, the *Admin Engine*, the administrative framework, takes a change request as the input, which we refer to as a *Task*. We aim to incorporate the requested Task in MLBAC administration. We assume that the Admin Engine has access to

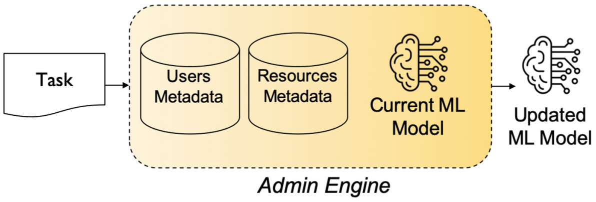


Fig. 2. Overview of MLBAC administration.

the user and resource metadata databases and the ML model, which is currently being used for decision making. We refer to this ML model as *Current ML Model* and denote as $\mathbb{F}_{\mathbf{current}}$. The objective of administration in MLBAC is to modify $\mathbb{F}_{\mathbf{current}}$ to capture the requested Task and generate an updated model. We also refer to this updated model as *Updated ML Model* and designate as $\mathbb{F}_{\mathbf{updated}}$.

3.1 Requirements

For the purpose of this paper, we generalize MLBAC as follows. An MLBAC model is trained using the existing access control state of a system, along with various pieces of available metadata values such as those of users and objects. Since the decision engine in MLBAC is an ML model, modifying its access control state is not as obvious as that of, say, ABAC, where a rule is typically adjusted to grant or deny existing accesses. It is often required to modify the model itself to accommodate any authorization-related changes. Consequently, the administrative tasks in MLBAC are somewhat simplified since we no longer need to worry about policy and attribute updates. In this paper, we focus on basic administrative tasks for MLBAC, including granting/revoking the access of one or multiple users to one or more resources.

Over time, by learning from proposed changes and observing the metadata of users and objects, MLBAC could intelligently adjust other "similar" accesses in the system. We believe smarter access control administration is one of the most significant benefits of MLBAC in practice, hence the focus of this work.

3.2 Problem Statement and Approach

In an ML model, to modify a piece of learned information, it is required to iteratively update the weights of its neurons (in the case of neural network) or parameters (for classical ML) starting with random initialization [37]. Consequently, we state the MLBAC administration problem as:

'Given an administrative task, update the MLBAC model's weights and/or parameters such that the updated model captures the desired changes in the access control state.'

By desired changes, we mean both the given administrative task and additional administrative tasks that are similar to the given task. The challenge specifically, then, is: what is the best approach to accurately learn to accommodate the given task and perform additional changes that are *similar* to the proposed task while keeping the existing access control state unchanged for all other users and resources that are vastly dissimilar? We indicate similar changes as changing access to other users and resources similar to the user and resource given in the proposed task. However, determining whether two users (or resources) are similar or not depends on the *type* of their metadata. If the metadata is real-valued (e.g., age, salary, etc.), it is possible to automatically determine other similar users and resources using distance measurement or clustering approaches [41]. This is also applicable for ordinal categorical values, where there is a notion of *order* among its values (e.g., degrees, clearances, job roles, etc.). However, in the case for nominal categorical values (e.g., department, expertise, etc.), there is no notion of order among them, and therefore one could only perform an equality check based on their values.

In practice, one could anticipate a mix of real-valued, ordinal categorical, and nominal categorical data. The model would automatically find additional similar administrative tasks for real-valued and ordinal categorical metadata values. For nominal categorical metadata values, we seek input from the system administrator (sysadmin) to determine the similarity between the user(resource) involved in the proposed task and other users(resources) in the system. We assume sysadmin will provide some 'similarity measurement criteria' in terms of user and resource metadata, which we refer to as *Criteria*. We further illustrate its syntax in Sect. 3.3.

3.3 Terminologies

This section introduces some terminologies that we repeatedly use for the explanation of administration in MLBAC.

- **Authorization Tuple.** An Authorization Tuple is user-permissions tuple $\langle user, resource, permissions \rangle$ that specifies the permissions of a *user* to a *resource*. For example, an Authorization Tuple $\langle u1, r1, \{op1, op3\} \rangle$ indicates that a user $u1$ has operations $op1$ and $op3$ access to a resource $r1$.
- **Task.** A Task is a change request which is expressed through a tuple of four elements $\langle user, resource, operation, access \rangle$, where the *access* could be either permit or deny. For example, a Task $\langle u1, r1, op3, deny \rangle$ is a request to revoke the $op3$ access of the user $u1$ to the resource $r1$.
- **AAT (Admin Authorization Tuple).** AAT is an updated Authorization Tuple generated from an existing one based on a Task. For example, $\langle u1, r1, \boldsymbol{op3, deny} \rangle$ is a given *Task*. Suppose that the existing Authorization Tuple in the system for user $u1$ and resource $r1$ is $\langle u1, r1, \{op1, \boldsymbol{op3}, op4\} \rangle$. The AAT with respect to the given Task would be $\langle u1, r1, \{op1, op4\} \rangle$. AAT, in effect, is the change that the admin seeks to make.
- **Criteria.** Criteria is defined as a tuple of user metadata name and value pairs and resource metadata name and value pairs that is expressed as:

$$\Big\langle \{umeta_0 \in \{val_0, \ldots, val_i\}, \ldots, umeta_m \in \{val_0, \ldots, val_j\}\},$$
$$\{rmeta_0 \in \{val_0, \ldots, val_k\}, \ldots, rmeta_n \in \{val_0, \ldots, val_l\}\} \Big\rangle$$

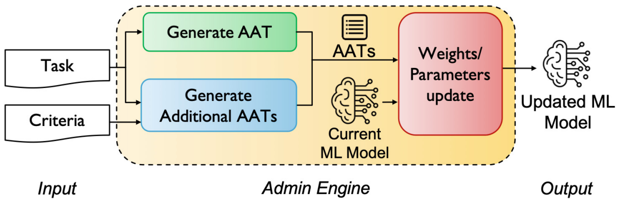


Fig. 3. Administration process flow in MLBAC.

where $\{umeta_0, \ldots, umeta_m\}$ is a set of user metadata names, $\{rmeta_0, \ldots, rmeta_n\}$ is a set of resource metadata names, and $\{val_0, \ldots\}$ indicates possible values for respective metadata. For example, a sample Criteria could be $\langle umeta1 \in \{val_0, val_1\}, rmeta4 \in \{val_2\}\rangle$. In this Criteria, the possible values of $umeta1$ are val_0 and val_1, and the possible value of $rmeta4$ is val_2.

- **Additional AAT.** The Additional AAT is a set of *similar* AAT determined based on users and resources similar to the user and resource in the Task. This paper determines similar users/resources based on the input Criteria. Section 3.4 discusses Additional AAT generation.
- **OATs (Other Authorization Tuples).** The OAT is a set of Authorization Tuples in the access control system that excludes AAT and Additional AAT.

3.4 Methodology

As shown in Fig. 3, the Admin Engine generates an Admin Authorization Tuple (AAT) for the given Task as described in Sect. 3.3. The AAT is the Authorization Tuple that we aim to integrate into the current ML model, $\mathbb{F}_{\mathbf{current}}$. Next, the Admin Engine generates Additional AAT based on the Task and Criteria. Both AAT and Additional AAT are independent of each other hence it is not required to maintain any specific order for their generation.

We generate Additional AAT based on a set of *similar* users and resources determined using Criteria, as discussed in Sect. 3.2. The Criteria consist of user and resource metadata names and value pairs. For each user, we compare their metadata values with respective metadata values stipulated in the Criteria. If it matches, we call the corresponding user as the *similar user*. Eventually, we determine all the similar users and follow the same approach for finding similar resources. These similar users and resources are the candidate users and resources for the Additional AAT generation. Next, we iterate over the list of candidate users and candidate resources to make user-resource pairs and determine a list of operations for each pair that the user has access to the resource and *update* their access according to the given Task. The user-resource pair with their updated access operation is an Additional AAT. Eventually, we obtain all the Additional AAT for the given Task and Criteria by repeating the same process. (Appendix A illustrates the pseudo-code of Additional AAT generation.) Collectively both AAT and Additional AAT are stored in a set that we refer to as **AATs**. Note that the 'size of AATs' indicates the number of elements in the set.

At this point, the Admin Engine has the input (AATs) to accommodate in its $\mathbb{F}_{\mathbf{current}}$ model and adjust the weights/ parameters to react accordingly for the newly added changes and produce the $\mathbb{F}_{\mathbf{updated}}$ model. This accommodation is not straightforward, and there are multiple underlying challenges. A naive solution to this problem could be to *retrain* an ML model based on newly generated AATs and original training data that we used to train the $\mathbb{F}_{\mathbf{current}}$. While this approach has its benefits, there are multiple shortcomings. For example, to retrain an ML model, one always has to maintain the original training dataset and the AATs of each administration. Also, retraining is expensive in terms of training time and resource consumption. Therefore, it might not be practical nor feasible for many systems to retrain an ML model to accommodate new changes.

A potential solution could be to update the weights/parameters of the $\mathbb{F}_{\mathbf{current}}$. However, the process of updating the weights/parameter values in an ML model has a direct correlation with the type of model in question. Suppose the underlying model is a classical ML algorithm such as SVM and Ensemble Methods. In that case, an incremental machine learning (a.k.a online learning) technique could be a prospective solution [7]. If the model is a neural network, a possible technique could be to use *fine-tuning* [20]. Fine-tuning performs internal adjustments to a trained neural network's (e.g., $\mathbb{F}_{\mathbf{current}}$ in MLBAC) weight based on a set of given examples (AATs in the case of MLBAC).

4 MLBAC Administration Prototype

This section implements two prototypes of MLBAC administration using ML models from symbolic and non-symbolic classes. We experiment with MLBAC administration to assess how well it reacts to the administrative changes. We apply MLBAC administration in a synthetically generated extensive system with thousands of users and resources. The following sections briefly introduce the simulated system, the ML models, and different administration strategies.

4.1 System for MLBAC Administration Experimentation

Access control administration is a continuous process where one could expect many change requests during the life of a system. A limited number of real-world access control-related datasets are available from Amazon [4,5]. These datasets have been used extensively in the literature for ML model training and ABAC policy mining and evaluating how accurately the trained model or mined policy can decide accesses [1,9,27,28]. For any access control administration experiment, we need a system where we will have continuous change requests during the system's life. The Amazon datasets in themselves do not provide such administrative tasks. Our prior work [28] provides a dataset[1] named $u5k\text{-}r5k\text{-}auth12k$ for a simulated access control system. We created the dataset using the data generation algorithm proposed by Xu et al. [39] (see Appendix B). The simulated

[1] https://github.com/dlbac/DlbacAlpha/tree/main/dataset/synthetic.

system has around five thousand users and five thousand resources. Also, there are eight user and eight resource metadata for each user and resource, respectively, and four operations. The dataset contains *nominal* categorical metadata values, as integers, of which each value denotes a category. (Section 3.2 briefly discussed nominal and ordinal categorical data.) When visualizing the dataset using t-SNE [23], we found that the samples overlap significantly and are not easily separable, indicating the simulated system is fairly complex [9,28] (see Appendix C). We train ML models for MLBAC using this dataset.

4.2 Symbolic and Non-symbolic ML Models

Among symbolic approaches, the RF algorithm got special attention in the access control domain due to its expressiveness of a decision in the form of a rule [9,22,35]. RF can achieve excellent performance in capturing the access control state of a system. However, if the access control state of the underlying system is complicated, a non-symbolic method such as a neural network-based system shows superior performance compared to the symbolic ones [9,28]. In this work, we develop an MLBAC administration prototype with an RF from the symbolic class to determine its efficiency from an administration perspective, which we refer to as RF-MLBAC. We also investigate another prototype with the neural network from the non-symbolic type. In particular, we consider ResNet [14] as our candidate neural network and refer to it as ResNet-MLBAC. We note that one could use other neural networks, including MLP [32], DenseNet [17], etc., although we do not anticipate any significant changes in our results.

Both RF and ResNet in MLBAC take user/resource metadata values as input to make corresponding access control decisions. Since the metadata values in our dataset are categorical, we encode them before applying them to the model [13]. Our experiment's ResNet architecture has a depth of 8, and the RF has 100 estimators (decision trees in the forest). As the dataset has four different operations, both models output the *probability* of granting the permission for a related operation. Given a feature vector x of the user and resource metadata, the ML model is defined as a prediction function f: $\hat{y} = f(x)$, where $\hat{y}$ is the predicted label or permission (grant (1) or deny (0)) of the operation *op*, obtained from comparing the probability of granting the permission from the output of the ML model with a *threshold*. We consider a threshold of 0.5 for our experiment.

4.3 Administration Strategies in MLBAC

We follow multiple strategies for accommodating a given Task in MLBAC. From a Task perspective, we propose single-Task and multi-Task administration approaches. Also, we examine two learning strategies that include retraining and sequential learning. We discuss them below.

Single-Task Administration. We administer each Task individually and replace the current ML model ($\mathbb{F}_{\mathbf{current}}$) with the updated model ($\mathbb{F}_{\mathbf{updated}}$).

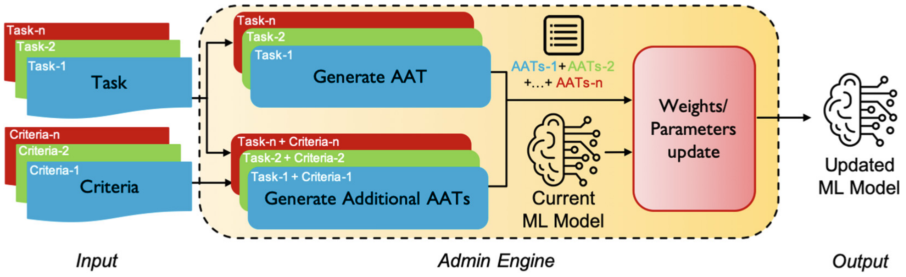


Fig. 4. Multi-task administration process flow in MLBAC.

In this case, we simulate that the sysadmin updates the underlying ML model after receiving any new Task. For single-Task administration, the Admin Engine determines AATs (i.e., both AAT and Additional AAT) for the given Task and then apply the generated AATs for the administration. Figure 3 is an illustration of single-Task administration.

Multi-task Administration. In practice, sysadmin may receive multiple unique Tasks together, or they may wait for additional Tasks to accumulate before initiating an administration. To simulate this, we investigate administrations with multiple simultaneous Tasks in MLBAC, which we refer to *multi-Task administration*. In this case, Admin Engine determines AATs for each Task individually and combines them. Then, the combined AATs are used for the administration. We use the term *Task count* to refer to the number of Tasks we consider for a multi-Task administration. Figure 4 illustrates the multi-Task administration process for n-Tasks. We experiment with 2-Tasks, 3-Tasks, and 6-Tasks administration for multi-Task administration with Task counts 2, 3, and 6.

Retraining. A naive solution to the administration problem could be to *retrain* an ML model based on newly generated AATs and initial training data that we used to train the $\mathbb{F}_{\mathbf{current}}$. The idea is to train a fresh model from scratch with the dataset that combines both initial training data and the samples generated for the respective Task (AATs in case of MLBAC)), as shown in Fig. 5 (left). The trained model will replace the existing $\mathbb{F}_{\mathbf{current}}$ model.

Retraining may not be feasible or practical for many systems due to some reasons. For instance, this process requires storing the *entire* initial training data for future administration. Also, retraining is expensive in terms of computation time and resource consumption. Model training is one of the most time-consuming parts of ML-based applications. One needs to spend the same amount of time to accommodate any new change in an existing system. Besides, retraining an ML model produces a new model that does not hold any previous history of access change and only portrays the data provided during training.

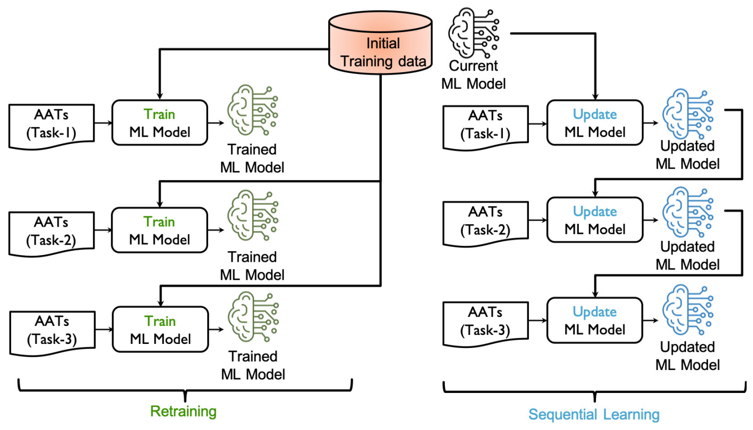


Fig. 5. Retraining (left) vs. sequential learning (right) strategies.

Sequential Learning. Sequentially learning Tasks is vital for developing an autonomous system. It reflects how a human learner identifies the materials to be learned [16, 25]. We also embrace this learning phenomenon to administer MLBAC, as illustrated in Fig. 5 (right). As shown, for Task-1, the Admin Engine utilizes the $\mathbb{F}_{\mathbf{current}}$ and AATs of Task-1 to update the existing model and generate $\mathbb{F}_{\mathbf{updated}}$ model. This $\mathbb{F}_{\mathbf{updated}}$ model replaces the $\mathbb{F}_{\mathbf{current}}$ model for deciding accesses and acts as input for the Task-2 administration.

Sequential Learning Process and Its Effect. For sequential administration in RF, we append additional estimators that learn new changes while keeping existing estimators untouched. Even though this method is not as efficient as incremental learning in other classical approaches, we did not find any better strategy for RF in the literature. For each single-Task administration, we append two estimators in the RF model. We append 5, 8, and 10 estimators for 2-Tasks, 3-Tasks, and 6-Tasks administration, respectively. Note that we performed trial and error with more estimators; however, increasing this number of estimators was efficient in performance and model size. On the other hand, for ResNet, we employ the *fine-tuning* technique (discussed in Sect. 3.4) to incrementally learn new changes and update the network's weights accordingly.

However, one of the major challenges of sequential learning is maintaining the existing access control state unchanged. An ML model could *forget* previous knowledge while learning new information. For example, Alice has read and write access to a resource projectA, and Bob has only read access to another resource projectB. Sysadmin received a Task '*to permit Bob with the write access to projectB*'. After administering the requested Task, the system correctly updates the access control state such that Bob has both read and write access to the projectB. However, there might be a case that the updated system could not make the correct access decision for Alice to the projectA. In other words, the system *forgot* Alice's access to projectA. Formally, in ML arena, this phenomenon is known as *catastrophic forgetting* [21,40]. In the case of RF, this is not a problem as the technique we followed does not modify existing estimators in the model but append new ones. However, this is a significant challenge for neural networks since the knowledge of the previously learned Task(s) starts decaying with the incorporation of the new Task [21].

Overcoming Catastrophic Forgetting in MLBAC. Catastrophic forgetting is a well-known problem in machine learning while updating the model, especially when dealing with a neural network. Fortunately, this is a well-studied problem in ML literature, and different approaches have been proposed to overcome this hurdle [16,34,40]. One of the common strategies is to replay previous knowledge in the form of training data (the dataset used to train the network) with new samples (AATs in MLBAC) during fine-tuning [34]. It may not be practical to store the training data in many applications if the samples are too large (e.g., image, video, etc.). However, this is not an issue for MLBAC as it works based on numerical user and resource metadata and attributes values. In our simulated system, each training sample is a *vector* of user and resource numerical metadata. Other real-world datasets [27] are also similar, which indicates the feasibility of storing the prior training data for MLBAC. To minimize the required storing space, we reserve a quarter of all the original training samples instead of keeping the entire training dataset, which we refer to as *Replay Data*. The Admin Engine can access the Replay Data and add them during administration along with AATs for the correspondent Task. After performing an administration, we append a quarter of AATs to the Replay Data, which reflects the current task's information during the next administration. Admin Engine ensures that the AATs and Replay Data are *mutually exclusive*.

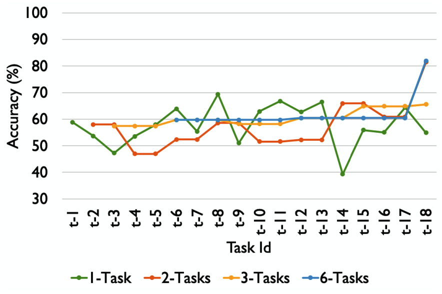
5 Evaluation

This section measures the performance for our simulated system to determine the feasibility and efficacy of the proposed strategies for MLBAC administration.

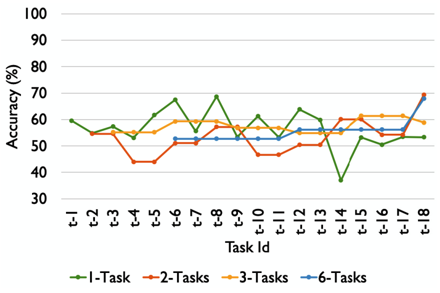
5.1 Evaluation Methodology

We experiment and evaluate administration performance in RF-MLBAC and ResNet-MLBAC prototypes on $u5k$-$r5k$-$auth12k$ dataset. The dataset has around twelve thousand samples (Authorization Tuples). We use 80% of the samples for training and 20% for testing the ML models. After initial training, the trained models' (RF and ResNet) performances are above 99% for the test samples, respectively, implying that $\mathbb{F}_{\mathbf{current}}$ is highly accurate in making access decisions.

We create a set of Tasks to simulate that the sysadmin received all those Tasks one by one during the system's life span. To that extent, we randomly



(a) Retraining



(b) Sequential Learning

Fig. 6. AATs performance in RF-MLBAC. (Color figure online)

construct eighteen distinct Tasks from the $u5k$-$r5k$-$auth12k$ dataset with different kinds of Criteria (Appendix D). For example, the Task Id *t-1* indicates the first Task and a Task: $\langle uid = 259, rid = 112, op3, permit\rangle$ means a user with $uid = 259$ needs *op3* access to a resource with $rid = 112$. Also, the Criteria: $\langle umeta0 \in \{9\}, umeta6 \in \{6\}, rmeta0 \in \{9\}, rmeta3 \in \{46\}\rangle$ specifies the user whose *umeta0* and *umeta6* metadata values are 6 and 9, respectively, could have *op3* access to resources with *rmeta0* and *rmeta3* metadata values 9 and 46, respectively. Besides, based on the Task and Criteria, the number of generated Additional AAT is 42, and combining the AAT gives an AATs of size 43. We ensure that every Task is independent concerning its change request and purpose. While updating the model, we use 80% of the AATs for training and 20% for testing the updated ML model. We have created a repository on GitHub consisting of the source code, dataset, and respective AATs, OATs, and ReplayData for each Task.[2]

We evaluate the administration performance in terms of accuracy. We define the *accuracy* as the measure (in percentage) of correct access authorization for a user to a resource with respect to the actual access control state (ground truth).

5.2 Results

To evaluate the administration performance in RF-MLBAC and ResNet-MLBAC, we assess how accurately the $\mathbb{F}_{\mathbf{updated}}$ model can capture the AATs (both AAT and Additional AAT). We also evaluate how well it can preserve the access control state of all other users and resources (OATs as described in Sect. 3.3).

We experiment and evaluate the performance for all eighteen Tasks with single-Task and multi-Task administrations. For multi-Task administration, we consider 2-Tasks, 3-Tasks, and 6-Tasks. For example, a 3-Tasks administration indicates, we use three different Tasks together for an administration. In the single-Task administration, it requires *eighteen* different administration to

[2] https://github.com/dlbac/MLBAC-Administration.

accomplish all the 18 Tasks as it administer one Task at a time. For multi-Task administration, the number of administration reduces with an increase in Task count. For example, for 3-Tasks administration, it requires six different multi-Task administrations to finish all the eighteen Tasks.

Administration Performance in RF-MLBAC

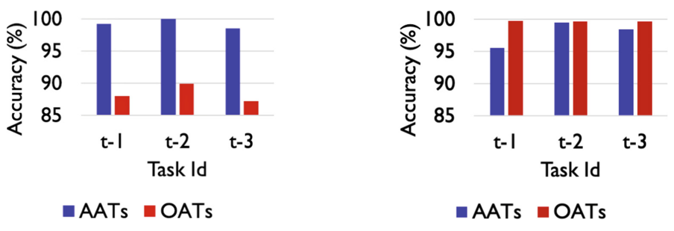
Retraining. As discussed, retraining is a naive approach and inefficient for both data and computation. We evaluate the retraining performance in the $\mathbb{F}_{\mathbf{updated}}$ model. We assess both AATs and OATs performance for single-Task and multi-Task (2, 3, and 6 Tasks) administrations. For OATs, the $\mathbb{F}_{\mathbf{updated}}$ model is as accurate as of the initial trained model with more than 99% accuracy. This performance is consistent across all the single and multi-Tasks administrations. However, in the case of AATs, as demonstrated in Fig. 6a, the accuracy range is 40% to 80%. In almost all the circumstances, the accuracy is inconsistent except for six-Tasks administration that shows a better and more persistent result across administrations with 60% accuracy. Overall, the low AATs performance indicates that the RF model can capture only a portion of new changes while accommodating the proposed Task.

Sequential Learning. We perform the same evaluation for sequential learning and apply both single and multi-Tasks administrations. Similar to retraining, the OATs performance is over 99% across administrations and consistent, indicating that the RF model could preserve the existing access control state better. This excellent result indicates that the RF model did not forget the initial access control state. This result is anticipated because RF-MLBAC appends new estimators to comprehend proposed changes instead of modifying their existing estimators while learning new changes.

However, we see a very opposite scenario in AATs performance. As shown in Fig. 6b, the accuracy of AATs varies in the range of 40% to 70%. For single-Task and two-Tasks administrations (green and orange lines in the figure), the AATs performance seems highly inconsistent compared to what we see for six-Tasks administration (blue line). However, the accuracy of six-Tasks administration is about 55%, which indicates that considering many Tasks together for a single administration may not provide a better result. On the contrary, the three-Tasks administration shows a consistent performance with around 60% accuracy across Tasks, suggesting an RF model's multi-Tasks administration with around three Tasks could be a potential administration to consider.

Administration performance in ResNet-MLBAC

Sequential Learning. Training a neural network from scratch is computationally costly, which is neither efficient nor feasible in access control. As a result, we did not take the naive (retraining) approach for this prototype. To begin with administration in ResNet-MLBAC, we administer the first three Tasks (t-1 to t-3) as a single-Task administration without providing any Replay Data to see the impact of sequential learning in the existing access control state. As



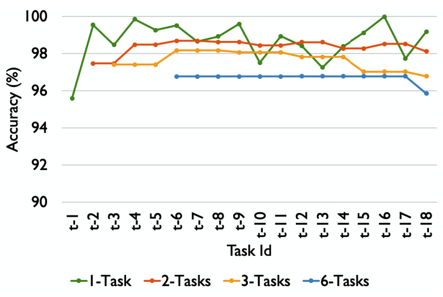
(a) Without Replay Data (b) With Replay Data

Fig. 7. Administration Performance in ResNet-MLBAC for Sequential Learning.

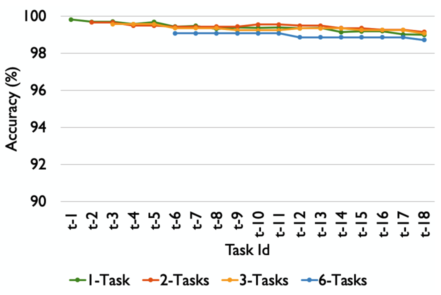
shown in Fig. 7a, the updated network captures the authorization for AATs with excellent accuracy, as opposed to what we observed in RF-MLBAC. However, for OATs, we see a lower accuracy (below 90%) in all three cases, indicating the network forgot (catastrophic forgetting) the access control state of a good amount of existing users and resources while learning new information. To overcome that, we apply Replay Data (as discussed in Sect. 4.3) along with AATs during administration. Figure 7b illustrates that combining Replay Data with AATs helps ResNet-MLBAC administration to significantly reduce the catastrophic forgetting. As shown in the figure, the accuracy of OATs is now above 99% across all three Tasks. Such a significant increase in OAT performance implies the remarkable impact of Replay Data in overcoming catastrophic forgetting.

Further, we experiment with all eighteen Tasks for single and multi-Task administrations. Figure 8a and Fig. 8b demonstrate the performance of AATs and OATs, respectively. As illustrated, the AATs performance range is 96% to 99% accuracy, significantly better than what we observed in the RF-MLBAC. As we see in the figures, for both AATs and OATs, the performance of 6-Tasks administration is low compared to other multi-Task administrations. Such inferior results indicate the infeasibility of using many Tasks under a multi-Task administration in ResNet-MLBAC administration. Similarly, the performance of AATs in single-Task administration is inconsistent across all the Tasks, which signifies that using single-Task administration in ResNet-MLBAC may produce very unstable performance. However, for both 2-Tasks and 3-Tasks multi-Task administrations, the result is persistent and progressing for both AATs and OATs, thereby proving feasible and better for ResNet-MLBAC administration.

Summary of Performance in RF-MLBAC and ResNet-MLBAC. Based on AATs and OATs performance in both prototypes, it is evident that the RF-MLBAC shows better results in preserving the existing access control state. It was expected as we did not change the estimators of the initial RF model. However, the AATs performance in RF-MLBAC is extremely low, indicating it could not well capture proposed changes. Besides, it has other drawbacks from model size and optimization perspectives. The size of the trained model gradually grows with the addition of new estimators in it. Also, each administration needs trial



(a) AATs Performance



(b) OATs Performance

Fig. 8. Performance in ResNet-MLBAC.

and error to determine an optimal number of new estimators to append that correlates with how many Task one consider for a multi-Task administration. On the other hand, ResNet-MLBAC administration shows a better accuracy for AATs and comparable performance in OATs, indicating it could better capture proposed changes while retaining the existing access control state intact.

Administration Cost Evaluation in RF-MLBAC and ResNet-MLBAC We evaluate administration costs for both administration prototypes concerning computation time for an administration. We observe that RF-MLBAC administration generally takes less than a second to complete an administration (considering a maximum of 18 Tasks for an administration). On the contrary, ResNet-MLBAC administration is slower than RF-MLBAC and varies with the increase of Tasks count for each administration, as depicted in Fig. 9. We measure the computation cost of single-Task and multi-Task administrations in ResNet-MLBAC to identify how administration time varies with the increase in *Task count* and how many Tasks are feasible for a single administration. We consider different sizes (1, 3, 6, 9, 12, 15, and 18 Tasks) multi-Task administration.

As shown in Fig. 9, it needs 10 s for a single-Task administration, which is the same for 3, 6, and 9-Tasks administrations, and becomes double for 12, 15, and 18-Tasks administrations. From a performance perspective, for 3 and 6-Tasks administrations, we see a balanced accuracy in AATs and OATs. Such results suggest the feasibility of using multi-Task administration in ResNet-MLBAC. However, considering many Tasks together (e.g., 12 or 15-Tasks administration) decreases the OATs performance, justifying the infeasibility of administering too many Tasks together under a single administration.

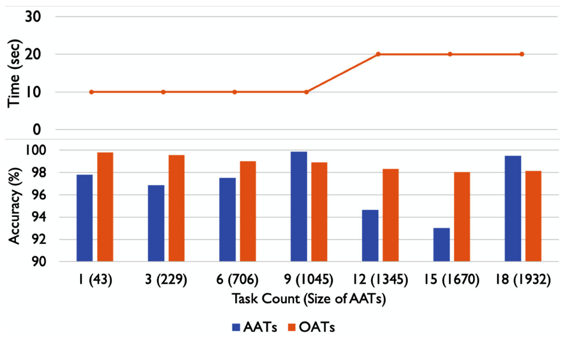


Fig. 9. Administration cost in ResNet-MLBAC for sequential learning.

6 Conclusion

This paper explores the access control administration problem for MLBAC. We review administration requirements in the same context and propose an administrative framework for MLBAC administration. We implement two prototypes, RF-MLBAC and ResNet-MLBAC, on a simulated system applying ML algorithms from symbolic and non-symbolic classes. Due to the uniqueness of the MLBAC administration problem, there are many underlying challenges, such as insufficient learning of the proposed changes and forgetting existing access information. We propose different strategies to overcome them. Also, we thoroughly evaluate both prototypes. Our empirical results summarize that the non-symbolic approach performs better than the symbolic one while adjusting for new changes in MLBAC administration.

Acknowledgements. 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.

A Additional AAT Generation

We depict the pseudo-code of Additional AAT generation in Algorithm 1.

B Data Generation

In our earlier work [28], we simulate a system using the data generation approach proposed by Xu et al. [39]. The algorithm generates a set of attributes for users and resources and a set of *rules* based on those attributes. Each *rule* is a tuple of the form $\langle UAE, RAE, OP, C \rangle$, where UAE is the user attribute expression, RAE is the resource attribute expression, C is the set of constraints, and OP is a set

Algorithm 1: Additional AAT Generation

```
inputs: Task, Criteria
output: Additional AAT
data: uList[ ] and rList[ ] are the list of all users and resources,
      respectively, with their metadata values in the system
generateAdditionalAAT (Task, Criteria)
    additionalAAT[ ] = ∅//An empty list of Additional AAT
    //The extractOperationAccess is a utility function that takes a Task,
    //and returns operation and access from the Task
    $operation_t$, $access_t$ = extractOperationAccess(Task)
    //getSimilarUsers and getSimilarResources are the utility functions
    //that take list of all users and resources, respectively,
    //and the Criteria, and return similar users and resources
    candidateUsers[ ] = getSimilarUsers(uList, Criteria)
    candidateResources[ ] = getSimilarResources(rList, Criteria)
    //getAccessOperations is a utility function that takes a user and
    //resource, returns user's set of access operations to the resource
    foreach $user_c$ in candidateUsers do
        foreach $resource_c$ in candidateResources do
            $Op_c$[ ] = getAccessOperations($user_c$, $resource_c$)
            if $Op_c \neq \emptyset$ then
                if ($access_t$ = permit AND $operation_t$ not in $Op_c$) OR
                  ($access_t$ = deny AND $operation_t$ in $Op_c$) then
                    cAAT [ ] = ∅
                    cAAT.append($user_c$)
                    cAAT.append($resource_c$)
                    if $access_t$ = permit then
                        $Op_c$.append($operation_t$)
                    else
                        $Op_c$.remove($operation_t$)
                    end
                    cAAT.append($Op_c$)
                    additionalAAT.append(cAAT)
                end
            end
        end
    end
    return additionalAAT
```

of operations. The attribute expressions express the sets of users and resources to which a rule applies. 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 meet the constraint stated in the rule. 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.

Finally, we create (or update if it already exists) an Authorization Tuple with a new operation for each generated user, resource, and operation combination that satisfies a rule. Each user and resource has eight user metadata and eight resource metadata in the simulated system. Also, there are four different operations (*op1, op2, op3, op4*) in the system, where a user could have access to one or more operations to a resource. For example, an Authorization Tuple for a user with uid = 101 and a resource with rid = 212 with {op1, op3} access is mapped in the dataset as $\langle$101|212|*3 9 2 6 13 19 30 55* | *10 21 78 3 9 13 29 23* |$\langle$*1 0 1 0*$\rangle\rangle$, where *3 9 2 6 13 19 30 55* and *10 21 78 3 9 13 29 23* represents user's and resource's eight metadata values, respectively.

C Dataset Visualization

We visualize the $u5k$-$r5k$-$auth12k$ dataset using the t-SNE plot [23]. We project 16 user-resource metadata to 2-dimensional feature space and plot them in Fig. 10. The plot's samples (represented by dots) overlap significantly. Users with similar metadata values have different access and are not easily separable.

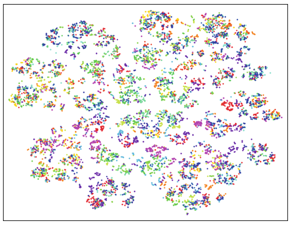


Fig. 10. t-SNE visualization of the $u5k$-$r5k$-$auth12k$ Dataset.

D List of Simulated Task and Criteria

We simulate eighteen distinct Tasks from the $u5k$-$r5k$-$auth12k$ dataset with different kinds of Criteria and report them in Table 1.

Table 1. List of task and criteria.

Task Id	Task	Criteria	Size of AATs
t-1	$\langle uid = 259, rid = 112, op3, permit \rangle$	$\langle umeta0 \in \{9\}, umeta6 \in \{6\}, rmeta0 \in \{9\}, rmeta3 \in \{46\} \rangle$	43
t-2	$\langle uid = 4624, rid = 4634, op4, deny \rangle$	$\langle umeta2 \in \{58, 49\}, umeta3 \in \{39\}, rmeta3 \in \{39\} \rangle$	94
t-3	$\langle uid = 1992, rid = 1858, op1, permit \rangle$	$\langle umeta2 \in \{11\}, rmeta2 \in \{11\}, rmeta3 \in \{48, 91\} \rangle$	92
t-4	$\langle uid = 5049, rid = 5177, op4, permit \rangle$	$\langle umeta1 \in \{6\}, umeta4 \in \{47, 71\}, rmeta1 \in \{6\} \rangle$	215
t-5	$\langle uid = 2034, rid = 2041, op2, deny \rangle$	$\langle umeta4 \in \{10\}, rmeta1 \in \{6, 10\}, rmeta4 \in \{10\} \rangle$	75
t-6	$\langle uid = 1348, rid = 1083, op2, permit \rangle$	$\langle umeta3 \in \{46, 50, 53\}, umeta5 \in \{13\}, rmeta3 \in \{46, 50, 53\}, rmeta5 \in \{13\} \rangle$	187
t-7	$\langle uid = 1345, rid = 1092, op4, permit \rangle$	$\langle umeta0 \in \{24, 64\}, umeta6 \in \{7\}, rmeta0 \in \{24, 64\}, rmeta6 \in \{7\} \rangle$	139
t-8	$\langle uid = 442, rid = 580, op3, permit \rangle$	$\langle umeta3 \in \{49\}, umeta5 \in \{47, 111\}, rmeta5 \in \{47, 111\}, rmeta7 \in \{49\} \rangle$	134
t-9	$\langle uid = 2599, rid = 2593, op1, permit \rangle$	$\langle umeta0 \in \{11\}, umeta1 \in \{17\}, rmeta0 \in \{11\}, rmeta1 \in \{17\} \rangle$	66
t-10	$\langle uid = 4112, rid = 1241, op2, permit \rangle$	$\langle umeta1 \in \{18\}, rmeta1 \in \{18\}, rmeta3 \in \{45, 47, 113\} \rangle$	75
t-11	$\langle uid = 2135, rid = 4875, op3, deny \rangle$	$\langle umeta2 \in \{13\}, umeta4 \in \{71, 96\}, rmeta2 \in \{13\}, rmeta4 \in \{71, 96\} \rangle$	118
t-12	$\langle uid = 660, rid = 560, op1, permit \rangle$	$\langle umeta3 \in \{88\}, umeta5 \in \{48, 111\}, rmeta5 \in \{48, 111\}, rmeta7 \in \{88\} \rangle$	107
t-13	$\langle uid = 2019, rid = 2056, op2, deny \rangle$	$\langle umeta4 \in \{12\}, rmeta1 \in \{78, 82\}, rmeta4 \in \{12\} \rangle$	121
t-14	$\langle uid = 1228, rid = 1088, op1, permit \rangle$	$\langle umeta2 \in \{11, 63\}, umeta5 \in \{20\}, rmeta5 \in \{20\} \rangle$	97
t-15	$\langle uid = 2825, rid = 3044, op2, permit \rangle$	$\langle umeta6 \in \{8\}, rmeta1 \notin \{6, 10\}, rmeta2 \in \{61, 62\}, rmeta6 \in \{8\} \rangle$	107
t-16	$\langle uid = 965, rid = 861, op4, permit \rangle$	$\langle umeta3 \in \{45\}, umeta7 \in \{20\}, rmeta3 \in \{45\}, rmeta6 \in \{20\} \rangle$	63
t-17	$\langle uid = 3745, rid = 3843, op3, permit \rangle$	$\langle umeta0 \in \{31\}, umeta6 \in \{2, 5, 9, 18\}, umeta7 \in \{4, 13\}, rmeta0 \in \{31\} \rangle$	83
t-18	$\langle uid = 2488, rid = 2495, op3, permit \rangle$	$\langle umeta1 \in \{58\}, rmeta1 \in \{58\}, rmeta2 \in \{58, 61\} \rangle$	116

References

1. Abu Jabal, A., et al.: Polisma - a framework for learning attribute-based access control policies. In: Chen, L., Li, N., Liang, K., Schneider, S. (eds.) ESORICS 2020. LNCS, vol. 12308, pp. 523–544. Springer, Cham (2020). https://doi.org/10.1007/978-3-030-58951-6_26
2. Alkhresheh, A., Elgazzar, K., Hassanein, H.S.: Adaptive access control policies for IoT deployments. In: IEEE IWCMC (2020)
3. Alohaly, M., Takabi, H., Blanco, E.: A deep learning approach for extracting attributes of ABAC policies. In: ACM SACMAT (2018)
4. Amazon, K.: Amazon employee access challenge in Kaggle (2013). https://www.kaggle.com/c/amazon-employee-access-challenge/
5. Amazon, U.: Amazon access samples data set (2011). http://archive.ics.uci.edu/ml/datasets/Amazon+Access+Samples
6. Argento, L., Margheri, A., Paci, F., Sassone, V., Zannone, N.: Towards adaptive access control. In: Kerschbaum, F., Paraboschi, S. (eds.) DBSec 2018. LNCS, vol. 10980, pp. 99–109. Springer, Cham (2018). https://doi.org/10.1007/978-3-319-95729-6_7
7. Benczúr, A.A., et al.: Online machine learning in big data streams. arXiv (2018)
8. Breiman, L.: Random forests. Mach. Learn. **45**, 5–32 (2001). https://doi.org/10.1023/A:1010933404324
9. Cappelletti, L., Valtolina, S., Valentini, G., et al.: On the quality of classification models for inferring ABAC policies from access logs. In: IEEE Big Data (2019)
10. Chang, C.C., Lin, I.C., Liao, C.T.: An access control system with time-constraint using support vector machines. Int. J. Netw. Secur. **2**(2), 150–159 (2006)
11. Cortes, C., Vapnik, V.: Support-vector networks. Mach. Learn. **20**, 273–297 (1995). https://doi.org/10.1007/BF00994018
12. Gumma, V., Mitra, B., Dey, S., Patel, P.S., Suman, S., Das, S.: PAMMELA: policy administration methodology using machine learning. arXiv (2021)
13. Hancock, J.T., Khoshgoftaar, T.M.: Survey on categorical data for neural networks. J. Big Data **7**(1), 1–41 (2020). https://doi.org/10.1186/s40537-020-00305-w

14. He, K., et al.: Deep residual learning for image recognition. In: IEEE CVPR (2016)
15. Hu, V.C., Ferraiolo, D., et al.: Guide to attribute based access control (ABAC) definition and considerations (draft). NIST Special Publication (2013)
16. Hu, W., et al.: Overcoming catastrophic forgetting for continual learning via model adaptation. In: ICLR (2018)
17. Huang, G., Liu, Z., Van Der Maaten, L., Weinberger, K.Q.: Densely connected convolutional networks. In: IEEE CVPR (2017)
18. Jha, S., Sural, S., Atluri, V., Vaidya, J.: An administrative model for collaborative management of ABAC systems and its security analysis. In: IEEE CIC (2016)
19. Karimi, L., Abdelhakim, M., Joshi, J.: Adaptive ABAC policy learning: a reinforcement learning approach. arXiv (2021)
20. Kaya, A., et al.: Analysis of transfer learning for deep neural network based plant classification models. Comput. Electron. Agric. **158**, 20–29 (2019)
21. Kirkpatrick, J., Pascanu, R., Rabinowitz, N., Veness, J., et al.: Overcoming catastrophic forgetting in neural networks. National Academy of Sciences (2017)
22. Liu, A., Du, X., Wang, N.: Efficient access control permission decision engine based on machine learning. Secur. Commun. Netw. **2021** (2021)
23. Van der Maaten, L., Hinton, G.: Visualizing data using t-SNE. JMLR **9**(11)(2008)
24. Martin, E., Xie, T.: Inferring access-control policy properties via machine learning. In: IEEE POLICY (2006)
25. McCloskey, M., Cohen, N.J.: Catastrophic interference in connectionist networks: the sequential learning problem. In: Psychology of Learning and Motivation (1989)
26. Ni, Q., Lobo, J., Calo, S., Rohatgi, P., Bertino, E.: Automating role-based provisioning by learning from examples. In: ACM SACMAT (2009)
27. Nobi, M.N., Gupta, M., Praharaj, L., Abdelsalam, M., Krishnan, R., Sandhu, R.: Machine learning in access control: a taxonomy and survey. arXiv (2022)
28. Nobi, M.N., Krishnan, R., Huang, Y., Shakarami, M., Sandhu, R.: Toward deep learning based access control. In: ACM CODASPY (2022)
29. Safavian, S.R., Landgrebe, D.: A survey of decision tree classifier methodology. IEEE Trans. Syst. Man Cybern. **21**, 660–674 (1991)
30. Sandhu, R., Munawer, Q.: The ARBAC99 model for administration of roles. In: IEEE ACSAC (1999)
31. Sandhu, R.S., et al.: Role-based access control models. Computer **29**, 38–47 (1996)
32. Schmidhuber, J.: Deep learning in neural networks: an overview. Neural Netw. **61**, 85–117 (2015)
33. Servos, D., Osborn, S.L.: Current research and open problems in attribute-based access control. ACM Comput. Surv. (CSUR) **49**, 1–45 (2017)
34. Shin, H., et al.: Continual learning with deep generative replay. arXiv (2017)
35. Srivastava, K., Shekokar, N.: Machine learning based risk-adaptive access control system to identify genuineness of the requester. In: Gunjan, V.K., Zurada, J.M., Raman, B., Gangadharan, G.R. (eds.) Modern Approaches in Machine Learning and Cognitive Science: A Walkthrough. SCI, vol. 885, pp. 129–143. Springer, Cham (2020). https://doi.org/10.1007/978-3-030-38445-6_10
36. Stoller, S.D.: An administrative model for relationship-based access control. In: Samarati, P. (ed.) DBSec 2015. LNCS, vol. 9149, pp. 53–68. Springer, Cham (2015). https://doi.org/10.1007/978-3-319-20810-7_4
37. Tajbakhsh, N., Shin, J.Y., et al.: Convolutional neural networks for medical image analysis: full training or fine tuning? IEEE Trans. Med. Imaging **35**, 1299–1312 (2016)
38. Xiang, C., Wu, Y., Shen, B., Shen, M., et al.: Towards continuous access control validation and forensics. In: CCS. ACM (2019)

39. Xu, Z., Stoller, S.D.: Mining attribute-based access control policies. TDSC **12**, 533–545 (2014)
40. Yoon, J., Yang, E., Lee, J., Hwang, S.J.: Lifelong learning with dynamically expandable networks. arXiv (2017)
41. Zhang, Y., Cheung, Y.M.: An ordinal data clustering algorithm with automated distance learning. In: AAAI Conference on Artificial Intelligence (2020)