Minimizing Privilege Assignment Errors in Cloud Services

Matthew W Sanders
Colorado School of Mines, Golden, CO
mwsanders@mines.edu

Chuan Yue
Colorado School of Mines, Golden, CO
chuanyue@mines.edu

ABSTRACT

The Principle of Least Privilege is a security objective of granting users only those accesses they need to perform their duties. Creating least privilege policies in the cloud environment with many diverse services, each with unique privilege sets, is significantly more challenging than policy creation previously studied in other environments. Such security policies are always imperfect and must balance between the security risk of granting over-privilege and the effort to correct for under-privilege. In this paper, we formally define the problem of balancing between over-privilege and under-privilege as the Privilege Error Minimization Problem (PEMP) and present a method for quantitatively scoring security policies. We design and compare three algorithms for automatically generating policies: a naive algorithm, an unsupervised learning algorithm, and a supervised learning algorithm. We present the results of evaluating these three policy generation algorithms on a real-world dataset consisting of 5.2 million Amazon Web Service (AWS) audit log entries. The application of these methods can help create policies that balance between an organization's acceptable level of risk and effort to correct under-privilege.

ACM Reference format:
Matthew W Sanders and Chuan Yue. 2018. Minimizing Privilege Assignment Errors in Cloud Services. In *Proceedings of Eighth ACM Conference on Data and Application Security and Privacy, Tempe, AZ, USA, March 19–21, 2018 (CODASPY '18),* 12 pages.
DOI: 10.1145/3176258.3176307

1 INTRODUCTION

Cloud computing has revolutionized the information technology industry. Organizations leverage cloud computing to deploy IT infrastructure that is resilient, affordable, and massively scalable with minimal up-front investment. Small startups can rapidly move from an idea to commercial operations and large enterprises can benefit from an elastic infrastructure that scales with unpredictable demand. Because of these benefits, cloud providers have seen significant growth recently with cloud computing industry revenue up 25% in 2016 totaling $148 billion [27]. Despite the wide adoption of cloud computing, there are still significant issues regarding security and usability that must be addressed. Privilege management is one such security and usability issue.

CODASPY '18, Tempe, AZ, USA

DOI: 10.1145/3176258.3176307

The principle of least privilege requires every privileged entity of a system to operate using the minimal set of privileges necessary to complete its job [19], and is considered a fundamental access control principle in information security [25]. Least privilege policies limit the amount of damage that can be caused by compromised credentials, accidental misuse, and intentional misuse by insider threats. Least privilege is also a requirement of all compliance standards such as the Payment Card Industry Data Security Standard, Health Insurance Portability and Accountability Act, and ISO 17799 Code of Practice for Information Security Management [21].

Despite the importance of implementing least privilege policies, they are not always implemented properly because of the difficulty of creating them and sometimes they are not implemented at all. Previous research on the use of least privilege practices in the context of operating systems revealed that the overwhelming majority of study participants did not utilize least privilege policies [17]. This was due to their partial understanding of the security risks, as well as a lack of motivation to create and enforce such policies. Failing to create least privilege policies in a cloud computing environment is **especially high risk** due to the potentially severe security consequences. However, it is also **significantly more difficult** to achieve least privilege in the cloud computing environment than in other environments due to the large variety of services and actions as detailed in Section 3.

Automatic methods for creating security policies that are highly maintainable have received a significant amount of research in works that address the Role Mining Problem (RMP). However, the maintainability of policies does not directly address how secure or complete a policy is. To directly address the goals of security and completeness in policies, we define the **Privilege Error Minimization Problem (PEMP)** where automatically generated policies for future use are evaluated directly on their security and completeness. The most important metric of a generated security policy should be how secure it is (minimizing over-privilege) and how complete it is (minimizing under-privilege).

We use machine learning methods to address the PEMP which is fundamentally a prediction problem. Audit logs contain the richest source of data from which to derive policies that assign privileges to entities. We mine audit logs of cloud services using one unsupervised and one supervised learning algorithm to address the PEMP along with a naive algorithm for comparison. Note that researchers often take a program analysis approach to find which privileges are needed by specific mobile or other types of applications; we do not take this approach to address PEMP because the privilege errors in PEMP are associated with privileged entities, not an application. The F-Measure is a commonly used metric for scoring in binary classification problems which we adapt to our problem. We show how the β variable of the F-Measure can be used to provide a weighted scoring between under-privilege and over-privilege. We present the results of our algorithms across a range of

β values to demonstrate how an organization can determine which approach to use based on its level of acceptable risk.

The main contributions of this paper are: (1) a formal definition of the PEMP which describes the problem of creating complete and secure privilege policies regardless of the access control mechanism, (2) a metric to assess how well the PEMP is solved based on the F-Measure, (3) a methodology of training and validating policy generation algorithms, and (4) one supervised and one unsupervised learning algorithm applied to generating least privilege policies and an analysis of their performance.

Section 2 reviews related works on role mining and automated least privileges. Section 3 presents a comparison of the privilege spaces of various environments and a description of our dataset. Section 4 formally defines the PEMP and a scoring metric for evaluating how well it is solved. Section 5 details specific algorithms and methods used in our approach to addressing the PEMP and Section 6 analyzes the results of these algorithms. Section 7 concludes this work and discusses potential research areas for future work.

2 RELATED WORK

There are two areas of work closely related to ours: role mining and implementing least privilege policies in other environments. Role mining refers to automated approaches to creating Role Based Access Control (RBAC) policies. Role mining can be performed in a top-down manner where organizational information is used or in a bottom-up manner where existing privilege assignments such as access-control lists are used to derive RBAC policies [7]. The problem of discovering an optimal set of roles from existing user permissions is referred to as the Role Mining Problem (RMP) [26].

While we do not directly attempt to solve the RMP or one of its variations, our work has aspects in common with works that do. The authors of [7] defined role mining as being a prediction problem which seeks to create permission assignments that are complete and secure by mining user permission relations. We also employ prediction to mine user permission relations and create policies to balance completeness and security. Our work differs from those that address RMPs in several key ways however. We mine audit log data produced by a system in operation, not existing or manually created user-permission assignments. We do not assume that the given data naturally fits into an RBAC policy that is easy to maintain and secure. Most importantly, instead of evaluating an RBAC configuration based on its maintainability, we focus on evaluating user privilege assignments based on their completeness (minimizing under-privilege) and security (minimizing over-privilege). We view our work as complementary to RMP research as once balanced user permission assignments are generated, existing RMP methods can be used to derive roles which are more compact.

Another area of research closely related to ours is works that use audit log data to achieve least privilege. Privileged entities often already possess the privileges necessary to do their jobs, thus roles can be derived from existing permissions via data mining methods [22]. Methods of automated policy generation have been studied in several environments. Polgen [24] is one of the earliest works in this area which generates policies for programs on SELinux based on patterns in the programs' behavior. Other notable examples of mining audit data to create policies include EASEAndroid [28] for mobile devices, ProgramCutter [29] for desktop applications, and Passe [2] for web applications. [16] used Latent Dirichlet Allocation (LDA), a machine learning technique to create roles from source code version control usage logs. In [4], the same group used a similar approach to evaluate conformance to least privilege and measured the over-privilege of mined roles in operating systems.

Previous approaches have several shortcomings which are addressed in this paper. Polgen guides policy creation based on logs but does not provide over-privilege or under-privilege metrics. EASEAndroid's goal is to identify malicious programs for a single-user mobile environment, not to create user policies. ProgramCutter and Passe help partition system components to improve least privilege but do not create policies for privileged entities. Only [16], [4] and [20] present metrics on over-privilege and under-privilege by comparing policies to usage. Key issues with these works is that they assume roles are stable, not accounting for change in user behavior over time, and use cross-validation for model evaluation which is not appropriate for environments where temporal relationships should be considered. We address these short comings using the rolling forecasting and sliding simulation methods discussed in Sections 4.3.2 and 5.3, respectively. Finally, our work addresses the trade-off between over- and under-privilege and the selection of different algorithms based on how an organization values over- vs. under-privilege. A metric based on the F-Measure for scoring over-privilege and under-privilege by comparing policies to usage and naive algorithm only for building policies was presented in [20] which we expand upon and use the naive algorithm presented in that work for comparison purposes.

3 DATA DESCRIPTION

The cloud environment is multi-user and multi-service, with **high risk** where errors in privilege assignments can cause significant damage to an organization if exploited. With a large number of services, unique privileges to each service, as well as federated identities and identity delegation, the cloud also presents **more complexity** to security policy administrators than environments previously studied for policy creation such as mobile, desktop, or applications. To quantify the scale of privilege complexity, we consider the size of the privilege spaces for three environments: Android 7, IBM z/OS 1.13, and AWS. Android [8] requires an application's permissions to be specified in a manifest included with the application with 128 possible privileges that can be granted. For IBM z/OS [10], we consider the number of services derived from the different types of system resource classes; there are 213 resource classes and five permission states that can be granted to every class. The privilege space of AWS is much larger however, with over 104 services and 2,823 unique privileges as of August 2017 [1].

Our dataset for training and evaluation consists of 5.2M AWS CloudTrail audit events representing one year of cloud audit data provided by a small Software As A Service (SaaS) company. To better understand how much of the privilege space is used in our dataset, statistics about privileged user behavior are shown in Table 1. This table separates the metrics by the first month, last month, and total for one year of data. *Users* is the number of active users during that time period. *Unique Services Avg.* is the average number of unique services used by active users. *Unique Actions Avg.* is the

average number of unique actions exercised by active users, and $\sum$ *Action Avg.* is the average of the total actions exercised by active users. The standard deviation is also provided for Unique Services, Unique Actions, and $\sum$ Actions metrics to understand the variation between individual users. For example, looking at both the Unique and $\sum$ Actions, we observe that their standard deviation is higher than the average for all time periods, indicating a high degree of variation between how many actions users exercise.

Table 1: One Year Total Usage of our Dataset

Metric	First Month	Last Month	One Year
Users	7	13	18
Unique Services Avg.	5.86	8.08	13.50
Unique Services StdDev.	2.97	5.22	9.04
Unique Actions Avg.	13.71	45.31	88.78
Unique Actions StdDev.	20.21	48.13	91.99
$\sum$ Actions Avg.	91.97	78.38	238.30
$\sum$ Actions StdDev.	299.89	261.95	1271.15

4 PROBLEM SCOPE AND APPROACH

The problem we address is that of automatically creating least privilege access control policies in the cloud environment.

4.1 Problem Definition

We refer to the problem formally as the Privilege Error Minimization Problem (PEMP) and define it using the notation from the NIST definition of RBAC [6].

- *USERS, OPS, and OBS (users, operations, and objects, respectively).*
- $PRMS = 2^{OPS \times OBS}$ *, the set of permissions*
- $UPA \subseteq USERS \times PRMS$*, a many-to-many mapping of user-to-permission assignments.*

Additionally we define the following terms:

- $UPE \subseteq UPA$*, a many-to-many mapping of user-permission relations representing permissions exercised by users during a time period.*
- *OBP observation period, the time-period during which exercised permissions (UPE) are observed and used for creating user-to-permission assignment UPA.*
- *OPP operation period, the time-period during which the user-to-permission assignments UPA is to be considered in operation.*

While both UPE and UPA are user-to-permission relations, UPE represents exercised permissions but UPA represents all assignments. Using the preceding terms, we now define the PEMP.

Definition 1. Privilege Error Minimization Problem (PEMP). *Given a set of users USERS, a set of all possible permissions PRMS, and a set of user-permissions exercised UPE, find the set of user-permissions assignments UPA that minimizes the over-privilege and under-privilege errors for a given operation period OPP.*

The PEMP is fundamentally a prediction problem. Given available information over time-period OBP, we seek to predict the set of permission assignments UPA that will be necessary for privileged entities to complete their tasks during a given operation time-period OPP. This UPA should bound the set of permissions exercised during the operation time-period as tightly as possible to avoid both unused permissions (over-privilege) and missing permissions (under-privilege). We have intentionally left the assessment metric of how privilege assignment errors are measured out of the problem definition. A problem may have many solutions as well as many metrics for determining if a problem is solved. This separation of the problem and assessment metrics allows for the discussion of metrics separate from the problem itself.

4.2 Algorithm Overview

Now that we have defined the PEMP as being a prediction problem, we adapt existing prediction algorithms to address it. We utilize two machine learning methods in this paper to generate privilege policies from mining audit log data. First, we employ clustering to find privileged entities which use similar permissions, making the problem analogous to that of finding similar documents in a text corpus. After finding similar users, we generate policies that combine the privileges used by clustered entities. The second machine learning method we employ is classification. Using a set of user-to-privilege relations exercised during the observation period, we train a classifier to learn which user-to-privilege relations should be classified as grant and which should be denied. Once trained, we use the classifier to generate policies for an operation period. More details on the application of these algorithms to generate least privilege policies are discussed in Section 5.

4.3 Model Assessment

We borrow techniques and terminology used in machine learning literature for assessing the effectiveness of our algorithms in addressing the PEMP. Using a standard approach for evaluating the effectiveness of a predictive model [11], we take a test dataset for which we know the expected (target) predictions that the model should make, present it to a trained model, record the actual predictions that made, and compare them to the expected predictions. We first present our method for scoring individual predictions, and then our method for splitting up the dataset into multiple partitions.

4.3.1 Scoring individual predictions. Policy generation for a given operation period is a two-class classification problem where every user-to-permission mapping in a generated policy falls into one of two possible classes: grant or deny. By comparing the predicted privileges to the target privileges, we can categorize each prediction into one of four outcomes:

- True Positive (TP): a privilege that was granted in the predicted policy and exercised during the OPP.
- True Negative (TN): a privilege that was denied in the predicted policy and not exercised during the OPP.
- False Positive (FP): a privilege that was granted in the predicted policy but not exercised during the OPP.
- False Negative (FN): a privilege that was denied in the predicted policy but attempted to be exercised during the OPP.

Using the above outcomes we can then calculate **precision**, **recall**, and the F_1 **measure**, a frequently used set of performance metrics in machine learning and information retrieval [11]. Precision and recall are defined as follows[11]:

$$precision = \frac{TP}{(TP + FP)} \tag{1}$$

$$recall = \frac{TP}{(TP + FN)} \tag{2}$$

In terms of this problem domain, precision is the fraction of permissions accurately granted by the predictor (TP) over all permissions granted by the predictor ($TP + FP$). If there were no permissions granted by the predictor that went unused in the OPP, then $precision = 1$. Thus a high precision value is an indicator of low over-privilege. Similarly, recall is the fraction of permissions accurately granted by the predictor (TP) over all permissions exercised in the OPP ($TP + FN$). If there were no permissions denied by the predictor that should have been granted, then $recall = 1$. Thus a high recall value is an indicator of low under-privilege.

Precision and recall can be collapsed into a single performance metric, the F_1 measure, which is the harmonic mean of precision and recall. For predictive assessment, it is often preferable to use a harmonic mean as opposed to an arithmetic mean. Arithmetic means are susceptible to large outliers which can dominate the performance metrics. The harmonic mean however emphasizes the importance of smaller values and thus gives a more realistic measure of model performance[11]. For example, the arithmetic mean when precision=0 and recall=1 is 0.5, however the harmonic mean of those same values is 0.

The F_1 measure is "balanced" because it gives equal weighting to precision and recall. For our assessment we utilize a general form that allows for a variable weighting between recall and precision (or, under-privilege and over-privilege), β. High β values increase the importance of recall, while low β values increase the importance of precision. The weighted measure, F_β is defined in Equation 3.

$$F_\beta = (1 + \beta^2) \cdot \frac{Precision \cdot Recall}{(\beta^2 \cdot Precision) + Recall} \tag{3}$$

The β weighting is important because it is not reasonable to expect all potential users of a policy generation tool to value over-privilege and under-privilege equally. Molloy et al. identified equal weighting between over- and under-assignments as a problem in several previous works addressing the RMP [15], and preferred to weight more importance to reducing over-privilege. It is also reasonable to expect that some organizations are willing to accept more risk from over-privilege to minimize the cost of privileged entities not being able to perform their duties due to under-privilege.

4.3.2 Scoring multiple predictions. Following the standard approach for evaluating model effectiveness described earlier, we will compare predicted results to expected (target) results. Rather than using a single operation period for our evaluation which may not be representative of the entire dataset, we must partition the dataset into multiple training and test sets using a sampling method. We then aggregate the results of evaluating these partitions to produce a single score for a proposed solution.

For our scenario however, we observe that there is **a temporal aspect to permissions and interdependencies between the exercised actions** which imposes specific restrictions on how we should partition the dataset. For example, a resource such as a virtual machine must be created before it can be used, modified or deleted. Methods such as hold-out sampling and k-fold cross validation which randomly partition a dataset do not account for interdependencies in the data and may not allow for learning algorithms to observe these dependent actions to occur. Thus we use a sampling approach for scenarios like ours which considers a time dimension with interdependent data referred to as "out-of-time sampling"; it is a form of hold-out sampling which uses data from one time period to build a training set and another period to build a test set[11]. The application of out-of-time sampling to generate and score multiple training and test sets is sometimes known as "rolling forecasting origin", which is similar to cross-validation but the training set consists only of observations that occurred prior to those in the test set [9]. Suppose k observations are required to produce a reliable forecast. Then rolling forecasting origin works as follows [9].

(1) Select the observation at time $k + i$ for the test set, and use the observations at times $1, 2, ..., k + i - 1$ to estimate the forecasting model. Compute the error on the forecast for time $k + i$.
(2) Repeat the above step for $i = 1, 2, ..., T - k$ where T is the total number of observations.
(3) Compute the forecast accuracy measures based on the errors obtained.

Adapting the above method to our domain, we allow the training set/observation period to be comprised of any set of dates before time $k + i$, and the test set/operation period is specifically at time $k + i$. We define the step size i to be of one day, which is an adequate amount of time to complete most tasks using related permissions. Also, when using an automated solution to generate permission policies, it is reasonable to expect that new solutions can be generated on at least daily basis.

The measure of forecast accuracy in our scenario is the F_β score for a given operation period described in Section 4.3.1, where a perfect prediction with no over-privilege and no under-privilege present would score a 1.0. We use a rolling mean to compute the accuracy of a proposed solution across all operation periods. Thus our quality measure used for assessing an automated solution to creating permission policies should maximize the average F_β measure across all operation periods:

$$\frac{1}{T-k} \sum_{i=1}^{T-k} F_\beta(Precision_i, Recall_i) \tag{4}$$

5 METHODOLOGY

This section describes the algorithms and techniques we design to address the PEMP in the cloud environment. We first present a naive algorithm which will be used to establish a performance baseline for us to compare the performance of our learning based approaches to. While the naive algorithm merely uses a privilege entity's observed privileges to build policies, the learning based approaches also account for the behavior of other users in generating policies. Each of these methods is applied for a single operation period. The evaluation of an algorithm across multiple operation periods is done using the method described in Section 4.3.2.

5.1 Naive Policy Generation

The naive approach shown in Algorithm 1 takes all privileges exercised during the observation period as input and combines them to form a privilege policy to be used during the operation period. This seems a reasonable approach for a policy administrator to take if

they needed to implement a least privilege policy in an environment where all privilege entities previously had unrestricted access to all permissions. By examining all previous access logs or only the access logs up to a specific point in the past, they can discover all privileges used by each privileged entity and thus expect this to be the set of privileges required for a privileged entity to perform their duties. Although infrequently used privileges will not be captured if they are outside of the observation period, policy generation algorithms can still achieve good results without knowing the frequency for which these privileges are exercised because infrequently used privileges will have little impact on the F_β score, particularly for low β values which value minimizing over-privilege. Furthermore, in a low β environment it is likely that infrequently used privileges should be denied by default and granted by exception instead of always being granted by a long-term policy.

Algorithm 1: Naive Policy Generator

Input: UPE The set of user-permissions exercised during the observation period OBP.
Output: UPA The mapping of user-to-permission assignments.

```
1  UPA ← ∅;
2  for user, perm ∈ UPE do
3      UPA_user ← roles_user ∪ perm;
4  end
5  return UPA
```

5.2 Unsupervised Policy Generation

Our unsupervised learning policy generation method (Algorithm 2) uses a clustering algorithm to find clusters of similar privileged entities based on their permissions exercised. By placing each permission exercised by an entity into a separate document and applying clustering to the document corpus (lines 2-5), we have made the problem analogous to finding similar text documents in a corpus. Once similar entities are grouped by clustering, each group is assigned a shared role and granted the combined permissions of all entities in that role (lines 6-14). Entities which do not belong to any cluster are granted only the privileges they used during the observation period just as in the naive method (lines 15-19). It is important to note that using this method of combining similar entities only grants permissions additional to those used during the observation period. This is **useful in environments where minimizing under-privilege is more important than minimizing over-privilege**.

There are several details of our application of clustering worth describing here. Each document is converted to a feature vector for clustering using a Term Frequency-Inverse Document Frequency (TF-IDF) vectorizer. TF-IDF is a common approach for finding similar documents in information retrieval [14]. The TF-IDF weighting has the advantage that it preserves information about how often each permission is exercised by a user. Once vectorization is complete, the specific clustering algorithm we use for finding similar users is the DBSCAN algorithm of the scikit-learn library [18], an implementation of the algorithm originally published in [5]. The DBSCAN algorithm has several advantages for our scenario, primary among them being that we do not need to specify the expected

Algorithm 2: Unsupervised Policy Generator

Input: UPE The set of user-permissions exercised during the observation period OBP.
Output: UPA The mapping of user-to-permission assignments.

```
1  UPA, documents ← ∅;
2  for user, perm ∈ UPE do
3      documents_user ← documents_user ∪ perm;
4  end
5  clusters, outliers ← DBSCAN(documents);
6  for cluster ∈ clusters do
7      role ← ∅;
8      for user, document ∈ cluster do
9          for perm ∈ document do
10             role ← role ∪ perm;
11         end
12     end
13     UPA_user ← role;
14 end
15 for user, document ∈ outliers do
16     for user, perm ∈ document do
17         UPA_user ← roles_user ∪ perm;
18     end
19 end
20 return UPA
```

number of clusters ahead of time unlike other popular clustering algorithms such as k-means. The performance of DBSCAN also scales well in regards to the number of samples given when compared to other clustering algorithms [23]. There is one relevant hyper-parameter for DBSCAN which we vary in our policy generation experiments, ϵ, which is the maximum distance between two samples for them to be considered as in the same cluster. We **explore three methods for calculating** ϵ: the mean distance between all points, median distance between all points, and middle point between the minimum and maximum points in the vector space.

5.3 Supervised Policy Generation

For the supervised learning approach, we design a classification algorithm to generate policies as follows (and in Algorithm 3). First we construct a training set of documents from the permissions exercised during the observation period and select a subset of previous data for creating the class labels (line 3). We then train a classifier using the training set for each permutation of the Classifier Algorithm Parameters (CAP) (lines 4-6). These multiple instances of the classifier with different permutations of the CAP are used for hyper-parameter selection using the "sliding simulation" method to be described in Section 5.3.2 . Next we create a set of possible permissions that may be exercised during the operation period based on the Policy Generation Parameters (PGP) (line 9). Each of the possible policy permissions is tested against the classifier which will predict that the permission should be either granted or denied, and the results of this classification are used to create the policy for the next operation period (lines 10-15).

Algorithm 3: Supervised Policy Generator

Input: *UPE* User-Permissions Exercised. The set of user-permissions exercised during the observation period *OBP*.
Input: *PRMS* The set of possible permissions.
Input: *TSP* Training Set Parameters. Mapping of parameters used to build the training set.
Input: *CAP* Classifier Algorithm Parameters. Mapping of parameters used to build the predicted policy from a trained classifier.
Input: *PGP* Policy Generation Parameters. Mapping of parameters used to build the predicted policy from a trained classifier.
Output: *UPA* Mapping storing the roles generated by each of the classifier instances.

```
UPA ← ∅;
for tParams ∈ permute(TSP) do
    featureVector, labelSet ←
      createTrainingSet(tParams, UPE);
    for clfParams ∈ permute(CAP) do
        clf ← decisionTree(clfParams);
        clf ← clf.train(featureVector, labelSet);
        for pParams ∈ permute(PGP) do
            roles ← ∅;
            possiblePrivs ←
              createPossiblePrivs(pParams, PRMS);
            for user, perm ∈ possiblePrivs do
                if clf.predict(user, perm) == 'granted' then
                    roles_user ← roles_user ∪ perm;
                end
            end
            UPA_tParams,clfParams,pParams ← roles;
        end
    end
end
return UPA
```

5.3.1 Classification Algorithm and Feature Selection. We use a **decision tree (DT)** classification algorithm for supervised learning, also from the scikit-learn library [18]. The algorithm implemented in the library is an optimized version, an implementation of the CART algorithm published in [3]. The advantages of the decision tree algorithm used are speed and the ability to display the set of rules learned during classification. It was also the top performing classification algorithm in our preliminary comparison of 15 different classification algorithms in the scikit-learn library.

We utilize five features available directly from the audit log data for training: the time at which a permission was exercised, the unique identifier of the executing entity, the type of entity (user or delegated role), the service which the action belonged to, and the type of action performed. Instead of using the absolute time of an action, we derive features capturing whether it was exercised on a weekend or weekday, as well as the specific day of the week. These are all bottom-up data attributes available directly from the access logs. Other top-down information such as job role or organization department was not available with our dataset (nor does it exist in many small organizations), but could easily be integrated with the exercised privilege information if available.

5.3.2 Sliding Simulation for Supervised Parameter Selection. Several hyper-parameters must be selected for our supervised learning approach. These include parameters for the decision tree classifier, the constructions of the training set, the policy construction from the trained classifier. Our method for selecting optimized hyper-parameters uses only out-of-sample data and is an adaptation of the "sliding simulation" method presented in [12].

The sliding simulation method of [12] is based on three premises. First, a model should be selected based on how well it predicts out-of-sample actual data, not on how well it fits historical data. Second, a model is selected from among many candidates run in parallel on the out-of-sample data. Third, models are optimized for each forecast horizon separately, making it possible to use different models and optimize parameters within models. The method operates by running several prediction models in parallel across a sliding window of data, computing the accuracy of each model for a given period and selecting the model(s) with the best score to be used in creating the forecast for the next period. Using this technique, the author in [12] showed that it outperformed the best method of a previous competition in statistical forecasting (the *M*-Competition [13]) by a large margin.

As in the sliding simulation method, we run many permutations of parameters in parallel on out-of-sample data and use the best performing parameters to create a future prediction. Modifications were implemented to adapt sliding simulation to our problem domain. Sliding simulation originally dealt with making numerical predictions and measuring the error between a predicted and actual value. In our scenario a security policy is the prediction and we use the F_β score presented in Section 4.3.1 as our scoring criteria. While [12] used all observation points before the forecast period, the most recent exercised permissions are most relevant to predicting future permissions; training a classifier with older and less relevant permissions had a negative effect on prediction accuracy.

5.4 Model Decomposition

Time series decomposition is a common technique used to improve predictions [9], it identifies patterns in data and decomposes the data into different models based on those patterns. **We applied time series decomposition to our data after recognizing significant differences between the privileges exercised during weekdays and weekends**. While given enough data and the proper features a supervised approach should be able to learn and use these patterns to make predictions, decomposing the data provides several advantages: (1) improves scores for both naive and unsupervised algorithms, (2) less training data is needed for the supervised approach since it does not need to learn the different behavior patterns in weekdays and weekends, and (3) information about weekday or weekend patterns can be used in hyper-parameters that control the creation of the training set for supervised learning.

We use two methods of decomposing the time series data which we term *filter decomposition* and *filler decomposition*. For the filter method, the days which do not fit into the chosen model are filtered out of each observation period in the sliding window evaluation

before the data are used by the algorithms. With the filler method, the end date of the sliding window evaluation is used as a starting point and the observation period is created by enlarging the window by moving the start date backward until the observation period is "filled" with only data matching the chosen model. Consider a sliding window evaluation with a window size of 10 days using these two decomposition methods. For the filter method, the number of days fitting the weekday model will vary from 6 to 8, and the number of days fitting the weekend model will vary from 2 to 4. For the filler method, the number of days fitting a model will always be 10 days when the sliding window size is 10 days.

The decomposition method used for evaluation is chosen based on the β value we wish to optimize for. For algorithms seeking to score well for $\beta > 1$, increasing the window size results in better scores, and the filter approach is used where the variations in the observation dataset size are smoothed out across larger windows. For experiments which seek to score well for $\beta < 1$, smaller window sizes score more favorably but the variable number of matching days which fit within a chosen time period can have undesirable effects on the results when using small window sizes. Thus the filler model is used in experiments for $\beta < 1$ which gives a consistent number of days for data points in each window.

6 RESULTS

This section analyzes the performance of our algorithms for generating security policies. We first examine the results using the complete model and then show how decomposition and the use of multiple decomposed models can improve on those results.

6.1 Complete Model Results

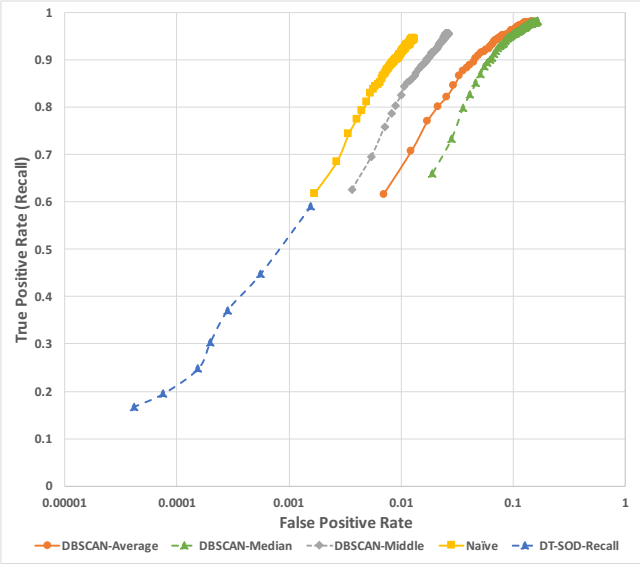

Figure 1: Receiver Operating Characteristic Curves

The Receiver Operating Characteristic (ROC) curve is a graphic commonly used to chart the performance of binary classifiers. It charts the trade-off between the True Positive Rate (TPR, also called recall) and the False Positive Rate (FPR) of a binary classifier, with the ideal performance having a TPR value of one and FPR of zero. While the ROC illustrates FPR, the rest of the charts in this section use F_β described in Section 4.3.1. The ROC curves for the naive, three unsupervised (DBSCAN) and one supervised (DT) algorithms across multiple observation period lengths are presented in Figure 1. All of the algorithms perform well in terms of minimizing the FPR with the unsupervised methods being able to provide higher recall than the naive approach but at the cost of higher FPR. The supervised approach is not able to score as well as the other algorithms in terms of recall but maintains a lower FPR for all data points. The use of specific observation period sizes for the sliding window method described in Section 4.3 prevents the data points from spanning the entire range of the chart which is typical for ROC curves.

The performance of the naive, three unsupervised, and two supervised algorithms across F_β values for $1/100 <= \beta <= 100$ is presented in Figure 2. Two separate methods are in this section for labeling the training data: substitution/overlapping daytype (SOD) where a day of the same type (weekday or weekend) is used which overlaps with the observation period, and substitution/non-overlapping day of week (SND) where a day on the same day of the week is used which was prior to (non-overlapping) the observation period. Additionally, the performance of the policy that allows all privileges are also shown in this chart for comparison. The scores on this chart represent the best performance of each algorithm regardless of the size of the sliding window used for the observation period. Some important trends are evident from this chart. For β values where $1 < \beta < 50$, the naive approach performs the best with the unsupervised methods scoring slightly better after $\beta > 50$. The policy that allows all privileges comes close to scoring as good as the naive approach at $\beta = 100$, but even for such a high β, the naive and unsupervised algorithms are still favorable over the allow all policy. While the performance of the unsupervised algorithms is not very compelling in this chart, later results using decomposition will show a larger performance gap between the naive and unsupervised methods for high β values. The supervised algorithms score relatively poorly for $\beta > 1$. For β values where $\beta < 1$, the supervised algorithms score significantly better than the naive algorithm as β decreases with the performance gap widening until $\beta < 1/30$, where the scores of the supervised and naive algorithms cease to improve as β decreases. The unsupervised algorithms score relatively poorly for $\beta < 1$.

The trends in these charts highlight the **strengths and weaknesses of each algorithm**. By granting users the privileges used by similar users, the unsupervised algorithms predict privileges a user may use in the future. But there is no mechanism for the unsupervised learning algorithm to learn which possible privilege grants may result in over-privilege and restrict these privileges accordingly. The supervised algorithms attempt to learn any patterns in the past data and use these to predict future privilege assignments. While privileges used previously are likely to be used again and rarely used privileges can be denied with some degree of confidence, it is difficult to predict the usage of a future privilege that has never been used before using only past patterns.

Figures 1 and 2 show the scores of algorithms regardless of the size of the observation period. We next examine the performance of these algorithms for fixed β values as the observation period size varies. We chose values $\beta = 80$ and $\beta = 1/10$ because these

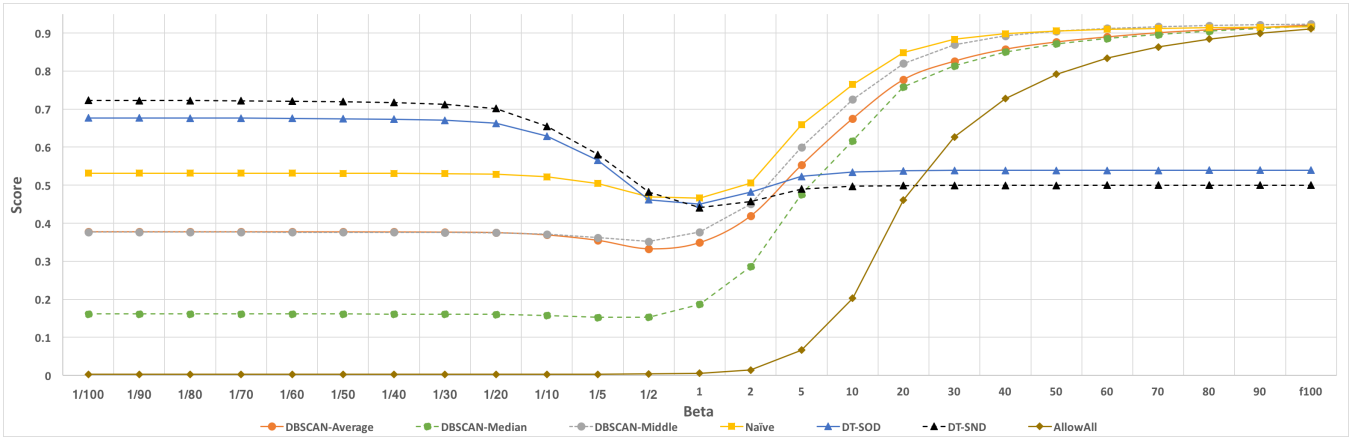

Figure 2: Beta Values Curves

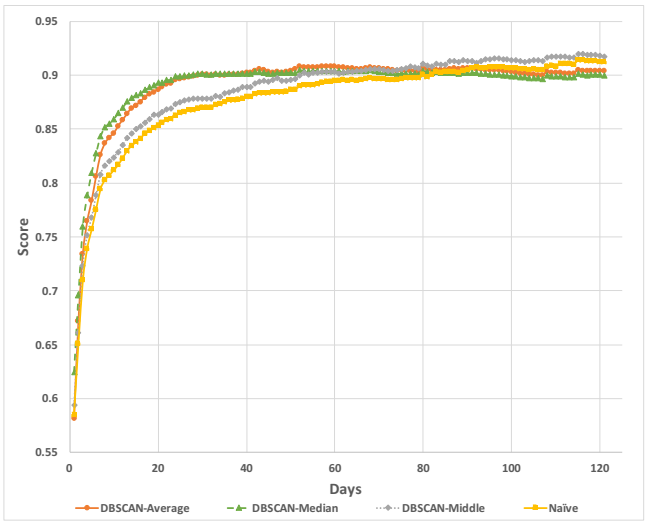

Figure 3: $\beta = 80$

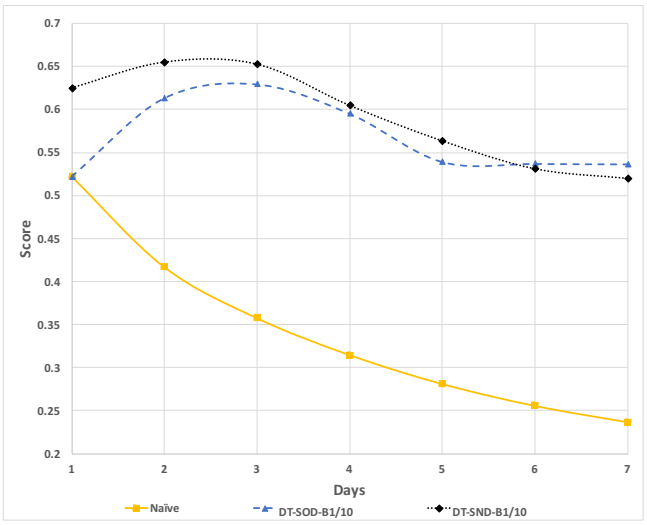

Figure 4: $\beta = 1/10$

seemed the most interesting in terms of the trade-offs between the various methods. The performance of the unsupervised and naive algorithms for $\beta = 80$ are shown in Figure 3. The choice of ϵ as the threshold for determining which users are alike presents interesting trade-offs between window size and score. In general, using the median for calculating ϵ consistently provides slightly better scores than the naive approach across all window sizes with the scores for both the unsupervised algorithm (with the middle method) and naive algorithm peaking at 115 days. Using the average and middle methods for calculating ϵ both provide better scores for observation periods < 40 days, but their scores level off there and begin to gradually decrease after peaking at 59 days for the average method and 68 days for the median method.

The performance of the supervised and naive algorithms for $\beta = 1/10$ are shown in Figure 4. The naive algorithm achieves its best performance with an observation period of one day and steadily declines after that. The supervised algorithms all achieve their best performance with an observation period size of 2 or 3 days and then decline until leveling off around six and seven days. Among the supervised methods, the SND approach performs the best for observation periods less than five days but declines more rapidly than the SOD labeling method. Although not charted here, the precision score of the supervised methods constantly increases and the recall score constantly decreases as the observation period increases. The increase in precision is not rapid enough to overcome the decrease in recall after the observation period exceeds 3 days however, which is why the scores for the supervised algorithms decrease or level off after that point. Conversely, the precision score of the naive method constantly decreases and the recall score constantly increases as the observation period increases.

6.2 Decomposed Models Results

In this section we present the results after decomposing the dataset in separate models for weekday and weekend data using the decomposition methods discussed previously in Section 5.4.

The performance of the complete and decomposed models for β values $>= 1$ for both the naive algorithm and the unsupervised algorithm (with the average method for calculating ϵ) are shown in Figure 5. For both algorithms, the weekday model performance is superior to the complete model for β values $>= 1$. The trend

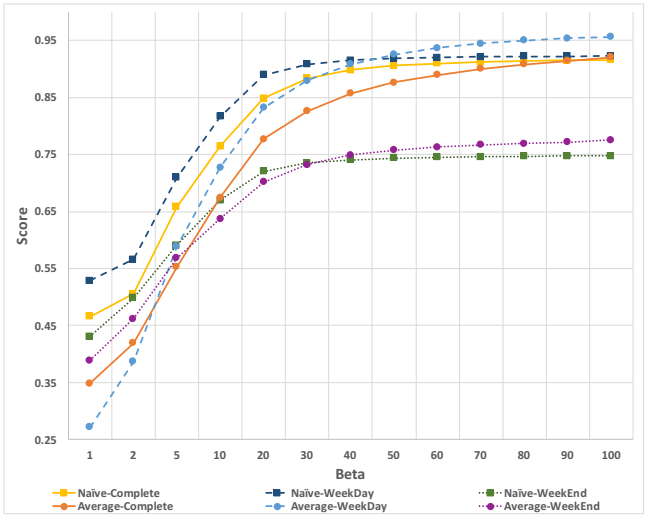

Figure 5: Decomposed Models Unsupervised $\beta >= 1$

previously illustrated in Figure 2 of the unsupervised algorithm under-performing the naive algorithm for low β but eventually outperforming it as β increases is also present in this chart but more pronounced. The performance gap between unsupervised and naive algorithms widens in the decomposed models with the unsupervised algorithm overtaking the naive algorithm at $\beta = 50$ for the weekday model and $\beta = 40$ for the weekend model, where previously the unsupervised algorithm did not outscore the naive algorithm in the complete model until $\beta = 90$. The weekend model performance is generally worse than the complete model performance. There are two primary reasons for this: first, there is less data available to the weekend model, with only 28% of the complete model data; second, the activity of users on the weekends is lower and highly inconsistent, making it harder to find similar entities and less likely that similar users will exercise similar privileges in a cluster if identified.

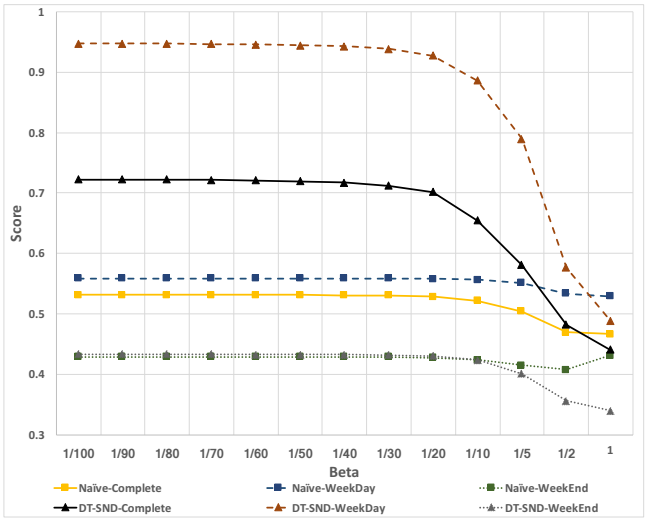

Figure 6: Decomposed Models, Supervised $\beta <= 1$

The performance of the complete and decomposed models for β values <= 1 for both the naive algorithm and the supervised algorithm (using the SND labeling method) are shown in Figure 6. As with the unsupervised algorithm and β values >= 1, the weekday model outperforms the complete model while the weekend model under-performs the complete model where β values <= 1 as well. The performance gap between the weekday and complete models for the supervised algorithm is much larger than in previously examined experiments. With the inconsistent activity of the weekend actions removed, the supervised algorithm is better able to identify and leverage patterns to create security policies. The performance of the supervised algorithm for the weekend model decreased substantially compared to the complete model however. For $\beta = 1/30$, the supervised weekend model scored 39% lower than the complete model, while the naive weekend model scored only 19% lower than its complete model. The reasons for the lower weekend model scores for the supervised algorithm are the same as the lower weekend model scores for the unsupervised algorithm: there is less data to work with and higher variability in that data.

6.3 Recomposed Models Results

Section 6.2 illustrated how decomposition improved scoring for the weekday model, but we are interested in finding the highest possible score across all days in the available dataset. **To improve the overall score, we combine two previously examined models** using one model and algorithm for the weekday policies and another model and algorithm for the weekend policies which we refer to this as a recomposed model. To build the recomposed model, we use policies from the weekday model when evaluating weekdays, but as the previously examined results have shown, the weekend models performed fairly poor so we will instead use policies generated by the complete model when evaluating weekends.

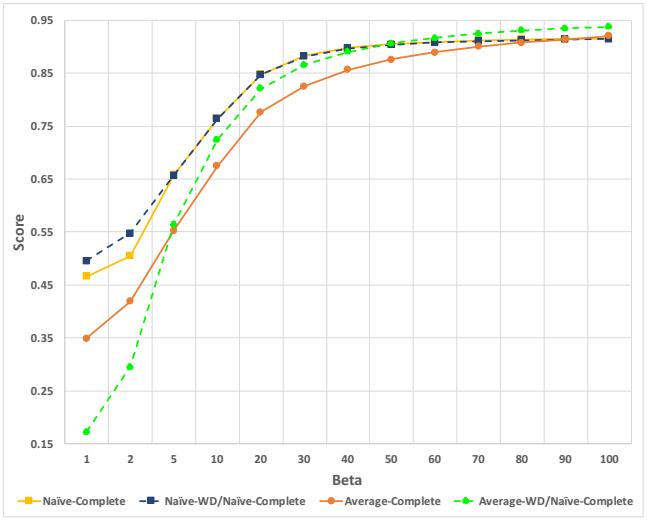

Figure 7: Recomposed Models, $\beta >= 1$

The performance of the complete and recomposed models for β values >= 1 for both the naive algorithm and the unsupervised algorithm (with the average method used for calculating ϵ) are shown in Figure 7. For the unsupervised algorithm, the recomposed model

outscores the complete model for β values >= 5, and outscores the naive algorithm for both the complete and recomposed models for β >= 50, with the performance gap increasing after that as β increases. For the naive algorithm however, the improved scores of the weekday model are not enough to offset the poorer scores of the complete model for the weekend days, thus the recomposed model using the naive algorithm scores almost the same as the complete model for $\beta > 5$. The scores for the highest β value tested are .9379 for the recomposed model with the unsupervised algorithm and .9149 for the recomposed model with the naive algorithm, an improvement of 2.5% over an already fairly high score.

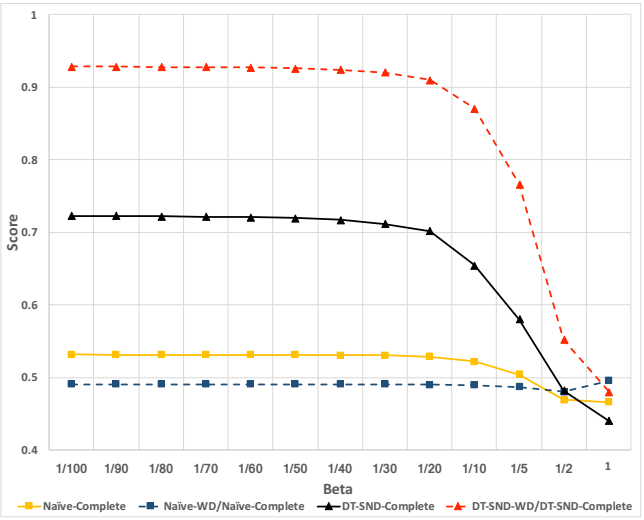

Figure 8: Recomposed Models, β <= 1

The performance of the complete and recomposed models for β values <= 1 for both the naive algorithm and the supervised algorithm (using the SNDT labeling method) are shown in Figure 8. For the recomposed model using the supervised algorithm, the significantly improved scores of the weekdays using the weekday model are combined with the weekends from the complete model to improve the overall scores by 89% compared to the naive complete model at $\beta = 1/100$. For the recomposed model using the naive algorithm, the improvement provided by the weekday model was not enough to offset the poor scores of the weekend policies in the complete model, resulting in the recomposed model scoring lower than the complete model for $\beta < 1/2$.

6.4 Results Summary

Creating security policies is inherently an optimization problem that must balance between minimizing over-privilege and minimizing under-privilege. How much one values achieving one of these objectives vs. the other can be expressed using the β value as described in Section 4.3. The results of this section demonstrate the **effectiveness algorithms and decomposition methods that can be used to create better security policies** for a cloud environment with "better" being expressed in terms of the F_β score.

We also presented the results of using decomposition methods to decompose the dataset into weekday and weekend models and then use the best aspects of the weekday and complete models for scoring across the complete dataset time period. Not all audit log datasets will exhibit similar behavior that benefits from such decomposition, but it is reasonable to expect many datasets consisting of audit log events generated by human privileged entities working a five-day work week will. Regardless of the decomposition method used, we find that the **unsupervised algorithm performs more favorably as β increases** due to its ability to use information from similar users to predict the future use of privileges. The unsupervised algorithm does not have a mechanism to deny privileges however, so its scores are relatively low for small β values. Conversely, **the supervised algorithm performs more favorably as β decreases** but poorly for large β values. The supervised algorithm is able to use the recurring patterns in data to score well for restricting privileges, but scores poorly at predicting possible new privileges that privileged entities may use. The naive approach performs well only for values near $\beta = 1$, representing its favorability for environments which value balancing over- and under-privilege nearly equally but it is outperformed by the other algorithms as the β value increases or decreases away from $\beta = 1$. **The key takeaway from these results is that how an organization values over-privilege vs. under-privilege will determine which algorithm is best suited for generating that organization's security policies; none of the three examined algorithms is clearly superior to the others for all likely scenarios.**

7 CONCLUSION

This paper addressed issues related to automatically creating least privilege policies in the cloud environment. We defined the Privilege Error Minimization Problem (PEMP) to directly address the goals of completeness and security when creating privilege policies, and introduced a weighted scoring mechanism to evaluate a policy against these goals. We adapted techniques from statistical forecasting and machine learning to train and evaluate a supervised and an unsupervised learning algorithm for automated policy generation. The results of our analysis show that the supervised algorithm performed well for reducing over-privilege while the unsupervised algorithm performed well for reducing under-privilege compared to a naive approach. These results demonstrate the potential to apply such automated methods to create more secure roles based on an organization's acceptable level of risk in accepting over-privilege vs. its desire to minimize the effort to correct under-privilege.

This paper suggests many possibilities for future research in automated least privileges approaches. The policy generation approaches described in this paper are based on features directly available in the audit logs such as the service name, user name, and privilege exercised. We would consider additional features for future research such as properties of the requesting entity and the resources being operated on such as a user's job title and organizational unit or the subnet(s) which a virtual resource operates within. Combining the ability of the unsupervised algorithm (to predict the use of future privileges based on clusters of similar users) with the ability of the supervised algorithm (to restrict privileges which are unlikely to be used in the future) may also improve scoring.

ACKNOWLEDGMENT

This research was supported in part by the NSF grant DGE-1619841.

REFERENCES

[1] Amazon Web Services. 2017. IAM Policy Generator Source Code. https://awsiamconsole.s3.amazonaws.com/iam/assets/js/bundles/policies.js. (2017). Accessed: 2017-05-04.

[2] Aaron Blankstein and Michael J Freedman. 2014. Automating isolation and least privilege in web services. In *Security and Privacy (SP), 2014 IEEE Symposium on*. IEEE, 133–148.

[3] Leo Breiman, Jerome Friedman, Charles J Stone, and Richard A Olshen. 1984. *Classification and regression trees*. CRC press.

[4] Suresh Chari, Ian Molloy, Youngja Park, and Wilferid Teiken. 2013. Ensuring continuous compliance through reconciling policy with usage. In *ACM Symposium on Access control models and technologies (SACMAT)*. ACM, 49–60.

[5] Martin Ester, Hans-Peter Kriegel, Jörg Sander, Xiaowei Xu, et al. 1996. A density-based algorithm for discovering clusters in large spatial databases with noise.. In *Knowledge discovery in databases (KDD)*, Vol. 96. AAAI Press, 226–231.

[6] David F Ferraiolo, Ravi Sandhu, Serban Gavrila, D Richard Kuhn, and Ramaswamy Chandramouli. 2001. Proposed NIST standard for role-based access control. *ACM Transactions on Information and System Security (TISSEC)* 4, 3 (2001), 224–274.

[7] Mario Frank, Joachim M Buhmann, and David Basin. 2010. On the definition of role mining. In *ACM Symposium on Access control models and technologies (SACMAT)*. ACM, 35–44.

[8] Google, Inc. 2017. Manifest.permission — Android Developers. https://developer.android.com/reference/android/Manifest.permission.html. (2017). Accessed: 2017-01-10.

[9] Rob J Hyndman and George Athanasopoulos. 2014. *Forecasting: principles and practice*. OTexts.

[10] IBM Corporation. 2012. z/OS Security Server RACF General User's Guide. https://www.ibm.com/support/knowledgecenter/en/SSLTBW_1.13.0/com.ibm.zos.r13.icha100/toc.htm. (2012). Accessed: 2017-05-17.

[11] John D Kelleher, Brian Mac Namee, and Aoife D'Arcy. 2015. Fundamentals of Machine Learning for Predictive Data Analytics. (2015).

[12] Spyros Makridakis. 1990. Sliding Simulation: A New Approach to Time Series Forecasting. *Management Science* 36, 4 (1990), 505–512.

[13] Spyros Makridakis, A Andersen, Robert Carbone, Robert Fildes, Michele Hibon, Rudolf Lewandowski, Joseph Newton, Emanuel Parzen, and Robert Winkler. 1982. The accuracy of extrapolation (time series) methods: Results of a forecasting competition. *Journal of forecasting* 1, 2 (1982), 111–153.

[14] Christopher D. Manning, Prabhakar Raghavan, and Hinrich Schütze. 2008. *Introduction to Information Retrieval*. Cambridge University Press, New York, NY, USA. 117–119 pages.

[15] Ian Molloy, Ninghui Li, Tiancheng Li, Ziqing Mao, Qihua Wang, and Jorge Lobo. 2009. Evaluating role mining algorithms. In *ACM Symposium on Access control models and technologies (SACMAT)*. ACM, 95–104.

[16] Ian Molloy, Youngja Park, and Suresh Chari. 2012. Generative Models for Access Control Policies: Applications to Role Mining over Logs with Attribution. In *ACM Symposium on Access control models and technologies (SACMAT)*. ACM, 45–56.

[17] Sara Motiee, Kirstie Hawkey, and Konstantin Beznosov. 2010. Do windows users follow the principle of least privilege?: investigating user account control practices.. In *Symposium on Usable Privacy and Security (SOUPS)*. ACM.

[18] Pedregosa, F., et al. 2011. Scikit-learn: Machine Learning in Python. *Journal of Machine Learning Research* 12 (2011), 2825–2830.

[19] Jerome H Saltzer and Michael D Schroeder. 1975. The protection of information in computer systems. *Proc. IEEE* 63, 9 (1975), 1278–1308.

[20] Matthew Sanders and Chuan Yue. 2017. Automated Least Privileges in Cloud-Based Web Services. In *Hot Topics in Web Systems and Technologies (HotWeb)*. IEEE.

[21] SANS Institute. 2010. A Compliance Primer for IT Professionals. https://www.sans.org/reading-room/whitepapers/compliance/compliance-primer-professionals-33538. (2010). Accessed: 2017-09-24.

[22] Jrgen Schlegelmilch and Ulrike Steffens. 2005. Role mining with ORCA.. In *ACM Symposium on Access control models and technologies (SACMAT)*. ACM, 168–176.

[23] scikit-learn developers. 2016. Overview of clustering methods. http://scikit-learn.org/stable/modules/clustering.html. (2016). Accessed: 2017-09-01.

[24] Brian T. Sniffen, David R. Harris, and John D. Ramsdell. 2006. Guided policy generation for application authors.. In *SELinux Symposium*.

[25] Harold F Tipton and Kevin Henry. 2006. *Official (ISC) 2 guide to the CISSP CBK*. Auerbach Publications.

[26] Jaideep Vaidya, Atluri Vijayalakshmi, and Qi Guo. 2007. The role mining problem: finding a minimal descriptive set of roles.. In *ACM Symposium on Access control models and technologies (SACMAT)*. ACM, 175–184.

[27] Bob Violino. 2017. Cloud Computing Sees Huge Growth Rates Across All Segments. http://www.information-management.com/news/infrastructure/cloud-computing-sees-huge-growth-rates-across-all-segments-10030682-1.html. (2017). Accessed: 2017-09-07.

[28] Ruowen Wang, William Enck, Douglas Reeves, Xinwen Zhang, Peng Ning, Dingbang Xu, Wu Zhou, and Ahmed M. Azab. 2015. EASEAndroid: Automatic Policy Analysis and refinement for security enhanced android via large-scale semi-supervised learning. In *USENIX Security Symposium*. USENIX, 351–366.

[29] Yongzheng Wu, Jun Sun, Yang Liu, and Jin Song Dong. 2013. Automatically partition software into least privilege components using dynamic data dependency analysis.. In *IEEE/ACM International Conference on Automated Software Engineering (ASE)*. IEEE Press, 323–333.