AUTOMATED METHODS FOR GENERATING LEAST PRIVILEGE ACCESS CONTROL POLICIES

by

Matthew W. Sanders
A thesis submitted to the Faculty and the Board of Trustees of the Colorado School of Mines in partial fulfillment of the requirements for the degree of Doctor of Philosophy (Computer Science).

Golden, Colorado
Date __________________________

Signed: ________________________
Matthew W. Sanders

Signed: ________________________
Dr. Chuan Yue
Thesis Advisor

Golden, Colorado
Date __________________________

Signed: ________________________
Dr. Tracy Camp
Professor and Head
Department of Computer Science
ABSTRACT

Access controls are the processes and mechanisms that allow only authorized users to perform operations upon the resources of a system. Using access controls, administrators attempt to implement the Principle of Least Privilege, a design principle where privileged entities operate using the minimal set of privileges necessary to complete their job. This protects the system against threats and vulnerabilities by reducing exposure to unauthorized activities. Although access control can be considered only one area of security research, it is a pervasive and omnipresent aspect of information security.

But achieving the Principle of Least Privilege is a difficult task. It requires the administrators of the access control policies to have an understanding of the overall system, each user’s job function, the operations and resources necessary to those job functions, and how to express these using the access control model and language of the system. In almost all production systems today, this process of defining access control policies is performed manually. It is error prone and done without quantitative metrics to help administrators and auditors determine if the Principle of Least Privilege has been achieved for the system.

In this dissertation, we explore the use of automated methods to create least privilege access control policies. Specifically, we (1) develop a framework for policy generation algorithms, derive metrics for determining adherence to the Principle of Least Privilege, and apply these to evaluate a real world dataset, (2) develop two machine learning based algorithms for generating role based policies and compare their performance to naive methods, and (3) develop a rule mining based algorithm to create attribute based policies and evaluate its effectiveness to role based methods. By quantifying the performance of access control policies, developing methods to create least privilege policies, and evaluating their performance using real world data, the projects presented in this dissertation advance the state of access control research and address a problem of great significance to security professionals.
# TABLE OF CONTENTS

<table>
<thead>
<tr>
<th>Section</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>ABSTRACT</td>
<td>iii</td>
</tr>
<tr>
<td>LIST OF FIGURES</td>
<td>viii</td>
</tr>
<tr>
<td>LIST OF TABLES</td>
<td>x</td>
</tr>
<tr>
<td>LIST OF ABBREVIATIONS</td>
<td>xi</td>
</tr>
<tr>
<td>ACKNOWLEDGMENTS</td>
<td>xiii</td>
</tr>
<tr>
<td>CHAPTER 1 INTRODUCTION</td>
<td>1</td>
</tr>
<tr>
<td>1.1 Automated Least Privileges in Cloud-Based Web Services</td>
<td>6</td>
</tr>
<tr>
<td>1.2 Minimizing Privilege Assignment Errors in Cloud Services</td>
<td>7</td>
</tr>
<tr>
<td>1.3 Mining Least Privilege Attribute Based Access Control Policies</td>
<td>7</td>
</tr>
<tr>
<td>CHAPTER 2 AUTOMATED LEAST PRIVILEGES IN CLOUD-BASED WEB SERVICES</td>
<td>9</td>
</tr>
<tr>
<td>2.1 Introduction</td>
<td>9</td>
</tr>
<tr>
<td>2.2 Related Work</td>
<td>10</td>
</tr>
<tr>
<td>2.3 Over-Privilege in Manually Generated Policies</td>
<td>11</td>
</tr>
<tr>
<td>2.4 Policy Generation and Evaluation</td>
<td>13</td>
</tr>
<tr>
<td>2.5 Metrics</td>
<td>16</td>
</tr>
<tr>
<td>2.6 Results</td>
<td>18</td>
</tr>
<tr>
<td>2.6.1 Impact of Varying the Operation Period</td>
<td>18</td>
</tr>
<tr>
<td>2.6.2 Impact of Varying the Observation Period</td>
<td>21</td>
</tr>
<tr>
<td>2.6.3 Summary of Results</td>
<td>24</td>
</tr>
</tbody>
</table>
CHAPTER 3 MINIMIZING PRIVILEGE ASSIGNMENT ERRORS IN CLOUD SERVICES

3.1 Introduction

3.2 Related Work

3.3 Data Description

3.4 Problem Scope and Approach

3.4.1 Problem Definition

3.4.2 Algorithm Overview

3.4.3 Model Assessment

3.4.3.1 Scoring individual predictions

3.4.3.2 Scoring multiple predictions

3.5 Methodology

3.5.1 Naive Policy Generation

3.5.2 Unsupervised Policy Generation

3.5.3 Supervised Policy Generation

3.5.3.1 Classification Algorithm and Feature Selection

3.5.3.2 Sliding Simulation for Supervised Parameter Selection

3.5.4 Model Decomposition

3.6 Results

3.6.1 Complete Model Results

3.6.2 Decomposed Models Results

3.6.3 Recomposed Models Results
CHAPTER 4 MINING LEAST PRIVILEGE ATTRIBUTE BASED ACCESS
CONTROL POLICIES .............................................. 58

4.1 Introduction .............................................. 58

4.2 Background .............................................. 60

4.2.1 Attribute Based Access Control (ABAC) ............... 60

4.2.1.1 ABAC Definition .................................... 60

4.2.1.2 ABAC Benefits .................................... 60

4.2.1.3 ABAC vs. RBAC .................................... 61

4.2.2 Rule Mining Methods .................................. 62

4.3 Related Work ............................................ 63

4.4 Least Privilege Policy Generation ......................... 63

4.5 ABAC Policy Mining ..................................... 64

4.5.1 ABAC Rule Mining Based Works ..................... 64

4.5.2 ABAC Policy Minimization Works ..................... 65

4.6 Problem Definition and Metrics ......................... 68

4.6.1 Problem Definition .................................... 68

4.6.2 Evaluation Metrics .................................... 69

4.6.2.1 Scoring Individual Predictions .................... 69

4.6.2.2 Scoring Policies Across Multiple Time Periods .... 71

4.6.2.3 Scoring Infinite Possible Resource Identifiers .... 72

4.7 Methodology ............................................. 72
**LIST OF FIGURES**

<table>
<thead>
<tr>
<th>Figure</th>
<th>Description</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>2.1</td>
<td>Number of Granted &amp; Used Actions by Role</td>
<td>13</td>
</tr>
<tr>
<td>2.2</td>
<td>Number of Granted &amp; Used Services by Role</td>
<td>14</td>
</tr>
<tr>
<td>2.3</td>
<td>Sliding Window Evaluation</td>
<td>15</td>
</tr>
<tr>
<td>2.4</td>
<td>User Evaluation as Opr Days Vary (Obs Days=7,28)</td>
<td>19</td>
</tr>
<tr>
<td>2.5</td>
<td>Role Evaluation as Opr Days Vary (Obs Days=7,28)</td>
<td>20</td>
</tr>
<tr>
<td>2.6</td>
<td>$TF_\beta$ score as Opr Days Vary (Obs Days=7)</td>
<td>21</td>
</tr>
<tr>
<td>2.7</td>
<td>User Evaluation as Obs Days Vary (Opr Days=1,7)</td>
<td>22</td>
</tr>
<tr>
<td>2.8</td>
<td>Role Evaluation as Obs Days Vary (Opr Days=1,7)</td>
<td>23</td>
</tr>
<tr>
<td>2.9</td>
<td>User $TF_\beta$ scores as Obs Days Vary (Opr Days=1)</td>
<td>24</td>
</tr>
<tr>
<td>2.10</td>
<td>Role $TF_\beta$ scores as Obs Days Vary (Opr Days=1)</td>
<td>25</td>
</tr>
<tr>
<td>3.1</td>
<td>Receiver Operating Characteristic Curves</td>
<td>46</td>
</tr>
<tr>
<td>3.2</td>
<td>Beta Values Curves</td>
<td>47</td>
</tr>
<tr>
<td>3.3</td>
<td>Naive vs. Unsupervised Algorithms, $\beta = 80$</td>
<td>49</td>
</tr>
<tr>
<td>3.4</td>
<td>Naive vs. Supervised Algorithms, $\beta = 1/10$</td>
<td>50</td>
</tr>
<tr>
<td>3.5</td>
<td>Decomposed Models Unsupervised $\beta &gt;= 1$</td>
<td>51</td>
</tr>
<tr>
<td>3.6</td>
<td>Decomposed Models, Supervised $\beta &lt;= 1$</td>
<td>52</td>
</tr>
<tr>
<td>3.7</td>
<td>Recomposed Models, $\beta &gt;= 1$</td>
<td>54</td>
</tr>
<tr>
<td>3.8</td>
<td>Recomposed Models, $\beta &lt;= 1$</td>
<td>55</td>
</tr>
<tr>
<td>4.1</td>
<td>Top 50 Attributes Ranked by Frequency</td>
<td>80</td>
</tr>
<tr>
<td>4.2</td>
<td>Comparison of Candidate Evaluation Metrics</td>
<td>87</td>
</tr>
</tbody>
</table>
Figure 4.3  Comparison of Methods for Calculating Coverage Rates . . . . . . . . . . 90
Figure 4.4  Performance as Itemset Frequency Varies . . . . . . . . . . . . . . . . . 91
Figure 4.5  Performance as Observation Period Varies . . . . . . . . . . . . . . . . . 92
Figure 4.6  Comparison of ABAC vs. RBAC Performance . . . . . . . . . . . . . . . . 93
LIST OF TABLES

Table 3.1 One Year Total Usage of our Dataset ......................... 32
Table 4.1 16 Month Total Usage of our Dataset .......................... 85
LIST OF ABBREVIATIONS

Attribute Based Access Control .................................................. ABAC
Area Under Curve ........................................................................ AUC
Amazon Web Services ................................................................. AWS
Federal Identity, Credential, and Access Management Architecture . . . . . . . . FICAM
False Negative ........................................................................... FN
False Positive .............................................................................. FP
False Positive Rate ...................................................................... FPR
Health Insurance Portability and Accountability Act .......................... HIPAA
Identity and Access Management ...................................................... IAM
International Organization for Standardization ............................... ISO
Latent Dirichlet Allocation .............................................................. LDA
National Institute of Standards and Technology .............................. NIST
Observation Period ...................................................................... OBP
Operation Period .......................................................................... OPP
Over-Privilege Rate ....................................................................... OPR
Payment Card Industry Data Security Standard ............................... PCI-DSS
Privilege Error Minimization Problem ............................................. PEMP
Principle of Least Privilege ............................................................ PoLP
Role Based Access Control ............................................................. RBAC
Receiver Operating Characteristic ................................................ ROC
Role Mining Problem .................................................. RMP
Software As A Service ........................................... SaaS
Term Frequency-Inverse Document Frequencyj ............ TF-IDF
Temporal Over-Privilege Rate .................................... TOPR
True Negative .......................................................... TN
True Positive .......................................................... TP
True Positive Rate .................................................... TPR
Under-Privilege Rate ................................................ UPR
Weighted Structural Complexity .................................. WSC
ACKNOWLEDGMENTS

I would like to express my utmost gratitude to my advisor Professor Chuan Yue. His wisdom, insights, guidance and unwavering patience were all crucial in my research and my personal growth as a researcher. I hope that one day I can exhibit the same virtuous qualities that he has shown while mentoring me. I am also grateful to my committee members, Professor Tracy Camp, Professor Nils Tilton, Professor Bo Wu, and Professor Dejun Yang for their time and support.

I would also like to thank my family for their love and encouragement. I would like to thank my mother Ruth, a teacher who taught me the importance of education and to always continue learning. I would like to thank my father Wiley, who instilled in me the perseverance needed to sustain me during my years of research. Finally, I would like to thank my wife Elizabeth, words cannot express how grateful I am for her support and sacrifices made during countless late nights and weekends. Without her, my research and Ph.D. pursuit would not have been possible.
CHAPTER 1
INTRODUCTION

Access controls are the processes and mechanisms that allow only authorized users to perform operations upon the resources of a system. They allow administrators and resource owners to specify which users can access a system, what resources those users can access, and what operations those users can perform. Using access controls, administrators implement the Principle of Least Privilege (PoLP), a design principle where privileged entities operate using the minimal set of privileges necessary to complete their job. This protects the system against threats and vulnerabilities by reducing exposure to unauthorized activities and provide access only for those who have been approved. Although access control can be considered only one area of security research, it is the most pervasive and omnipresent aspect of information security [1]. Because the PoLP is so fundamental to secure design, it is specified in all widely accepted security compliance standards:

- Payment Card Industry (PCI) Data Security Standard (DSS) v3.1, Requirement 7: *Restrict access to cardholder data by business need to know.*

- Health Insurance Portability and Accountability (HIPAA), 164.312(a)(3)(ii)(B): *Implement procedures to determine that the access of a workforce member to electronic protected health information is appropriate.*

- National Institute of Standards and Technology (NIST) Special Publication 800-53, Security and Privacy Controls for Federal Information Systems and Organizations, AC-6: *The organization employs the principle of least privilege, allowing only authorized accesses for users (or processes acting on behalf of users) which are necessary to accomplish assigned tasks in accordance with organizational missions and business functions.*
• National Institute of Standards and Technology (NIST) Special Publication 800-171, Protecting Controlled Unclassified Information in Nonfederal Systems and Organizations, 3.1.5: *Employ the principle of least privilege, including for specific security functions and privileged accounts.*

• DOD Instruction 8500.2 Information Assurance (IA) Implementation, Control ECLP-1 Least Privilege: *Access procedures enforce the principles of separation of duties and “least privilege.”* Access to privileged accounts is limited to privileged users. Use of privileged accounts is limited to privileged functions; that is, privileged users use non-privileged accounts for all non-privileged functions.

As information systems have become more complex, access controls have also evolved to meet the diverse requirements of these information systems. Early access control models such as Access Control Lists (ACLs) consisting of a list of user permissions attached to each system object were sufficient for simpler systems. But these models are woefully inadequate for modern systems where it is not uncommon to deal with thousands of users with federated identities from multiple systems, each system with its own type of resources and operations, possibly using different access control models.

In modern systems, the complexity of managing access controls and implementing the PoLP often exceeds the capacity of manual management. While implementing the PoLP is a desirable and sometimes mandatory requirement for software systems, proper implementation can be difficult and is often not even attempted. Previous research into the use of least privilege practices in the context of operating systems [2] revealed that the overwhelming majority of study participants did not utilize least privilege policies. This was due to their partial understanding of the security risks, as well as a lack of motivation to create and enforce such policies.

In addition to information systems becoming more complex, they have also become more empowering for their users, increasing the possible damage that may be caused by access
control errors. For example, Cloud Computing provides cheap on demand access to computing and storage resources for its users. With this increased power also comes increased consequences of access control mistakes. The Amazon Simple Storage Service (S3) is just one of many popular cloud services. S3 provides the ability for users to easily and securely store data in the cloud and allow other users to read or modify that data. While the access controls and operations of the S3 service are relatively simple to understand and manage, there were at least seven major incidents in 2017 where the mismanagement of S3 access controls led to significant data breaches [3]:

- May 2017: Booz Allen Hamilton exposed battlefield imagery and administrator credentials to sensitive systems of the National Geospatial Agency (NGA).
- June 2017: Deep Root Analytics exposed personal data of 198 million American voters.
- July 2017: Dow Jones & Co. exposed personally identifiable information of 2.2 million people.
- July and September 2017: Verizon Wireless exposed personally identifiable information of over 6 million customers and sensitive corporate information.
- September 2017: BroadSoft exposed personally identifiable information of 4 million Time Warner Cable customers.
- September 2017: Accenture exposed hundreds of gigabytes of data, including private signing keys and plaintext passwords.

Another common class of security breaches resulting from poor access control and the power of cloud computing is cryptojacking attacks enabled by compromised cloud credentials. Cryptojacking is any attack involving the unauthorized use of computing resources to
mine cryptocurrency. The cloud computing form of cryptojacking attacks occur when users accidentally expose their cloud computing credentials such as in publicly shared source code. Attackers find these credentials and use them to mine cryptocurrency at the victim’s expense. Many such incidents have been documented in news articles with organizations such as Tesla [4], The L.A. Times [4], Gemalto [5], and Aviva [5] being just some of the documented victims of such attacks. These attacks are increasingly common with attackers continually searching open source code repositories such as GitHub for access keys [6]. Improved authentication methods may have prevented these attacks, but even with perfect authentication, insider threats and accidental misuse are still security issues. The PoLP helps reduce the damage possible from such threats. In the cryptojacking scenario, reducing the number of users that can create virtual instances or reducing the number of instances any single user can create alone would reduce the damage caused by such attacks.

It is important to note that these breaches are not the result of previously unknown vulnerabilities being exploited, nor due to the efforts of unusually capable and determined attackers. Instead, these are attacks of opportunity made possible by human errors in managing the access controls of an organization’s resources. The negative impacts of such access control misconfigurations are pervasive and growing. In 2017, security research firm RedLock found that 53% of organizations using cloud storage services such as Amazon S3 had inadvertently exposed one or more such services to the public. It appears that this is trending upwards despite growing awareness about the risks of misconfigurations [5]. The damage from such incidents may have been reduced or prevented all together by stricter adherence to the PoLP which would restrict the access to such resources to fewer people.

This thesis presents metrics, methods, and experimental results of using automated methods to implement least privilege access control policies across three separate but related projects. While the cloud computing environment is the focus of this work because of access to available data and because it is one of the most complex environments in terms of access control, the problems of access control errors are not unique to the cloud environment and
this work is relevant to addressing such problems in other environments as well.

Before describing solutions, we must first analyze and define the problem of automating least privileges. There exists a large body of work mining Role Based Access Control (RBAC) access control policies from existing permissions or audit logs in order to create the smallest (and most maintainable) RBAC policies with metrics to support these goals. However, these previous works have neglected to address methods and metrics for measuring the security of policies in terms of the least privilege. Instead of focusing on maintainability, we argue that the security of policies and their adherence to the PoLP is the most important goal when considering automated methods of building access control policies. Our first project, “Automated Least Privileges in Cloud-Based Web Services” provides an analysis of over-privilege present in the access control policies of a real world dataset. It also defines a methodology and metrics for quantifying the security of policies in terms of over-privilege and under-privilege. Unlike previous approaches which often treat access control policies and audit logs as fixed sets, our approach considers how these both change over time to better analyze the risk of over-privilege in policies.

In our second project, “Minimizing Privilege Assignment Errors in Cloud Services”, we implement three separate policy generation algorithms to create RBAC least privilege policies by mining a real world dataset of audit logs. Our algorithms consist of a naive approach, an unsupervised algorithm based on clustering, and a supervised algorithm based on machine learning classification. Using the same metrics and evaluation methodology as the first project, we analyze and compare the performance of these three algorithms. These metrics include a weighting that allows administrators to express how much they value minimizing under-privilege vs. minimizing over-privilege which we use to determine which algorithm performs ‘best’ as this weighting varies.

While RBAC is the de-facto access control model in government and industry, the Attribute Based Access Control (ABAC) is becoming more popular. ABAC provides the ability to create security policies using attributes that may be associated with users, objects, or the
operating environment. By using the wealth of attribute information in the audit logs and the greater expressive power of ABAC policies it is possible to create access control policies which simultaneously reduce under- and over-privilege when compared to RBAC. Creating such ABAC policies is the focus of our third project, “Mining Least Privilege Attribute Based Access Control Policies”. In this project, we implement an algorithm based on association rule mining techniques to create ABAC least privilege policies by mining a real world dataset of audit logs. We adapt the metrics of our previous works and use the same methods to evaluate policies over time in terms of under- and over-privilege errors. In addition to showing the effectiveness of our own algorithm, this project also provides a methodology and quantitative comparison showing the ability of ABAC to reduce under-privilege and over-privilege when compared to RBAC which may be valuable to access control researchers regardless of their interest in policy mining techniques.

The remainder of this chapter briefly describes each of these three projects, one in each subsection. Each project’s goals, methods, and results are described in detail in separate chapters of this thesis.

1.1 Automated Least Privileges in Cloud-Based Web Services

The PoLP is a fundamental guideline for secure computing that restricts privileged entities to only the permissions they need to perform their authorized tasks. Achieving least privileges in an environment composed of many heterogeneous web services provided by a third party is an important but difficult and error prone task for many organizations. This paper explores the challenges that make achieving least privileges uniquely difficult in the cloud environment and the potential benefits of automated methods to assist with creating least privilege policies from audit logs. To accomplish these goals, we implement two frameworks: a Policy Generation Framework for automatically creating policies from audit log data, and an Evaluation Framework to quantify the security provided by generated roles. We apply these frameworks to a real world dataset of audit log data with 4.3 million events from a small company and present results describing the policy generator’s effectiveness. Re-
results show that it is possible to significantly reduce over-privilege and administrative burden of permission management.

1.2 Minimizing Privilege Assignment Errors in Cloud Services

The PoLP 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.

1.3 Mining Least Privilege Attribute Based Access Control Policies

Implementing effective and secure access control policies is a significant challenge. Too much over-privilege increases the risk of damage to the system via compromised credentials, insider threats, and accidental misuse. Policies that are under-privileged prevent users from being able to perform their duties. Access control policies are rarely perfect in these regards and administrators must create policies that balance between the two competing goals of minimizing under-privilege vs. minimizing over-privilege. The access control model used to implement policies plays a large role in the ability to construct secure policies and the Attribute Based Access Control (ABAC) model continues to gain in popularity as the solution
to many access control use cases because of its advantages in granularity, flexibility, and usability. ABAC allows administrators to create access control policies based on the attributes of the users, operations, resource, and environment. Due to the flexibility of ABAC however, it can be difficult to determine which attributes and value combinations would create the best policies in terms of minimizing under- and over-privilege. To address this problem, we introduce a method of mining ABAC policies from audit logs to generate ABAC policies which minimize both under- and over-privilege. We also explore optimization methods for dealing with large ABAC privilege spaces, and present experimental results of our methods using a real-world dataset demonstrating the effectiveness of our methods.
CHAPTER 2
AUTOMATED LEAST PRIVILEGES IN CLOUD-BASED WEB SERVICES

2.1 Introduction

The commoditization of web services by cloud computing providers enables the outsourcing of IT services on a massive scale. The business model of providing software, platform, and infrastructure components via web services has seen tremendous growth over the last decade and is forecast to continue expanding at a rapid pace [7]. From small startups to large companies such as Netflix, Expedia, and Yelp [8], many organizations rely on services provided by a third party for their mission critical operations. While the adoption of these hosted web services continues, there are significant security and usability concerns yet to be solved. Privilege management is a key issue in managing the operation of the diverse array of web services available.

The principle of least privilege is a design principle where privileged entities operate using the minimal set of privileges necessary to complete their job [9]. Least privileges protect against several threats, primarily among them being the *compromise of privileged entities’ credentials and functions* by a malicious party. Other relevant threats mitigated by least privileges include *accidental misuse*, whereby privileged entities may delete or misconfigure resources which they do not require access to. Another threat is *intentional misuse*, where insiders can abuse over-privileges to cause more damage than they would be able to under a least privilege policy.

While implementing the principle of least privilege is a desirable and sometimes mandatory requirement for software systems, proper implementation can be difficult and is often not even attempted. Previous research into the use of least privilege practices in the context of operating systems [2] revealed that the overwhelming majority of study participants did not utilize least privilege policies. This was due to their partial understanding of the security
risks, as well as a lack of motivation to create and enforce such policies. In comparison to
the operating system environment, the use of third party web services present a much larger
number of services, resource types, access control policy languages, and audit mechanisms
even within a single service provider making it significantly more difficult to manage access
control.

The main contributions of this paper are: (1) an exploration of the challenges and ben-
efits of implementing an automated least privileges approach for third party web services
using real world data, (2) a concrete implementation of a framework for generating least
privilege policies from audit log data, and (3) metrics and methodology for quantifying the
effectiveness of least privilege policies. Related works are described in Section 2.2. The
motivating example of a real world dataset of manually created policies is analyzed in Sec-
tion 2.3. Automated least privilege generation and evaluation frameworks used are describe
in Section 2.4, the metrics used to evaluate adherence to PoLP are described in Section 2.5
and the results of our analysis are described in Section 2.6.

2.2 Related Work

Addressing the administrative burden of access control management is a well-studied
problem. While many access control models have been researched, Role Based Access Con-
trols (RBAC) remains a common model for implementing access control policies. The fun-
damental premise of RBAC is to create a set of permissions for each functional role required
to perform a job, and then assign privileged entities to these roles [10]. This allows policy
creators to reason about access controls in terms of privileges needed to perform a task and
the tasks an entity must perform.

A significant amount of work has been published on role mining methods which create
more maintainable RBAC policies from existing privilege assignments. The basic RMP uses
the minimal set of roles as the measure of goodness for deriving roles [11]. Alternatives to
the minimum number of roles as a goodness metric for role mining algorithms have also
been explored. A discussion of these alternative goodness metrics is given in [12], which
include measuring similarity with existing roles, minimizing the number of user-role assignment and permission-role assignment relations, metrics that seek to reduce administrative cost, weighted structures that assign adjustable weights to assignment relationships, and minimizing the number of edges in the role hierarchy graph.

Another related area of research uses audit data to create least privilege policies. Privileged entities often already possess the privileges necessary to do their jobs, thus roles can be derived from existing permissions via data mining methods [13]. Notable examples of mining data to create least privilege policies include EASEAndroid [14] for mobile devices, ProgramCutter [15] for desktop applications, and Passe [16] for web applications. However, these approaches do not provide a quantified assessment of how well they achieve the PoLP.

Like role mining, our research aims to reduce the administrative burden of creating access control policies. However, instead of seeking to make roles more easily maintainable, we seek to reduce administrator burden by generating secure and complete policies via easily and frequently repeatable automated methods. The focus of this research is directly on quantifying and improving the security of automatically generated privilege assignments regardless of their size and complexity, thus we are addressing a problem different from the RMP.

### 2.3 Over-Privilege in Manually Generated Policies

To illustrate the challenges of creating least privilege policies and to highlight the potential of using an automated approach to policy generation, we examine a real world dataset of policies manually created by administrators. The Amazon Web Services (AWS) CloudTrail [17] logs of a company which provides a Software as a Service (SaaS) product were analyzed (with permission). The audit logs contained 4.3M events collected over a period of 307 days. During this period, 37 unique roles and 15 unique users exercised privileges. Data gathered from the logs were analyzed and compared with the account Identity and Access Management (IAM) [18] policies as they existed at the end of the collection period. To quantify the effectiveness of these manually created policies at limiting over-privilege, we
compare the actions and services granted by these policies to those exercised in the audit log data.

The privileged entities considered in this paper are users and virtual machine instances which can both be assigned to roles. In our dataset, users were granted unconstrained access making their comparison with exercised privileges somewhat uninteresting, but also demonstrating a situation where achieving least privilege policies on users was not even attempted. In contrast to users, virtual machines in our dataset were not granted unrestricted access but were assigned roles manually created by administrators with the intent of constraining the virtual machines to least privilege policies. While data for both users and roles were analyzed, this section focuses on role policies granted to virtual machines to illustrate the over-privilege present in manually created policies. As the results show, over-privilege was common for these roles even though the role creators had the benefit of familiarity with the application and the privileges it required. Services and actions not supported by CloudTrails were excluded from these results.

Of the 37 unique roles identified in the dataset, 14 were present in the AWS IAM data at the end of the collection period (those not found in the IAM policies had been deleted during the collection period). Figure 2.1 shows a comparison between the actions granted and used by virtual machine roles during the observation period. Even though the policies for each role were intended to approximate least privileges, clearly there is a significant difference between the number of actions granted and number of actions used. The average number of actions granted to these 14 roles was 61.14, while the average number of privileges used was 2.92.

The comparison of privileges granted to those actually used at the service level of granularity is shown in Figure 2.2. Significant over-privilege is present at the service level, with every role being granted privileges to at least one service for which it did not perform any actions. The average number of services used by roles was 1.71 while the average number of services granted was 5.07.
The results presented in this section demonstrate the over/privilege present in a real world dataset of manually created policies with significant over-privilege present at both the action and service level for all virtual machine roles. Achieving least privilege policies for users was not even attempted in the dataset. These results underscore the difficulty and administrative burden of achieving least privilege policies in a cloud environment and provide motivation for an automated least privilege policy generation approach.

2.4 Policy Generation and Evaluation

This section describes the frameworks for generating and evaluating least privilege policies. First we present a framework for generating least privilege policies from audit logs. We then present a framework for evaluating the effectiveness of a policy generator.
The process of generating policies begins with ingesting the raw data audit logs for a given observation period into a datastore. Once ingested, the logs are normalized by creating a projection of the events onto each unique privileged entity identified in the audit logs for a specified observation period. Next, the policy generator algorithm is applied to the normalized data. The generator implemented for this paper uses a simple counting based approach which creates policy grants for each action an entity successfully exercised during the observation phase. After policy generation is complete, additional modifications may be made to the policies such as denying access to privileges which can be used to escalate privileges. The policy generation framework is a bottom-up approach to building RBAC policies where exercised permissions are used to create roles. This design can also be applied to audit log data that have been previously collected in an organization’s environment, and
does not require an active presence in the cloud environment during log collection.

We next implemented a framework for evaluating the generated policies. This evaluation framework simulates the application of an automated least privileges policy generator across varying observation periods and operation periods. The purpose of these simulations is to provide a quantified evaluation of the effectiveness of our current and future policy generators if they were to be adopted in production by an organization. The information obtained from these simulations can help determine how long the observation period should be, how long these generated policies should be used for, and how effective the policy generator is. For these evaluations we chose one day as the finest granularity of time period as this provides enough time for entities to complete tasks requiring related privileges.

![Figure 2.3: Sliding Window Evaluation](image)

The evaluation framework uses a sliding window approach to perform its duties. It repeatedly generates observation and operation phases of predetermined sizes and compares the policy generated during the observation phase to the privileges exercised during the operation phase. Each of these single evaluations is a trial and multiple trials for the same evaluation parameters are achieved by incrementing the dates of the observation phase and
operation phase by a fixed amount. Figure 2.3 provides a visual representation of how the sliding window technique is used to generate evaluation trials using the available audit log data.

2.5 Metrics

The PoLP implies two competing fundamental requirements. **Minimize Over-Privilege**: Privileged entities should not be granted more permissions than necessary to complete their tasks. **Minimize Under-Privilege**: Privileged entities should be granted all of the permissions that are necessary to complete their assigned tasks. Balancing between these requirements presents a trade-off between accepting risk and administrative overhead. To assess the effectiveness of automatically generated policies, we quantify their adherence to these requirements for meeting PoLP. We provide a variable weight to balance between these requirements so that organizations of automated least privilege policy tools can determine how to vary the observation length, operation length, and resource level restrictions depending on how much they value over-privilege vs. under-privilege.

To provide a quantitative evaluation we adopt terminology common to statistical hypothesis testing. The granting of a privilege by the policy generator is a positive prediction and the denial of a privilege is a negative prediction. For each evaluation trial, if the policy generated from the events of the observation phase granted a privilege which was exercised during the operation phase it is a **True Positive** (TP), while a granted privilege that was not exercised during the operation phase is a **False Positive** (FP). Similarly, if the automatically generated policy denied a privilege which was exercised during the operation phase it is a **False Negative** (FN). Privileges which were denied by the policy and not exercised during the operation phase are a **True Negative** (TN).

Precision and recall are metrics commonly used in hypothesis testing. **Precision** is the fraction of granted privileges that are exercised, high precision values indicate low over-privilege. **Recall** is the fraction of exercised privileges that are granted, high recall values indicate low under-privilege. The case where all privileges are denied is redefined to
be $Precision = 1$ because there is no possibility of over-privilege, and the case where all privileges are granted is redefined to be $Recall = 1$ because there is no possibility of under-privilege. To present more intuitive metrics, we take the compliment of precision and recall to create metrics where lower values are more favorable: the Over Privilege Rate ($OPR$) in Equation 2.1 and Under Privilege Rate ($UPR$) in Equation 2.2, respectively.

$$OPR = 1 - Precision = \frac{UnexercisedGranted}{AllGranted} \quad (2.1)$$

$$UPR = 1 - Recall = \frac{ExercisedDenied}{AllExercised} \quad (2.2)$$

It is important to consider the amount of time which over-privilege exists. While the cost of under-privilege is a decreased ability for privileged entities to perform their tasks, high over-privilege can result in compromises of confidentiality, integrity, and availability if the over-privilege is exploited by an attacker. The longer that over-privilege exists the greater the possibility of it being exploited, thus we introduce an additional weight on the $OPR$ to account for the amount of time which unused privilege grants existed. The Temporal Over Privilege Rate ($TOPR$) in Equation 2.3 is the $OPR$ multiplied by the number of days the privileges went unused (the length of the operation period).

$$TOPR = OPR \cdot OperationPeriodLength \quad (2.3)$$

$OPR$ and $UPR$ are two individual metrics for measuring the generated least privilege policies. To provide a single metric that weights minimal over-privilege vs. minimal under-privilege, we use the F-score metric (Equation 2.4). Higher $\beta$ values for the F-score indicate a higher weight for recall, which indicates a higher weight for minimal under-privilege. Lower $\beta$ values for the F-score weight minimal over-privilege higher. We use a temporally weighted version of the F-score, $TF_{\beta}$ (Equation 2.5), that accounts for the length of time which an over-privilege was granted. To incorporate a temporal weighting of over-privilege in $TF_{\beta}$, we divide the precision by the operation period length because precision is the compliment
of OPR and thus is directly tied to how we score over-privilege. Note that $F_\beta$ and $TF_\beta$ are equivalent for the finest granularity of the operation period which is one day in our simulations.

$$F_\beta = (1 + \beta^2) \cdot \frac{\text{Precision} \cdot \text{Recall}}{(\beta^2 \cdot \text{Precision}) + \text{Recall}} \quad (2.4)$$

$$TF_\beta = (1 + \beta^2) \cdot \frac{\text{Precision}}{\text{OperationPeriodLength}} \cdot \frac{\text{Recall}}{(\beta^2 \cdot \frac{\text{Precision}}{\text{OperationPeriodLength}}) + \text{Recall}} \quad (2.5)$$

The F-score is the harmonic mean of precision and recall. The advantage of using the harmonic mean F-score over arithmetic mean is that low scores for either precision or recall will result in an overall low F-score which avoids allowing extreme policies to achieve favorable scores. Consider an example policy which grants all privileges to an entity. This would result in a perfect score in terms of precision (1), but the worst possible score in terms of recall (0). The resulting F-score in this example would be 0 while arithmetic mean score would be 0.5, the same as if precision and recall were both 0.5. This equal scoring between an extreme policy and a balanced policy is not desirable in applications which values both precision and recall.

2.6 Results

This section presents the results of our analysis tying together all of the work described thus far. We consider the behavior of users and roles granted to virtual machines separately when evaluating the effectiveness of their policies because they have different usage patterns which produce significantly different scores. The behavior of virtual machines is fairly consistent in both the actions and resources used while users are less predictable.

2.6.1 Impact of Varying the Operation Period

The results of evaluating the least privilege policy generator for observation periods of 7 and 28 days as the operation phase varies from 1 to 7 days are shown for users in Figure 2.4 and for virtual machine roles in Figure 2.5. The results for both entity types show that as the
length of the operation phase increases, the \textit{UPR} also increases which is to be expected as privileged entities use privileges that were not observed during shorter operation phases. For virtual machine roles, there is very little difference between the \textit{UPR} for 7 days of operation vs. 28 days of operation. As we will see later in the metrics, the most variability in virtual machine permissions exercised occurs during the first few days of the observation phase.

As the operation phase increases entities are more likely to use privileges they may not have exercised previously during shorter periods. Thus the unweighted \textit{OPR} decreases for both entity types as the operation period increases. However, the \textit{TOPR} in Figure 2.4 increases as the operation phase increases, indicating that the new privileges exercised during each additional day of the operation phase do not reduce over-privilege enough to offset the over-privilege caused by leaving the unexercised privileges granted to the entities longer. The

![Figure 2.4: User Evaluation as Opr Days Vary (Obs Days=7,28)](image-url)

As the operation phase increases entities are more likely to use privileges they may not have exercised previously during shorter periods. Thus the unweighted \textit{OPR} decreases for both entity types as the operation period increases. However, the \textit{TOPR} in Figure 2.4 increases as the operation phase increases, indicating that the new privileges exercised during each additional day of the operation phase do not reduce over-privilege enough to offset the over-privilege caused by leaving the unexercised privileges granted to the entities longer. The
effect is more pronounced users than virtual machine roles - the virtual machine roles have lower \(TOPR\) scores for all operation and observation periods.

To determine a recommended operation period based on how much one values minimal over-privilege vs. minimal under-privilege, we use the \(TF_\beta\) metric (Formula 2.5). Figure 2.6 shows the combined \(TF_\beta\) score for both user and virtual machine role data for varying operation period lengths and \(\beta\) values. In these charts \(\beta = 10\) represents that minimal under-privilege is considered to be 10 times more important than minimal over-privilege while \(\beta = 0.1\) represents that minimal over-privilege is 10 times more important than minimal under-privilege. All of the calculated \(TF_\beta\) scores constantly decrease as the operation period increases indicating the smallest operation period of one day is the optimal choice for minimizing temporal weighted over-privilege and under-privilege. The higher \(\beta\) values show generally higher scores which decrease less as the operation period increases, indicating
that increasing the operation period would have a less negative impact for those that value minimal under-privilege.

### 2.6.2 Impact of Varying the Observation Period

Next we evaluate the impact of varying the observation period. The results of evaluating the automated least privilege policy generator for operation phases of lengths 1 and 7 days as the observation phase varies from 1 to 28 days are shown for users in Figure 2.7 and for virtual machine roles in Figure 2.8. As the observation period increases the \( UPR \) decreases for users at a logarithmic rate because more privileges exercised by users are captured during longer observation phases. For virtual machine roles however there is little benefit in increasing the observation period beyond two days as these virtual machines are unlikely to exercise additional privileges that have not been exercised after the first day of observation. For both
entity types the $UPR$ is again lower for the 1 day operation period vs. the 7 day operation period.

For both entity types the $OPR$ and $TOPR$ increase as the observation phase increases because longer observation phases result in entities being granted more privileges. This is intuitively obvious for users as they are likely to use some privileges periodically which are captured during the observation phase, and then not use them again for extended periods of time or at all during the operation phase. Although the virtual machine roles are unlikely to spontaneously use new privileges like users, not all privileges are exercised on a daily basis.

To determine a recommended observation period based on how much one values minimal over-privilege vs. minimal under-privilege, we again use the $TF_\beta$ metric. For this evaluation the user and virtual machine role scores are presented separately because (unlike varying
the operation phase in Figure 2.6) the dissimilar behavior patterns of users and virtual machines produce different recommended observation periods. Figure 2.9 displays the $TF_\beta$ scores for user entities as the observation phase varies and the operation phase remains fixed at one day. The decreasing scores for $\beta = 0.1, 1, 2$ imply that organizations which value minimal over-privilege should choose a lower observation period. Even if minimal under-privilege is valued twice as much as minimal over-privilege as indicated by $\beta = 2$, the $OPR$ rises significantly faster than the under-privilege rate decreases as the observation period increases (as shown in Figure 2.7). For $\beta = 5, 10$ the $TF_\beta$ increases as the observation period increases before eventually decreasing at 8 days for $\beta = 5$ and stabilizing at 13 days for $\beta = 10$ as the increasing $OPR$ outweighs the more heavily rated but slower to decline $UPR$. The $TF_\beta$ scores for virtual machine roles are presented in Figure 2.10. The role based scores
Figure 2.9: User $TF_\beta$ scores as Obs Days Vary (Opr Days=1)

for low $\beta$ again show that organizations which value minimal over-privilege should use small observation periods, while organizations which value minimal under-privilege will see little or no benefit in extending the observation period for these roles as the under-privilege rate showed little decline for observation periods over two days (as shown in Figure 2.8).

2.6.3 Summary of Results

The results of this section quantify the effectiveness of our policy generator applied to real world hosted web service audit log dataset. They describe how the performance of the policy generator is affected by varying the observation period and operation period. Based on this evaluation, we found that the actions of users were relatively difficult to predict compared to virtual machine roles with incidents of under-privilege being much higher for users. Virtual machines could be constrained to their actions used during their
first couple days operation to significantly reduce over-privilege present in their policies. For both types of privileged entities, increasing the operation period increased under-privilege while increasing the observation period increased over-privilege.

The conclusions drawn from these results are valuable because they quantify the performance that can be expected by adopting an automated least privilege approach and they provide a benchmark by which to judge future policy generation algorithms. The generation of these results also demonstrates the application of the policy generation and evaluation frameworks which can be used for evaluating future algorithms.

2.7 Summary

This paper explored the challenges and benefits of automating least privilege policies in a cloud computing environment. Previous research in role mining approaches in other envi-
environments were examined and unique aspects of automated role mining in a cloud computing environment were identified. A bottom-up design to generate least privilege policies was implemented to illustrate the potential of an automated least privilege approach and the results of evaluation on real world audit log data were presented. The results showed that even when administrators attempt to manually create least privilege policies there is significant room for improvement upon these policies. Metrics for evaluating the effectiveness of least privilege policy generators were presented for the same data set. These results showed the trade-offs between over-privilege and under-privilege that can be achieved by varying the observation period, operation period, and resource constraints for the presented policy generator and these results provide benchmarks for future policy generators to be evaluated against.
CHAPTER 3
MINIMIZING PRIVILEGE ASSIGNMENT ERRORS IN CLOUD SERVICES

3.1 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 [19]. 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.

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 [20], and is considered a fundamental access control principle in information security [1]. 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 [2]. 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 3.3.

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 $\beta$ 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 $\beta$ 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 3.2 reviews related works on role mining and automated least privileges. Section 3.3 presents a comparison of the privilege spaces of various environments and a description of our dataset. Section 3.4 formally defines the PEMP and a scoring metric for evaluating how well it is solved. Section 3.5 details specific algorithms and methods used in our approach to addressing the PEMP and Section 3.6 analyzes the results of these algorithms. Section 3.7 concludes this work and discusses potential research areas for future work.

3.2 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 [22]. The problem of discovering an optimal set of roles from existing user permissions is referred to as the Role Mining Problem (RMP) [23].

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 [22] 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 [13]. 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 [14] for mobile devices, ProgramCutter [15] for desktop applications, and Passe [16] for web applications. [25] used Latent Dirichlet Allocation (LDA), a machine learning technique to create roles from source code version control usage logs. In [26], 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 [25], [26] and [27] 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 shortcomings using the rolling forecasting and sliding simulation methods discussed in Sections 3.4.3.2 and 3.5.3, 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 [27] which we expand upon and use the naive algorithm presented in that work for comparison purposes.

### 3.3 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 [28] 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 [29], 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 [30].

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 3.1. 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 ∑ *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 ∑ Actions metrics to understand the variation between individual users. For example, looking at both the Unique and ∑ 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 3.1: One Year Total Usage of our Dataset

<table>
<thead>
<tr>
<th>Metric</th>
<th>First Month</th>
<th>Last Month</th>
<th>One Year</th>
</tr>
</thead>
<tbody>
<tr>
<td>Users</td>
<td>7</td>
<td>13</td>
<td>18</td>
</tr>
<tr>
<td>Unique Services Avg.</td>
<td>5.86</td>
<td>8.08</td>
<td>13.50</td>
</tr>
<tr>
<td>Unique Services StdDev.</td>
<td>2.97</td>
<td>5.22</td>
<td>9.04</td>
</tr>
<tr>
<td>Unique Actions Avg.</td>
<td>13.71</td>
<td>45.31</td>
<td>88.78</td>
</tr>
<tr>
<td>Unique Actions StdDev.</td>
<td>20.21</td>
<td>48.13</td>
<td>91.99</td>
</tr>
<tr>
<td>∑ Actions Avg.</td>
<td>91.97</td>
<td>78.38</td>
<td>238.30</td>
</tr>
<tr>
<td>∑ Actions StdDev.</td>
<td>299.89</td>
<td>261.95</td>
<td>1271.15</td>
</tr>
</tbody>
</table>

3.4 Problem Scope and Approach

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

3.4.1 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 [31].

- *USERS, OPS, and OBS* (users, operations, and objects, respectively).
- *PRMS = 2OPS×OBS*, the set of permissions
- *UPA ⊆ USERS × 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.
3.4.2 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 3.5.

3.4.3 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 [32], 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.

3.4.3.1 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 [32]. Precision and recall are defined as follows[32]:

\[
\text{precision} = \frac{TP}{TP + FP} \quad (3.1)
\]
\[
\text{recall} = \frac{TP}{TP + FN} \quad (3.2)
\]

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 $\text{precision} = 1$. 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 $\text{recall} = 1$. 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[32]. 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), $\beta$. High $\beta$ values increase the importance of recall, while low $\beta$ values increase the importance of precision. The weighted measure, $F_\beta$ is defined in Equation 3.3.

$$F_\beta = \frac{(1 + \beta^2) \cdot \text{Precision} \cdot \text{Recall}}{(\beta^2 \cdot \text{Precision}) + \text{Recall}} \quad (3.3)$$

The $\beta$ 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 [33], 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.

### 3.4.3.2 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 restric-
tions 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[32]. 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 [34]. Suppose k observations are required to produce a reliable forecast. Then rolling forecasting origin works as follows [34].

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 \(F_\beta\) score for a given operation period described in Section 3.4.3.1, 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 $F_\beta$ measure across all operation periods:

$$\frac{1}{T-k} \sum_{i=1}^{T-k} F_\beta(Precision_i, Recall_i)$$  \hspace{1cm} (3.4)

3.5 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 3.4.3.2.

3.5.1 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 $F_\beta$ score, particularly for low $\beta$ values which value minimizing over-privilege. Furthermore, in a low $\beta$ 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. $UPA \leftarrow \emptyset$
2. for $user, perm \in UPE$ do
3. \hspace{1em} $UPA_{user} \leftarrow roles_{user} \cup perm$;
4. end
5. return $UPA$

3.5.2 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
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. \( UPA, \text{documents} \leftarrow \emptyset; \)
2. \( \text{for user, perm} \in UPE \text{ do} \)
3. \( \quad \text{documents}_{\text{user}} \leftarrow \text{documents}_{\text{user}} \cup \text{perm}; \)
4. \( \text{end} \)
5. \( \text{clusters, outliers} \leftarrow \text{DBSCAN}(\text{documents}); \)
6. \( \text{for cluster} \in \text{clusters do} \)
7. \( \quad \text{role} \leftarrow \emptyset; \)
8. \( \quad \text{for user, document} \in \text{cluster do} \)
9. \( \quad \quad \text{for perm} \in \text{document do} \)
10. \( \quad \quad \quad \text{role} \leftarrow \text{role} \cup \text{perm}; \)
11. \( \quad \text{end} \)
12. \( \text{UPA}_{\text{user}} \leftarrow \text{role}; \)
13. \( \text{end} \)
14. \( \text{for user, document} \in \text{outliers do} \)
15. \( \quad \text{for user, perm} \in \text{document do} \)
16. \( \quad \quad \text{UPA}_{\text{user}} \leftarrow \text{roles}_{\text{user}} \cup \text{perm}; \)
17. \( \quad \text{end} \)
18. \( \text{end} \)
19. \( \text{return UPA} \)
documents in information retrieval [35]. 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 [36], an implementation of the algorithm originally published in [37]. The DBSCAN algorithm has several advantages for our scenario, primary among them being that we do not need to specify the expected 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 [38]. There is one relevant hyper-parameter for DBSCAN which we vary in our policy generation experiments, $\epsilon$, which is the maximum distance between two samples for them to be considered as in the same cluster. We explore three methods for calculating $\epsilon$: the mean distance between all points, median distance between all points, and middle point between the minimum and maximum points in the vector space.

### 3.5.3 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 3.5.3.2. 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.

1. $UPA \leftarrow \emptyset$
2. for $tParams \in \text{permute}(TSP)$ do
   3.  $\text{featureVector, labelSet} \leftarrow \text{createTrainingSet}(tParams, UPE)$;
   4.  for $clfParams \in \text{permute}(CAP)$ do
       5.      $clf \leftarrow \text{decisionTree}(clfParams)$;
       6.      $clf \leftarrow clf.train(\text{featureVector, labelSet})$;
       7.      for $pParams \in \text{permute}(PGP)$ do
               8.         $\text{roles} \leftarrow \emptyset$
               9.         $\text{possiblePrivs} \leftarrow \text{createPossiblePrivs}(pParams, PRMS)$;
             10.        for $\text{user, perm} \in \text{possiblePrivs}$ do
                        11.           if $\text{clf.predict(user, perm)} == \text{'granted'}$ then
                             12.               $\text{roles}_{\text{user}} \leftarrow \text{roles}_{\text{user}} \cup \text{perm}$;
                        end
                   end
               end
       13.   end $UPA_{tParams, clfParams, pParams} \leftarrow \text{roles}$;
  end
15. end
16. return $UPA$
3.5.3.1 Classification Algorithm and Feature Selection

We use a decision tree (DT) classification algorithm for supervised learning, also from the scikit-learn library [36]. The algorithm implemented in the library is an optimized version, an implementation of the CART algorithm published in [39]. 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.

3.5.3.2 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 [40].

The sliding simulation method of [40] 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 [40] showed that it outperformed the best method of a previous competition in statistical forecasting (the M-Competition [41]) 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 $F_\beta$ score presented in Section 3.4.3.1 as our scoring criteria. While [40] 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.

### 3.5.4 Model Decomposition

Time series decomposition is a common technique used to improve predictions [34], 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 $\beta$ 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.

### 3.6 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.

#### 3.6.1 Complete Model Results

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
Figure 3.1: Receiver Operating Characteristic Curves

(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 $F_\beta$ described in Section 3.4.3.1. The ROC curves for the naive, three unsupervised (DBSCAN) and one supervised (DT) algorithms across multiple observation period lengths are presented in Figure 3.1. 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
Figure 3.2: Beta Values Curves

A lower FPR for all data points. The use of specific observation period sizes for the sliding window method described in Section 3.4.3 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 $F_\beta$ values for $1/100 \leq \beta \leq 100$ is presented in Figure 3.2. 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 $\beta$ 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 $\beta$, 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 $\beta$ values. The supervised algorithms score relatively poorly for $\beta > 1$. For $\beta$ values where $\beta < 1$, the supervised algorithms
score significantly better than the naive algorithm as $\beta$ decreases with the performance gap widening until $\beta < 1/30$, where the scores of the supervised and naive algorithms cease to improve as $\beta$ 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.

Figure 3.1 and Figure 3.2 show the scores of algorithms regardless of the size of the observation period. We next examine the performance of these algorithms for fixed $\beta$ values as the observation period size varies. We chose values $\beta = 80$ and $\beta = 1/10$ because these 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 3.3. The choice of $\epsilon$ 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 $\epsilon$ 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 $\epsilon$ 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 3.4. 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
3.6.2 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 3.5.4.

The performance of the complete and decomposed models for $\beta$ values $\geq 1$ for both the naïve algorithm and the unsupervised algorithm (with the average method for calculating $\epsilon$) are shown in Figure 3.5. For both algorithms, the weekday model performance is superior to the complete model for $\beta$ values $\geq 1$. The trend previously illustrated in Figure 3.2 of
the unsupervised algorithm under-performing the naive algorithm for low $\beta$ but eventually outperforming it as $\beta$ 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.

The performance of the complete and decomposed models for $\beta$ values $\leq 1$ for both the naive algorithm and the supervised algorithm (using the SND labeling method) are shown in Figure 3.6. As with the unsupervised algorithm and $\beta$ values $\geq 1$, the weekday model outperforms the complete model while the weekend model under-performs the complete model where $\beta$ values $\leq 1$ 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.

### 3.6.3 Recomposed Models Results

Section 3.6.2 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.

The performance of the complete and recomposed models for $\beta$ values $\geq 1$ for both the naive algorithm and the unsupervised algorithm (with the average method used for calculating $\epsilon$) are shown in Figure 3.7. For the unsupervised algorithm, the recomposed model outscores the complete model for $\beta$ values $\geq 5$, and outscores the naive algorithm for both the complete and recomposed models for $\beta \geq 50$, with the performance gap increasing after that as $\beta$ 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 $\beta$ 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 3.7: Recomposed Models, $\beta \geq 1$

The performance of the complete and recomposed models for $\beta$ values $\leq 1$ for both the naive algorithm and the supervised algorithm (using the SNDT labeling method) are shown in Figure 3.8. 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 $\beta < 1/2$. 

54
3.6.4 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 $\beta$ value as described in Section 3.4.3. 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 $F_\beta$ 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 $\beta$ 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 $\beta$ values. Conversely, the supervised algorithm performs more favorably as $\beta$ decreases but poorly for large $\beta$ 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 $\beta$ 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.

3.7 Summary

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 auto-
tomated 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 ex-
ercised. 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 organiza-
tional 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.
CHAPTER 4
MINING LEAST PRIVILEGE ATTRIBUTE BASED ACCESS CONTROL POLICIES

4.1 Introduction

Access control is a key component of all secure computing systems but implementing effective and secure access control policies is a significant challenge. Access control policies are predictions about which privileged entities will exercise specific operations upon specific objects under various conditions and accurately predicting the future is always difficult. Too much over-privilege increases the risk of damage to the system via compromised credentials, insider threats, and accidental misuse. Policies that are under-privileged prevent users from being able to perform their duties. Both of these conflicting goals are expressed by the principle of least privilege which requires every privileged entity of a system to operate using the minimal set of privileges necessary to complete its job [20]. The principle of least privilege is a fundamental access control principle in information security [1] and is a requirement in security compliance standards such as the Payment Card Industry Data Security Standard (PCI-DSS), Health Insurance Portability and Accountability Act (HIPAA) and ISO 17799 Code of Practice for Information Security Management [21].

Many access control models have been introduced to address the challenges of creating and administrating secure and effective access control policies, with different approaches attempting to balance between the competing goals of ease of use, granularity, flexibility, the ability to leverage aspects unique to a specific domain, and scalability. Access control models are constantly evolving, but Attribute Based Access Control (ABAC) continues to gain in popularity as the solution to many access control use cases because of its flexibility, usability, and ability to support information sharing across disparate organizations. ABAC allows security policies to be created based on the attributes of the user, operation, and environment at the time of an access request.
The flexibility of ABAC policies is both a major strength and a hindrance. With the ability to create policies based on many attributes, administrators face difficult questions such as what constitutes “good” ABAC policies, how to create them, and how to validate them? Additionally, the ABAC privilege space of a system can be extremely large, so how can administrators determine which attributes are most relevant in their systems? We address these issues by taking a rule mining approach to create ABAC policies from audit logs. Rule mining methods are a natural fit for creating ABAC policies because security policies are a set of rules regarding the actions that users can perform upon resources. By identifying common patterns of usage between the attributes and values from audit logs, rules can be created based on an organization’s acceptable level of risk regarding under- vs. over-privilege. By using out-of-sample validation to evaluate the effectiveness of the generated policies on a dataset of 4.7M Amazon Web Service (AWS) log events, our experiments show that our rule mining based approach is effective at generating policies which minimize the instances of under-privilege (which allows users to perform their necessary duties), while also minimizing over-privilege (which reduces security risks to the system).

We address the problem of creating least privilege ABAC policies using rule mining techniques in this research through the following contributions: 1) a definition for the ABAC Privilege Error Minimization Problem ($P_{EMP_{ABAC}}$) which addresses balancing between under- and over-privilege errors in security policies, 2) an algorithm for automatically generating least privilege ABAC policies from mining audit logs, 3) an algorithm for scoring ABAC policies using out-of-sample validation, 4) feature selection, scalability, and performance optimization methods for processing large ABAC privilege spaces, 5) a quantitative analysis of the performance of our mining algorithm using a real-world dataset consisting of over 4.7M audit log entries, and 6) a performance comparison of automatically generated ABAC policies created by our mining algorithm with automatically generated role based policies.
The rest of this paper is organized as follows. Section 4.2 provides background information on the ABAC model and rule mining methods. Section 4.3 reviews related work specific to mining access control policies. Section 4.6 formally defines the ABAC version of the privilege error minimization problem of mining ABAC policies with minimal under- and over-privilege assignment errors and defines metrics for evaluating policies. Section 4.7 details specific algorithms and methods used in our approach for addressing the problem defined in Section 4.6. Section 4.8 analyzes the results of applying our algorithms to a real-world dataset. Section 4.9 concludes and discusses potential future work.

4.2 Background

4.2.1 Attribute Based Access Control (ABAC)

4.2.1.1 ABAC Definition

NIST defines ABAC as “An access control method where subject requests to perform operations on objects are granted or denied based on assigned attributes of the subject, assigned attributes of the object, environment conditions, and a set of policies that are specified in terms of those attributes and conditions” [42]. Attributes are any property of the subjects, objects, and environment encoded as a name:value pair. Subjects may be a person or non-person entity (such as an autonomous service), objects are system resources, operations are functions executed upon objects at the request of subjects and environment conditions are characteristics of the context in which access requests occur and are independent of subjects and objects [42].

4.2.1.2 ABAC Benefits

Allowing any property to be encoded instead of restricting the model to predetermined attributes and relationships between objects gives ABAC unlimited flexibility. Because of this flexibility, ABAC is able to implement other access control models such as Discretionary Access Control (DAC), Mandatory Access Control (MAC), and Role Based Access Control (RBAC).
By using identity federation and basing access decisions on policies using an abstracted common set of attributes, decisions can be externalized with policies established across organizational boundaries [43]. Because of these characteristics, the Federal Identity, Credential, and Access Management (FICAM) Roadmap 2.0 called out ABAC as a recommended access control model for promoting information sharing between diverse and disparate organizations [42].

4.2.1.3 ABAC vs. RBAC

The Role Based Access Control (RBAC) model has been the de-facto access control standard for industry and academia for more than two decades [44]. Using RBAC, administrators identify privileges needed for common job functions, create roles for each function and assign users to their appropriate roles for performing their duties. This simplifies the administrators’ task compared to DAC and provides more granularity than MAC.

However, as access control needs have become more complex and applied to more diverse domains, organizations have found that RBAC does not provide sufficient granularity, becomes too difficult to manage, or does not support their information sharing needs. Organizations facing these challenges may meet them using an ABAC based system. Consider the case of an administrator that wishes to restrict operations needed for performing a database backup to a specific maintenance window timeframe and a specific location or IP address range. Such constraints can be easily expressed using ABAC attributes, but cannot be expressed using only the user, operation, and object semantics of the RBAC model. Another common problem with RBAC is “Role Explosion”, where the need to define and assign users many roles to access diverse sets of different applications within an organization makes maintenance of the many roles unmanageable. ABAC is able to address this problem by defining policies based on user attributes (for example their job title, supervisor, or skill set in an HR database) so that access control decisions are made according to attributes of the user at the time of the access request.
4.2.2 Rule Mining Methods

Frequent itemset mining and association rule mining are two popular rule mining methods for identifying patterns in commercial databases [45] with applications in many diverse fields. Frequent itemset mining is the first step in association rule mining and is a deterministic method that identifies common patterns in a database of transactions. The frequent itemset problem is defined as follows, given a transaction database \( DB \) and a \textit{minimum support threshold} \( \epsilon \), find the complete set of frequent patterns in the database. The set of items is \( I = \{a_1, ..., a_n\} \) and a transaction database is \( DB = (T_1, ..., T_m) \), where \( T_i(i \in [1...m]) \) is a transaction which contains a set of items in \( I \). The \textit{support} of a pattern \( A \) (where \( A \) is a set of items), is the fraction of transactions containing \( A \) in \( DB \), \( \text{support}(A) = \frac{|\{T_i \in DB | A \subseteq T_i\}|}{|DB|} \). A pattern is \textit{frequent} if \( A \)'s support is \( \geq \epsilon \), (the minimum support threshold) [46].

Association rule mining uses itemsets identified by a frequent itemset mining algorithm to identify rules of the form \( X \Rightarrow Y \) where \( X \subset I, Y \subset I \), and \( X \cap Y = \emptyset \). The first itemset \( X \) is the “antecedent” and the second itemset \( Y \) is the “consequent”. The confidence of a rule \( X \Rightarrow Y \) is the proportion of the transactions that contain \( X \) which also contain \( Y \), \( confidence(X \Rightarrow Y) = \frac{\text{support}(X \cup Y)}{\text{support}(X)} \). Given a transaction database \( DB \), minimum support threshold \( \epsilon \) and minimum confidence \( c \), the association rule mining problem is to find all of the rules in \( DB \) that have \( \text{support} \geq \epsilon \) and \( confidence \geq c \) [47].

The output of frequent itemset mining is many subsets of items that occurred within the transaction database \( DB \), while the output of association rule mining is two subsets \( (X \Rightarrow Y) \) implying the probability that \( Y \) occurs in the transaction database given \( X \). In the context of creating security policies, there is a clear translation of frequent itemsets into ABAC rules. Just as frequent itemsets state whether a pattern occurred or not (with a given \( \text{support} \geq \epsilon \)), security policies must make a binary decision about whether a request should be allowed or denied.
4.3 Related Work

We group related work into two categories: those that deal with generating RBAC least privilege policies, and those that address the problems of modifying existing ABAC policies or creating ABAC policies of minimal size. To the best of our knowledge, our work is the first to address the problem of automatically creating least privilege ABAC policies.

4.4 Least Privilege Policy Generation

We first consider the set of related works which generate RBAC least privilege policies from audit logs. In [48], the authors formally define the Privilege Error Minimization Problem (PEMP) which seeks to minimize the under-privilege and over-privilege assignment errors of a policy put into operation. Naive, unsupervised learning, and supervised learning algorithms are designed to mine RBAC policies using attribute information from audit logs. Policy evaluation was performed by using out-of-sample validation over discretized time periods. Our work uses a similar evaluation method but designs a rule mining algorithm to generate ABAC policies. With the ability to use attributes in mined policies (vs. user, operation, and resource ids only in RBAC), we are able to generate policies that simultaneously reduce both under- and over-privilege when compared to RBAC policies in [48].

Another important work in generating least privilege policies is [25] which used Latent Dirichlet Allocation (LDA) to create least-privilege RBAC policies from source code version control usage logs. This method also used user attribute information in the mining process although the resulting policies were RBAC policies. In [25], the authors introduced the $\lambda$–Distance metric for evaluating candidate rules, which added the total number of under-assignments to the total number of over-assignments with $\lambda$ acting as a weighting factor on the over-assignments to specify how much the metric values over-privilege vs. under-privilege for a particular application.

Because the under- and over-assignments in $\lambda$–Distance are not normalized before being added, it is easy for one side to dominate the equation. Extreme changes in $\lambda$ may be needed
to trade off between under- and over-privilege, or slight changes to \( \lambda \) may cause extreme changes in the resulting policies depending on the sizes of the log entries and privilege space. This makes it difficult for an administrator to choose a \( \lambda \) value which accurately captures their organization’s desired balance between under- and over-privilege.

4.5 ABAC Policy Mining

4.5.1 ABAC Rule Mining Based Works

One early work on applying association rule mining to ABAC policies was [49], which used the Apriori algorithm [50] to detect statistical patterns from access logs of a set of lab doors in a research lab. The dataset consisted of 25 physical doors and 29 users who used a smart-phone application and Bluetooth to open the doors. The authors used the output of the mining algorithm to identify policy misconfigurations by comparing mined rules with existing rules. The performance of the algorithm was measured in terms of the trade-off between success in detecting and guiding the repair of misconfigurations vs. the inconvenience to users of suggesting incorrect modifications to policies. The dataset used in [49] was "somewhat small" as the authors noted, leaving questions as to its scalability in terms of users and attributes where we use a much larger dataset in terms of the number of events and attributes.

Another work, [51] presents a tool named Rhapsody which builds upon Apriori-SD [52], a version of the Apriori algorithm modified for subgroup discovery. This work is similar to our own in that it also seeks to create ABAC policies of minimal over-privilege by mining logs however, it does not provide a weighting method for balancing between under-privilege and over-privilege, nor does it consider large and complex privilege spaces. Instead, Rhapsody uses a simpler model of attributes of Users and Permissions only instead of the Users, Operations, Resources and Environment attributes we use. Rhapsody uses a metric termed reliability to quantify the confidence of a rule and all its significant refinements to assist in simplifying and reducing over-privilege of policies. While Rhapsody is designed to operate on “sparse” audit logs where only a small amount (\( \leq 10\% \)) of all possible log entries are
likely to occur in the mined logs, our work is designed to operate on logs several orders of magnitude more sparse than those of Rhapsody using optimization techniques described in Section 4.7.3. One important limitation of Rhapsody is that run-time grows exponentially with the maximum number of rules a request may satisfy, limiting number of attributes that can be considered to “less than 20” [51], which would prevent a direct comparison using our dataset of over 1,700 attributes. We also employ different metrics and scoring methodology for evaluating policies compared to Rhapsody. The authors of [51] use the F-score metric which was suitable for RBAC policy evaluation in our previous works, but we found to be too dominated by the Precision component when scoring ABAC policies so we have chosen to evaluate policies in terms of True Positive Rate and False Positive Rate separately. Furthermore, we use a sliding window approach to evaluate policies over time which retains their temporal dependencies vs. the random sampling cross-validation approach used in [51].

In [53], the authors used association rule mining to mutate existing ABAC policies as a moving target defense against attackers who could compromise values of attribute stores (with stores possibly distributed across multiple organizations). By expanding an existing policy with new rules that use highly correlated attributes identified by using association rule mining techniques on audit logs, this method provides additional protection in the event that attribute values used by the original policy rules are compromised. While [53] also used rule mining of audit logs, it did not create new policies nor did it aim to achieve least privilege policies. Experimental results dealt with identifying correlations between attributes but the analysis of the security of the results was qualitative so there were no metrics of goodness similar to ours to use as a comparison between [53] and this work.

### 4.5.2 ABAC Policy Minimization Works

A few papers have been published to address the **ABAC Mining Problem** which deals with finding ABAC policies of minimal size given a set of authorizations or audit log entries. The ABAC Mining Problem was addressed by Xu & Stoller in [54], then formally defined by another group of researchers in [55]. The metric for evaluating the minimal size of ABAC
policies in these works is Weighted Structural Complexity (WSC), which was introduced in [56] to measure the size of RBAC policies and adapted to ABAC policies in [57].

In [54], Xu & Stoller presented an algorithm for mining ABAC policies from operational logs. Their algorithm attempts to create policies that cover all the entries found in an audit log while also minimizing the number of over-assignments and the WSC of the policy through a process of merging and simplifying candidate rules. The authors defined $Q_{rul}$ (Equation 4.1), a quality metric for evaluating candidate rules during the mining process. In this quality metric, $|\{p\}|$ is the number of user-permissions in the possible permission universe covered by a candidate rule $p$. $|\{p\} \cap UP|$ represents the number of user-permissions in the logs covered by $p$, but not covered by existing rules in the policy. $WSC(p)$ represents the WSC score of rule $p$. The number of over-assignments granted by the rule is $|\{p\} \setminus UP(L)|$ where $L$ is the operation log. Balancing between the number of over-assignments produced by a rule $p$ and the number of log entries covered by $p$ is achieved by varying the over-assignment weight, $\omega'_0$.

\[
Q_{rul}(p, UP) = \frac{|\{p\} \cap UP|}{WSC(p)} \times \left(1 - \frac{\omega'_0 \times |\{p\} \setminus UP(L)|}{|\{p\}|}\right)
\] (4.1)

The authors of [55] define the ABAC Mining problem as follows: Given the set of authorizations $A$, the set of user attribute conditions ($U_C$), and the set of object attribute conditions ($O_C$), the ABAC Mining Problem is to discover the minimum set of access rules $\Pi$ such that there exists a rule $r \in \Pi$ where user $u$ is allowed to perform permission $p$ on object $o$ iff $a = \langle u, o, p \rangle \in A$. Two algorithms are presented for addressing this problem: an exponential run-time algorithm based on identifying functional dependencies between attributes, and an asymptotic run-time bottom-up algorithm based on finding more general rules from a candidate set of rules. As in [54], the authors use WSC to evaluate their mined policies, but with the important difference that no over-privileges are allowed in the mined policies while [54] optionally allowed over-assignments using the variable over-assignment weight, $\omega'_0$.
Both [54] and [55] mine rules and calculate coverage based on user-permission tuples, where a tuple $\langle u, o, r \rangle$ contains a user, operation, and resource only, instead of considering all of the valid attribute combinations in the privilege space. This reduces the computational complexity of mining and evaluating rules, but unfortunately presents a problem for accurately evaluating ABAC policies because such a tuple may be both allowed and denied unless considering all the attributes of the user, operation, and resource at the time of the user request. The authors of [55] identify and address this problem by denying all instances of a tuple if any single instance of that tuple is denied. This significantly reduces the granularity and flexibility advantages of the ABAC model. This issue is further complicated when evaluating policies over time because attribute values may change. To address these problems, we base our metrics on the entire ABAC privilege space of valid attribute:value pairs instead of the individual users, operations, and resources.

Another key difference between our work and all previous works cited in ABAC mining is the evaluation of policies for least-privilege over time. None of the previous works on ABAC policy mining captured the performance of mined policies in terms of under-privilege vs. over-privilege when put into operation, which we contend is the most important measure of a security policy. We use out-of-sample validation on a real world dataset to evaluate the under-privilege and over-privilege rates of policies over time using a sliding window of observation and operation periods, a method originally described in [48]. While minimizing complexity (evaluated by WSC) is desirable in that it makes policies easier to maintain by administrators, we see it as less important than least privilege performance over time. This is especially true when using automated methods to build policies where less administrator involvement is necessary. Methods for minimizing ABAC policy complexity are complementary to our work as once least privilege policies are identified, then methods for minimizing policy complexity can be applied.
4.6 Problem Definition and Metrics

The problem we address in this paper is minimizing privilege assignment errors in ABAC policies. Access control can be viewed as a prediction problem. The statements that comprise a policy are predictions about which entities should be granted privileges to perform specific operations upon the specific resources necessary to perform their jobs. The goal of this work is to automatically generate policies that are accurate access control predictions. There have been many access control related papers with similar but not entirely the same goals. To help clarify the specific problem this paper addresses we formally define it as the ABAC Privilege Error Minimization Problem (\(PEMP_{ABAC}\)) in this section. We also define specific metrics to be used in evaluating the performance of proposed solutions (in the form of ABAC policies).

4.6.1 Problem Definition

Our problem definition is based on the Privilege Error Minimization Problem (PEMP) originally defined in [48]. The PEMP defined the problem of creating least privilege RBAC policies which consisted of users, operations, and objects. Like the original PEMP, our problem seeks to minimize the under- and over-privilege assignment errors in policies and uses the notions of observation and operation periods for evaluation. However, users, operations, and resources are only some of the attributes available when creating ABAC policies so we adapt the problem definition to the ABAC privilege space.

The size of an ABAC privilege space is determined by the attributes and values of valid ABAC policies. \(A\) is the set of valid attributes which can be used in policies. As in other ABAC mining works [49, 53, 54], we assume all attributes and values present in the logs can also be used in building policies. Each individual attribute \(a_i \in A\) has a set of atomic values \(V_i\) which are valid for that attribute. All values for an attribute are the attribute’s range \(\text{Range}(a_i) = V_i\). The Cartesian product of all possible attribute:value combinations is \(\xi = V_1 \times \ldots \times V_n = \{(v_1, \ldots, v_n) | v_i \in V_i \text{ for every } i \in \{1, \ldots, n\}\}.\) However, some attribute:value
pairs are not valid when present in combination with other \textit{attribute:value} pairs because of dependencies between them. For example, some operations are only valid on certain resource types so combinations including both \textit{operation:DeleteUser} and \textit{resourceType:File} are not be valid. The valid privilege universe $\xi'$ is the set of all possible \textit{attribute:value} combinations when considering the dependency relationships between all attributes and values.

Any measure of security policy accuracy must also take time into account because the amount of risk from over-privileges accumulates over time. Over-privilege carries the risk that an unnecessary privilege will be misused, and this risk increases the longer the over-privilege exists. To capture risk across a specified time period, we define the \textit{Operation Period} (\textit{OPP}) as the time period during which security policies are evaluated against user operations. With the concepts of the valid privilege universe $\xi'$ and operation period \textit{OPP} defined, we now define the ABAC specific version of the Privilege Error Minimization Problem $PEMP_{ABAC}$ (Definition 1).

\textbf{Definition 1.} $PEMP_{ABAC}$: ABAC Privilege Error Minimization Problem. \textit{Given the universe of all valid attribute:value combinations $\xi'$, find the set of attribute:value constraints that minimizes the over-privilege and under-privilege errors for a given operation period OPP.}

\subsection*{4.6.2 Evaluation Metrics}

We use terminology from statistical hypothesis testing for assessing the effectiveness of our algorithm in addressing the $PEMP_{ABAC}$. We first present our method for scoring individual predictions, and then our method for splitting up the dataset and evaluating the algorithm’s performance over multiple time periods.

\subsubsection*{4.6.2.1 Scoring Individual Predictions}

Policy evaluation for a given operation period is a two-class classification problem where every possible event in the ABAC privilege space falls into one of two possible classes: grant or deny. By applying the policies generated from the observation period data to the
privileges exercised in the operation period, 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.

\[
TPR = \frac{TP}{TP + FN} \quad (4.2)
\]
\[
FPR = \frac{FP}{FP + TN} \quad (4.3)
\]

Using the above outcomes we then calculate **True Positive Rate (TPR)** also known as **Recall** and **False Positive Rate (FPR)** as shown in Equations 4.2 and 4.3, respectively.

As with the problem definition, these metrics are also derived from [48] but adapted from RBAC to be more suitable to the ABAC privilege space. Where [48] used metrics based on TPR and Precision, we used TPR and FPR instead. Precision \( \frac{TP}{(TP + FP)} \) is suitable when considering the users and operations because the universe of possible grants is roughly on the same order of magnitude as the number of unique log events. When dealing with the ABAC universe, the number of possible unique attribute:value combinations is likely to be many orders of magnitude greater than the number of events in the operational logs. To avoid over-fitting, ABAC rules must grant a large number of attribute:value privileges in absolute terms (on the order of hundreds or thousands of attribute:value combinations in our experiments), but are actually still quite small relative to the universe of possible attribute
combinations (which totals in the millions or billions). Stated another way, Precision is not a suitable metric for use in mining ABAC policies from logs because it uses one term (TP) which is driven primarily by the number of entries in the log, and another term (FP) which is driven by the size of the privilege universe. On the other hand, both terms in the TPR (TP and FN) are log derived, and both terms in FPR (FP and TN) are policy derived metrics.

TPR and FPR are the metrics used to evaluate a policy in terms of under-privilege and over-privilege, respectively. If all privileges exercised in the OPP were granted, there was no under-privilege for the policy being evaluated so $FN = 0$, and $TPR = 1$. As the number of erroneously denied privileges (FNs) grows, $TPR \to 0$, thus TPR represents under-privilege. For the edge case that no privileges were exercised in the OPP we redefine TPR to be $TPR = 1$, as no under-privilege is possible in this case. If all privileges granted by the policy were exercised during the OPP, there was no over-privilege for the policy being evaluated so $FP = 0$ and $FPR = 0$. As the number of erroneously granted privileges (FPs) grows, $FPR \to 1$, thus FPR represents over-privilege.

### 4.6.2.2 Scoring Policies Across Multiple Time Periods

To score policies across multiple time periods, we use out-of-time validation [32], a temporal form out-of-sample validation. In out-of-sample validation, a set of data is used to train an algorithm (training set) and a separate set of non-overlapping data is used to test the performance of the trained algorithm (test set). In our evaluation, the training and test sets are contiguous and the test time period immediately follows the training time period. The training set is referred to as the Observation Period (OBP), while the test set is the Operation Period (OPP) defined previously in Section 4.6.1. It is important to note that this method preserves the temporal interdependencies between actions. For example, if an employee moves to a new position within the organization, one would expect the privileges mined for that employee in the future time periods would be very different from those mined in the past time periods. Methods such as k-fold cross validation which randomly partition a dataset (and used in [25] to evaluate policies) do not account for these temporal inter-
dependencies. When charting metrics for multiple time periods, we use the average of all individual scores. This gives equal weight to each operation period score.

4.6.2.3 Scoring Infinite Possible Resource Identifiers

Quantifying the number of resources allowed or denied by a policy implies that there is a known value for the number of possible resources in the system. This presents a challenge for any least-privilege scoring approach that is not unique to the ABAC model or our methodology. While every system has finite limits on the resource identifier length and number of resources, these can be so numerous that we can consider them as too large to quantify and treat them as being infinite. For example, consider how many possible file names there are for the ext4 file system with up to 255 bytes for the file name, $2^{255}$ possible distinct file names exist, excluding the file path [58].

Instead of counting all possible resource identifiers, we use the resource identifiers present in the OBP and OPP for our policy scoring calculations. This approach presents several advantages over other possible approaches such as using all values in the dataset, or introspecting the environment for the resources present (which would be prohibitively time consuming for our dataset). Only the recently used resources are counted, giving them greater importance, and all necessary data is available in the audit logs. This also implies that the valid privilege space $\xi^i$ may vary in size between scoring periods depending on the resource identifiers present.

4.7 Methodology

This section presents both the algorithm we used to generate policies for addressing the $PEMP_{ABAC}$ problem as well as the algorithm we used to score these policies across multiple operation periods.
4.7.1 Rule Mining
4.7.1.1 Scoring Candidate Rules

Our rule mining algorithm operates similarly to the mining algorithms presented in [25, 54] in that it considers the set of uncovered log entries and iteratively generates many candidate rules, scores them, and selects the best scoring rule for the next iteration until all of the given log events are covered by the set of generated rules. A critical component of this approach is the metric used to evaluate candidate rules. Before describing the algorithm design, we will first detail the metric used for evaluating candidate rules generated during the mining process. We propose a candidate scoring metric termed the $C_{\text{score}}$ in this paper using the following definitions.

- $c$ is an ABAC constraint specified as a attribute:value pair, or single key and a set of values key:{values}. Values are required to be discrete, continuous values must be binned to be used by the mining algorithm. $r$ is a rule consisting of one or more constraints. $p$ is a policy consisting of one or more rules.
- $\mathbb{L}$ is the complete set of log entries for the dataset, $\mathbb{L}_{\text{OBP}}$ is the set of logs in the observation period $\text{OBP}$, $\mathbb{L}_{\text{OBP}} \subseteq \mathbb{L}$.
- $\mathbb{L}_{\text{OBP}}(c)$ is the set of log entries which meet (are "covered by") the set of constraints $c$. The constraint set may be specified by the use of a rule $r$ or policy $p$, $\mathbb{L}_{\text{OBP}}(c) \subseteq \mathbb{L}_{\text{OBP}}$.
- $\xi'$ is the privilege universe of valid log events as defined previously in Section 4.6.1.

The CoverageRate (Equation 4.4) is the ratio of all logs in the observation period covered by a candidate rule $r$ that are not already covered by other rules in the policy $p$ ($|\mathbb{L}_{\text{OBP}}(r) \setminus \mathbb{L}_{\text{OBP}}(p)|$) to the remaining number of log entries not covered by any rules in the policy ($|\mathbb{L}_{\text{OBP}} \setminus \mathbb{L}_{\text{OBP}}(p)|$). A candidate rule that covers more log entries is considered higher quality than a rule that covers fewer log entries. The numerator of the OverPrivilegeRate (Equation 4.5) first finds the number of valid attribute:value combinations in the privilege
universe which are covered by a rule \((\xi'(r))\), minus those attribute:value combinations which occur in the set of uncovered logs \(L_{OBP}(r) \setminus L_{OBP}(p)\), the result is the total number of over-assignments for rule \(r\). The total over-assignments are then normalized using the total number of valid combinations in the valid privilege universe \(|\xi'|\). A candidate rule which has fewer over-assignments is considered higher quality than a rule that has more over-assignments. The candidate score \(C_{score}\) (Equation 4.6) is then the \(\omega\) weighted addition of the CoverageRate and the complement of the OverPrivilegeRate. By normalizing the under-assignments using the number of log entries and the over-assignments using the size of the valid privilege universe, the effect of varying the weight \(\omega\) in the \(C_{score}\) is more predictable and results in better performance when compared to the \(\lambda\)-Distance metric which also uses a variable weighting between over-assignments and under-assignments but does not normalize these values (see Section 4.8.2 for \(C_{score}\) vs. \(\lambda\)-Distance comparison details).

\[
CoverageRate(r, p, L_{OBP}) = \frac{|L_{OBP}(r) \setminus L_{OBP}(p)|}{|L_{OBP} \setminus L_{OBP}(p)|}
\]  
(4.4)

\[
OverPrivilegeRate(r, p, L_{OBP}, \xi') = \frac{|\xi'(r) \setminus (L_{OBP}(r) \setminus L_{OBP}(p))|}{|\xi'|}
\]  
(4.5)

\[
C_{score}(r, p, L_{OBP}, \xi', \omega) = CoverageRate(r, p, L_{OBP}) + \omega \times (1 - OverPrivilegeRate(r, p, L_{OBP}, \xi'))
\]  
(4.6)

4.7.1.2 Rule Mining Algorithm

Our algorithm for mining an ABAC policy from the logs of a given observation period is presented in Algorithm 4. Note that we use arithmetic operators =, +, - when describing integer operations, and set operators \(\leftarrow, \cup, \setminus, \in\) when describing set operations. As mentioned previously, the algorithm iteratively generates candidate rules from the set of uncovered logs. To avoid confusion between the original set of log entries for the observation period \(L_{OBP}\) and the current set of uncovered log entries which is updated for each iteration of the algorithm, we copy \(L_{OBP}\) to \(L_{uncov}\) at line 2. The \(FP\)-growth algorithm [46] is used to mine frequent itemsets from the set of uncovered observation period log entries (line 4).
Algorithm 4: Rule Mining Algorithm

**Input:** \( L_{OBP} \) The set of log entries representing user actions during the observation period \( OBP \).

**Input:** \( \omega \) under-privilege vs. over-privilege weighting variable.

**Input:** \( \epsilon \) Threshold value for minimum itemset frequency.

**Input:** \( \xi \) The set of all valid attribute:value combinations that comprise the privilege universe.

**Output:** \( policy \) The set of ABAC rules that make up the policy to be applied during the operation period \( OPP \).

1. \( policy \leftarrow \emptyset \);
2. \( L_{uncov} \leftarrow L_{OBP} \);
3. \textbf{while} \( |L_{uncov}| > 0 \) \textbf{do}
   4. \( itemsets \leftarrow FP-growth.frequentItemsets(L_{uncov}, \epsilon) \);
   5. \( candidateRules \leftarrow \emptyset \);
   6. \textbf{for} \( itemset \in itemsets \) \textbf{do}
      7. \( rule \leftarrow createRule(itemset) \);
      8. \( coverageRate = \frac{|L_{uncov}(rule)|}{|L_{uncov}|} \);
      9. \( overAssignmentRate = \frac{|\xi(rule)|-|L_{uncov}(rule)|}{|\xi|} \);
      10. \( rule.C_{score} = coverageRate + \omega \times (1 - overAssignmentRate) \);
      11. \( candidateRules \leftarrow candidateRules \cup rule \);
   12. \textbf{end}
   13. \( bestRule \leftarrow sortDescending(candidateRules, C_{score})[0] \);
   14. \( policy \leftarrow policy \cup bestRule \);
   15. \( L_{uncov} \leftarrow L_{uncov} \setminus L_{uncov}(bestRule) \);
16. \textbf{end}
17. \textbf{return} \( policy \)
The itemsets returned by the \textit{FP-growth} algorithm are sets of \textit{attribute:value} statements, and each of these itemsets is used to create a candidate rule which is then scored using the $C_{score}$ metric (lines 6-12). After all candidates are scored, the highest scoring rule is selected and added to the policy; then all log entries covered by that rule are removed from the set of uncovered log entries (lines 13-15). The mining process continues until all log entries are covered (lines 3-16).

\subsection*{4.7.2 Policy Scoring}

Once the observation period logs have been mined to create a policy, that policy is scored using the events that took place during the operation period immediately following the mined observation period as described in Algorithm 5. Each event during the operation period is evaluated against the mined policy (lines 3-10), events allowed by the policy are TPs, while events denied by the policy are FNs. A unique combination of \textit{attribute:value} pairs may occur multiple times within the same time period. The TPs and FNs are both values based on the number of times an event occurs in the log. The set of unique events that were exercised in the operation period and granted by the policy is also maintained (line 6) in order to calculate the FPs later (line 15). By counting each TP and FN instead of unique occurrences, the resulting TPR is frequency weighted. Events that occur more frequently in the operation period have a greater impact on the resulting TPR than those events that occur less frequently.

While the TPs, FNs, and resulting TPR are based on the frequency weighted count of events present in the log, the FPs, TNs and resulting FPR cannot be frequency weighted because each unique valid event of the privilege universe is either granted or denied only once by the policy. To obtain these values (FP, TN, FPR), we first determine how many unique events out of the valid privilege space are granted by the policy (lines 11-14). It is important to note that enumerating the entire privilege space and testing every valid event against the policy would be much more computationally intensive than our approach, which is to use information about the valid privilege space to enumerate only the valid events allowed by
Algorithm 5: Policy Scoring Algorithm

Input: $\mathbb{L}_{OPP}$ The set of log entries representing user actions during the operation period $OPP$.
Input: $\xi$ The set of all valid attribute:value combinations that comprise the privilege universe.
Input: $policy$ The set of ABAC rules that make up the policy to be applied during the operation period $OPP$.
Output: $TPR, FPR$ The true positive and false positive rates of the $policy$ evaluated against the operation period $OPP$.

1 $TP = FN = 0$
2 $exercisedGrantedEvents \leftarrow \emptyset$
3 for $event \in \mathbb{L}_{OPP}$ do
4    if $policyAllowsEvent(policy, event)$ then
5        $TP = TP + 1$
6        $exercisedGrantedEvents \leftarrow exercisedGrantedEvents \cup event$
7    else
8        $FN = FN + 1$
9    end
10 end
11 $eventsAllowedByPolicy \leftarrow \emptyset$
12 for $r \in policy$ do
13    $eventsAllowedByPolicy \leftarrow eventsAllowedByPolicy \cup \xi(rule)$
14 end
15 $FP = |eventsAllowedByPolicy \setminus exercisedGrantedEvents|$
16 $TN = |privUniverse| - (TP + FN + FP)$
17 if $TP + FN == 0$ then
18    $TPR = 1$
19 else
20    $TPR = TP/(TP + FN)$
21 end
22 $FPR = FP/(FP + TN)$
23 return $TPR, FPR$
each rule. Most mined rules only allow a small percentage of the privilege space except in cases of extreme $\omega$ values.

Once the set of all the unique events allowed by a policy has been enumerated, we remove the set of unique events which occurred and were granted during the operation period to obtain the number of total FP events for the policy (line 15). At this point we have obtained the unique sets of TPs, FNs, and FPs, so any remaining privilege in the valid privilege universe not in these sets must be a TN (line 16). With these values calculated, we can then calculate the TPR and FPR, with the caveat that in the case where no privileges were exercised during the operation period, we define $TPR = 1$ because there could not be any instances of under-privilege (lines 18-22). The purpose of the $policyAllowsEvent()$ function is self-explanatory and trivial to implement, so the implementation of this method is omitted due to space considerations.

4.7.3 Optimizations For Large Privilege Spaces

Dealing with the large number of possible $attributes:value$ combinations that may comprise an ABAC privilege space can be a significant challenge compared to the simpler RBAC privilege space. Using all attributes and values present from logs may make the privilege universe computationally impractical to process. But discarding too many attributes or important attributes may result in less secure policies. We address these issues by using feature selection and partitioning methods to make large ABAC privilege spaces more manageable.

4.7.3.1 Preprocessing And Feature Selection

Intuitively, attributes which occur infrequently in the logs or have highly unique values are poor candidates for use in creating ABAC policies. Uncommon attributes are difficult to mine meaningful patterns from because there is less data available to identify patterns from. Also, rules created with uncommon attributes are less useful in access control decisions because future access requests are unlikely to use these attributes as well. Using attributes with unique values (the attribute value is never or rarely duplicated across log entries) is likely
to result in over-fitting for any rules created with those attributes. Following this reasoning, we perform preprocessing on our dataset to select and bin the most useful attributes as follows.

1. Remove unique and redundant attributes using Uniqueness where \( Uniqueness = \frac{\text{UniqueValues}}{\text{AttributeOccurrences}} \).

2. Remove redundant correlated attributes.

3. Sort attributes by Frequency = \( \frac{\text{AttributeOccurrences}}{\text{TotalLogEntries}} \). Select attributes above a frequency threshold, \( \theta \).

4. Sort the remaining values by Uniqueness, high Uniqueness are candidates for binning or removal.

Our full dataset contained 1,748 distinct attributes (see Section 4.8.1 for dataset description). In step (1) attributes with Uniqueness \( \approx 1.0 \) nearly always have unique values, and Uniqueness \( \approx 0.0 \) implies the attribute values are nearly always the same. Resource identifiers are given an exception to the uniqueness test in this step as they are expected to have high uniqueness. For our dataset, we identified and removed two always unique attributes, eventID and requestID, and one attribute that always had the same value accountId. We confirmed that these attributes would always meet the uniqueness criteria with the AWS documentation. Applying step (2), we identified three distinct attributes for the user name with a 1:1 correlation and removed two of them. For step (3) we selected two thresholds to build two datasets for experimentation, \( \theta = 0.1 \) and \( \theta = 0.005 \), we term the privilege universes built using these thresholds \( \xi_{0.1} \) and \( \xi_{0.005} \), respectively. Figure 4.1 charts the rank of the top 50 most common attributes after our feature selection process was complete. The attribute frequency follows the common power law distribution with a “long tail”; the remaining attributes not charted here occurred in less than 0.2% of the log entries.

Next we apply step (4) of our process to our dataset. Some of the remaining attributes still have fairly high Uniqueness values which are difficult to mine meaningful rules from. In our
Figure 4.1: Top 50 Attributes Ranked by Frequency

dataset, some of these attributes such as checksum values are not relevant to creating security policies and can be discarded. Others are attributes which may benefit from binning into a smaller subset of values. There were three such attributes in our dataset: sourceIPAddress, userAgent, and eventName. The sourceIPAddress is an IPv4 address with over 4 billion possible values. After consulting with the system administrator of the dataset provider, we found that it was unlikely they would use rules based on the raw IP address since users will change IPs frequently. Instead, they preferred to derive the geographical location from the IP address so IPs were binned by U.S. states and each country the organization’s users may log in from. The userAgent attribute is the AWS Command Line Interface (CLI), Software Development Kit (SDK), or web browser version used when making a request. This field benefits from binning as users are likely to perform similar requests from a web browser, but they may upgrade their browser version regularly. Without binning the many different browser versions into a single group, a mining algorithm would not effectively learn user
patterns. Again, the dataset provider agreed that the raw value was too granular for use so the `userAgent` attribute was binned into 10 buckets. The `eventName` attribute is the name of the operation. This attribute is already effectively binned because each `eventName` can only be associated with one `eventSource` which is the AWS service name associated with the operation. We derived two additional attributes to bin `eventName`, one based on whether it was a Create, Read, Update, Delete, or Execute operation, and a second derived attribute based on the first word of the `eventName`. For example the operation “StartInstance” is binned into a bucket with other attributes that begin with “Start”. Experiments showed this improved TPR with a negligible decrease in FPR at a ratio of 20:1.

4.7.3.2 Mining Algorithm Optimizations

The resulting ABAC privilege space may still be quite large even for a modest dataset after applying the feature selection and binning methods as just described in Section 4.7.3.1. This section describes partitioning techniques we applied to split up the privilege space during the policy mining process. Partitioning techniques (as used in databases to split large tables into smaller parts) are used to both reduce the memory footprint of our algorithms, and to improve performance by performing operations in parallel across multiple processors.

The rule mining algorithm (Algorithm 4) uses partitioning to improve the run time and space efficiency for storing and searching the privilege universe $\xi'$. The total number of valid combinations of $\xi'$ was on the order of billions for some of our experiments, but Algorithm 4 only needs to determine the number of privileges covered by a rule and it does not need to enumerate and store all possible privilege combinations in memory. This is a subtle but important difference because it means we can calculate the number of valid privilege combinations by splitting $\xi'$ into smaller sets of independent partitions to perform this calculation. The total number of valid privilege combinations covered by a rule is the product of the number of valid privilege combinations covered by each separate partition, i.e., $|\xi'(r)| = |P_1(r)| \times \ldots \times |P_n(r)|$ where the attributes of each partition $P_i$ are independent of the attributes in all other partitions.
To create these partitions, the AWS documentation was used to identify dependencies between attributes in our dataset. Next, a simple depth first search was used to identify connected components of interdependent attributes. The valid attribute: value combinations for all attributes in each connected component were then enumerated and stored into one inverted index for each partition. Finding the number of valid privilege combinations covered by a rule in a partition \(|P_n(r)|\) is accomplished by searching the inverted index using the rule’s attribute: value constraints as search terms. As a result of this partitioning, our queries were performed against three indexes on the order of thousands to hundreds of thousands of documents vs. a single index that would have been on the order of hundreds of millions to billions of documents if such a partitioning scheme were not in use.

For our dataset, a depth first search identified one connected component of all user attributes, and another connected component of operations and resources. Operations and resources were connected because most operations are specific to a single or set of resource types. We grouped all other attributes that were independent of users and operations into a third component which included environment attributes such as the sourceIPAddress and userAgent. Although this grouping of attributes by components was obtained from processing our specific dataset, it is reasonable to assume that the user attributes are independent of the valid operation and resource attribute combinations in other datasets as well. This is also consistent with the NIST ABAC guide which defines environment conditions as being independent of subjects and objects [42].

Due to the large number of candidate rules generated by the \(FP-growth\) algorithm, scoring of candidate rules is the most computationally intensive part of Algorithm 4 in our experiments (except for those with fairly large \(\epsilon\) values which generate few candidates). The search against the inverted index is also parallelized to improve performance.

### 4.7.3.3 Scoring Algorithm Optimizations

To improve the run time performance of the policy scoring algorithm (Algorithm 5) and enable it to deal with a privilege space larger than the available memory, we again
employ partitioning and parallelization methods. As mentioned in 4.7.2, Algorithm 5 must enumerate the set of all privilege combinations covered by a rule in order to identify the total unique number of privilege combinations covered by a policy. If extreme values for \( \omega \) are chosen, it is possible for Algorithm 4 to generate rules with a large number of over-privileges, possibly the entire privilege space. Therefore, Algorithm 5 must be able to deal with the possibility that it will have to enumerate all privilege combinations of \( \xi^i \), although again, this only happens for extreme values of \( \omega \), and this is for the out-of-sample validation for policy scoring only, not the rule mining algorithm.

To deal with the possible need to enumerate a large portion or even all of the privilege space, we partitioned \( \xi^i \) along two attributes so that the values of those attributes are placed into separate partitions. As with any partitioning, choosing a key that nearly equally splits the universe of possible values is important. For our experiments, we chose to partition the \( \xi^i \) space along the attributes associated with the operation name and the user name. The overall correctness of the algorithm is independent of the partition keys used, and 1...n partitions may be used for each attribute depending on the size of the privilege space and available memory.

Each of these partitions is operated on in parallel when evaluating each rule of the policy. Unique hashes of the enumerated events are used in order to deduplicate events which may be generated by more than one rule. This partitioning and parallelization takes place within lines 11-14 of Algorithm 5. We describe these optimizations here because they are useful in speeding up and scaling the algorithm when dealing with a large number of \textit{attribute:value} pairs, but we omit it from the pseudo-code in Algorithm 5 in order to simplify the presentation of the parts of the algorithm necessary for correctness.

### 4.8 Results

We use the Receiver Operating Characteristic (ROC) curve to compare the performance of various algorithms and parameters. The ROC curve is a graphic commonly used to chart the performance of binary classifiers. It charts the trade-off between the TPR and the FPR.
of a binary classifier, with the ideal performance having a TPR value of one and FPR value of zero. Our charts also include the Area Under the Curve (AUC) which measures the area underneath the ROC curve. This provides a single quantitative score that incorporates both the FPR and TPR as the weighting metric is varied with higher AUC scores being more favorable.

First, we describe our dataset used for these experiments. Next we present experimental results and analysis to justify our choices for the candidate evaluation metric $C_{score}$, including a comparison of several possible methods for normalizing the CoverageRate variable. Then we examine the effect of varying the two adjustable input variables to the mining algorithm, the length of the observation period ($|L_{OBP}|$), and the minimum support threshold ($\epsilon$). Finally, we compare the performance of our ABAC algorithm and policies to that of an RBAC based approach.

### 4.8.1 Dataset Description

We examine the performance of our ABAC policy generation algorithm on a real-world dataset. Our dataset was provided by a Software As A Service (SaaS) company and consists of 4.7M user generated AWS CloudTrail audit events representing 16 months of audit data for 38 users. We used user-generated audit events only, filtering out those events generated by non-person entities. Events generated by non-person entities were very consistent, and it is easy to derive very low under- and over-privilege security policies for them directly from audit logs without using sophisticated methods. Addressing the problem of minimizing the privilege assignment errors for human users is much more challenging so we chose to focus on human generated log events only.

To better understand the privilege space used in our dataset and highlight the high degree of variability in user behavior, statistics about user behavior are shown in Table 4.1. This table separates the metrics by the first month, last month, and total for 16 months 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 the number of actions that users exercise.

Table 4.1: 16 Month Total Usage of our Dataset

<table>
<thead>
<tr>
<th>Metric</th>
<th>First Month</th>
<th>Last Month</th>
<th>16 Months</th>
</tr>
</thead>
<tbody>
<tr>
<td>Users</td>
<td>17</td>
<td>26</td>
<td>38</td>
</tr>
<tr>
<td>Unique Services Avg.</td>
<td>12.94</td>
<td>12.11</td>
<td>22.66</td>
</tr>
<tr>
<td>Unique Services StdDev.</td>
<td>10.16</td>
<td>9.98</td>
<td>16.70</td>
</tr>
<tr>
<td>Unique Actions Avg.</td>
<td>65.76</td>
<td>62.92</td>
<td>168.34</td>
</tr>
<tr>
<td>Unique Actions StdDev.</td>
<td>76.11</td>
<td>73.30</td>
<td>178.52</td>
</tr>
<tr>
<td>( \sum ) Actions Avg.</td>
<td>9138.35</td>
<td>9664.19</td>
<td>123659.82</td>
</tr>
<tr>
<td>( \sum ) Actions StdDev.</td>
<td>20279.87</td>
<td>14124.15</td>
<td>235915.45</td>
</tr>
</tbody>
</table>

Based on our dataset of 4.7M user generated events, we derive two privilege universes using our feature selection methodology described in Section 4.7.3.1. \( \xi_{0.1} \) used 15 attributes and consisted of 510M unique \( attribute:value \) combinations. \( \xi_{0.005} \) used 40 attributes, 25 of which were resource identifiers so the universe size varied between 1.5B and 8.6B unique \( attribute:value \) combinations depending on the number of resources used during the \( OBP \) and \( OPP \) periods. All of the experiments in this section use \( \xi_{0.1} \) except for Section 4.8.4 which uses \( \xi_{0.005} \).

4.8.2 \( C_{score} \) Analysis

We consider three criteria in the design and evaluation of the \( C_{score} \) metric for selecting a single rule from many candidate rules generated by the \( FP-growth \) algorithm during each iteration of our rule mining algorithm. \( C1:AUC \) is the Area Under the ROC Curve.
C2: Smoothness means that TPR values should increase monotonically as the FPR increases. And, C3: Interpretability means that the effect of changing the weighting variable should be predictable and easy to understand by an administrator who uses the metric in a policy mining algorithm.

4.8.2.1 Evaluating Candidate Scoring Metrics

We propose the candidate scoring metric $C_{score}$ in Section 4.7.1.1, $\lambda$–Distance is presented in [25], and $Qru{l}$ is presented in [54]. All of these metrics use the number of over-assignments and number of log entries covered with a weighting variable for adjusting the importance between over-assignments and coverage in their scoring of candidates. However, these metrics differ in how they normalize these numbers (if at all) and how they implement the weighting between them. The results of varying the over-assignment weightings for these candidate evaluation methods are shown in Figure 4.2.

Four distinct versions of the $Qru{l}$ metric are presented in Figure 4.2. $Qru{l}$ is the metric as presented in [54] (and in this paper as Equation 4.1). In [54], the authors also described $Qru{l}Freq$, a frequency weighted variant of $Qru{l}$ which should be a fairer comparison with our frequency weighted policy scoring algorithm (Algorithm 5). The authors of [54] provide their source code on their website. After inspecting this source code, it appears that the scoring algorithms implemented in the source code for $Qru{l}$ and $Qru{l}Freq$ are slightly different from those presented in the paper. Instead of using the number of privileges covered by a rule out of the entire privilege universe ($[p]$) as the denominator for the over-assignments side of the metric, the implemented metrics instead use the number of privileges covered by a rule out of the log entries not covered by other rules already in the policy ($|[p] \cap UP|$). These “as-implemented” metrics, $Qru{l}Impl$ and $Qru{l}FreqImpl$, perform more favorably than their counterparts so we include them in our comparison here along with the versions as documented in [54].

All of the examined metrics performed relatively well with high AUC values, but the $C_{score}$ metric has the highest AUC value, thus being the most favorable metric per the
Figure 4.2: Comparison of Candidate Evaluation Metrics

criterion \(C1: AUC\). While we do not provide a quantitative score for \(C2: Smoothness\), it is evident from Figure 4.2 that the \(C_{score}\) is much closer to a monotonic function than the other metrics whose \(TPR\) values increase and decrease several times as the \(FPR\) increases. The \(Qrul\) and \(QrulFreq\) methods are particularly poor in terms of smoothness as they both have an inflection point near \(\omega' = 1\), where increasing the weighting slightly after that point causes the term \(\frac{w_0' \times |p| \cup P(L)}{|p|}\) of Equation 4.1 to be > 1. The \(Qrul\)-based metrics then take the complement of this value, causing this side of the equation to produce negative values. Furthermore, increasing the weighting beyond a certain point causes the metric to only select those candidate rules which have zero over-assignments, resulting in the unterminated
portion of the ROC curve for Qrul (QrulFreq has a similar inflection point that is difficult to discern in Figure 4.2 at FPR = 0.0013).

Unlike the Qrul and λ–Distance metrics, Cscore normalizes both the number of logs covered and over-assignments into a ratio between [0, 1] before applying the weighting. This makes the weighting variable independent of the size of the privilege universe and number of log entries and thus easier to understand and apply. In Figure 4.2, varying the ω weighting of the Cscore results in the FPR values between ω = \(\frac{1}{10}\) and ω = 10, and varies the charted FPR between FPR = 0.05 and FPR = 0.998 at relatively even intervals. To achieve a similar spread across the FPR scores with QrulFreqImpl and λ–Distance, the variable weighting for those metrics must be varied between \(\frac{1}{100}\) and \(\frac{1}{2000}\). QrulImpl achieved the second highest AUC score due to an unusually good score near FPR = 0.34, but QrulImpl is difficult to assign a weighting to with predictable results. For example, the QrulImpl score at FPR = 0.34, TPR = 0.9998 was achieved with \(\omega'_{0} = \frac{1}{100000}\), but the next score at FPR = 0.49, TPR = 0.9988 was achieved with \(\omega'_{0} = \frac{1}{500000}\), which is a significant difference that is difficult to determine without experimentation and consideration of the privilege space and log sizes. Because of its predictability and more even distribution of results, we find Cscore best meets our evaluation criterion C3:Interpretability.

4.8.2.2 Methods of Calculating CoverageRate

The CoverageRate (Equation 4.4) of the Cscore (Equation 4.6) is the number of log entries covered by rule \(r\) normalized to the range [0, 1], so that it can be compared with the weighted value of the OverPrivilegeRate (Equation 4.5) normalized to the same range. There are several possible ways to compute such a coverage rate however, and it is not immediately clear which would perform the best without experimentation. We consider four possible methods of computing the CoverageRate and analyze their performance here:

- \(\frac{|\text{uncov}(r)|}{|\text{uncov}|}\): The frequency weighted number of logs covered out of the total number of uncovered logs.
• $\frac{|L_{uncov}(r)|}{|L_{uncov}|}$: The unique number of logs covered out of the set of unique uncovered logs.

• $\frac{|L_{uncov}(r)|}{|L_{OBP}|}$: The frequency weighted number of logs covered out of the total number of logs in the observation period.

• $\frac{|\{L_{uncov}(r)\}|}{|\{L_{OBP}\}|}$: The unique number of logs covered out of the set of unique log entries during the observation period.

The results of applying the four separate methods of computing the CoverageRate are presented in Figure 4.3 and identified in that chart by the denominator of each method. As evident in Figure 4.3, the $\frac{|L_{uncov}(r)|}{|L_{uncov}|}$ method performed the best for two of our criteria for selecting a candidate metric: $C1: AUC$ and $C2: Smoothness$. The frequency weighted methods $\frac{|L_{uncov}(r)|}{|L_{uncov}|}$ and $\frac{|L_{uncov}(r)|}{|L_{OBP}|}$ performed about the same in terms of $C3: Interpretability$ with $\omega = \frac{1}{10}$ resulting in scores in the upper-left most part of the chart. The methods using the number of unique log entries performed less favorably in terms of $C3: Interpretability$ with their upper-left most points being reached near $\omega = \frac{1}{256}$, a value farther away from 1 and more difficult to find without experimentation.

4.8.3 Effect of Varying Algorithm Parameters

In addition to the $\omega$ variable which is varied to generate the points along all of the ROC curves in this section (with the exception of the RBAC algorithm curve in Figure 4.6), there are two other parameters which can be varied as inputs to Algorithm 4: the threshold value used by the $FP-growth$ algorithm, $\epsilon$, and the length of the observation period $|L_{OBP}|$.

4.8.3.1 Effect of Varying Itemset Frequency Threshold

The minimum support threshold ($\epsilon$) is used to specify that a pattern is considered a “frequent” pattern if that pattern occurs in $\geq \epsilon$ of the examined entries. Increasing $\epsilon$ causes fewer candidate patterns to be identified by the $FP-growth$ algorithm. The results of varying $\epsilon$ between $[0.05, 0.1, 0.2, 0.3]$ are shown in Figure 4.4. For both $\epsilon = 0.2$ and $\epsilon = 0.3$, we observe inflection points in the chart as $\omega$ decreases because a lower $\omega$ value favors more
granular rules in order to lower the over-privilege rate; however, higher $\epsilon$ values result in fewer and less granular patterns being identified by the $\text{FP}−\text{growth}$ algorithm. Stated another way, low $\omega$ values generally result in lower $\text{FPR}$ values, while high $\epsilon$ values generally result in higher $\text{FPR}$ values. The inflection points occur as a result of conflicting instructions between low $\omega$ and high $\epsilon$ values.

As a result of generating more candidates for the mining algorithm to evaluate, lower $\epsilon$ values generally result in higher $\text{AUC}$ scores as well. The trade-off for more candidate policies however is an increase in run time. At $\omega = \frac{1}{10}$, the average mining times for $\epsilon = 0.05, 0.1, 0.2, 0.3$ were 29.8, 15.3, 2.8, and 1.2 minutes, respectively. The other charts in
Figure 4.4: Performance as Itemset Frequency Varies

this section were generated using $\epsilon = 0.1$ as it offered the best trade-off between performance, stability, and run time.

4.8.3.2 Effect of Varying Observation Period Length

When mining policies with a variable observation period length, a larger observation window generally results in higher $TPR$ but also higher $FPR$ as a result of the mining algorithms being given more privileges in larger observation periods as previously observed in [48]. While this trend is also present with our mining algorithm, it is much less noticeable than with the naive RBAC mining approach.
The results of varying the observation period length between $|L_{OBP}| = [7, 15, 30, 45, 60]$ days are shown in Figure 4.5. As $|L_{OBP}|$ increases, the TPR generally increases compared to lower $|L_{OBP}|$ periods of similar FPR values, and the resulting ROC curve becomes smoother. As with $\epsilon$, we observe a trade-off between $|L_{OBP}|$ and run time. At $\omega = \frac{1}{16}$, the average mining times for $|L_{OBP}| = [7, 15, 30, 45, 60]$ were 5.7, 6.5, 7.2, 10.5 and 12.8 minutes, respectively. The other charts in this section were generated using $|L_{OBP}| = 30$ days as it offered the best trade-off between performance, stability, and run time.
The final experiment we run is to compare the performance of our ABAC algorithm against an RBAC mining algorithm. For this comparison, we use the naive algorithm presented in [48], which builds an RBAC policy based on the permissions exercised during an observation period. Other role mining algorithms would perform very similarly because the role mining problem is designed to fit a set of roles to a given matrix of user to permission assignments, just with variations on how those users and permissions are grouped by roles to minimize WSC. Although this RBAC algorithm is fairly simple, it performed quite well in
the scenario that sought an equal balance between low over-privilege and low under-privilege when compared to more sophisticated algorithms [48].

The ROC curve of our ABAC algorithm and the naive RBAC algorithm from [48] are presented in Figure 4.6. For this comparison, the ABAC algorithm used a fixed observation period size of 30 days, an itemset frequency \( \epsilon = 0.1 \), and the over-privilege weight varied between \( \omega = \left[ \frac{1}{8192}, \ldots, 16 \right] \) by powers of 2 to generate the data points. For the RBAC algorithm, there is no variable similar to \( \omega \) that can be used as a parameter to instruct the algorithm to directly vary the importance between under-privilege and over-privilege. However, varying the observation period length effectively serves this purpose by causing more or fewer privileges to be granted by the algorithm, so the observation period length was varied between \([3, 7, 15, 30, 45, 60, 75, 90, 105, 120]\) days to generate the data points for the RBAC algorithm in Figure 4.6.

The ABAC algorithm significantly outperformed the RBAC algorithm across the ROC curves in Figure 4.6. With only 30 days worth of data, the ABAC algorithm was able to correctly grant more privileges (higher TPR) than the RBAC algorithm with 120 days of data. The ABAC algorithm was also able to correctly restrict more unnecessary privileges (lower FPR) than the RBAC algorithm operating on only 3 days of data. This is due to the ability of the ABAC algorithm to identify and use patterns and create policies based on attributes vs. the RBAC algorithm which is restricted to using only RBAC semantics.

4.9 Summary

This paper explored methods for automatically generating least privilege ABAC policies that balance between minimizing under- and over-privilege assignment errors. We defined the ABAC Privilege Error Minimization Problem \((ABAC_{PEMP})\). We also presented metrics and methodology for evaluating ABAC policies using out-of-sample validation. We adapted techniques from unsupervised rule mining to create an algorithm which automatically performs ABAC policy generation by mining audit logs with a variable weighting between under- and over-privilege. We described optimization methods using feature selection, partitioning,
and parallelization to mine and score large ABAC privilege spaces. Finally, we presented the results of applying our algorithm on a real-world dataset which demonstrated its effectiveness as well as the better performance of our ABAC policies over mined RBAC policies.

This work suggests many possibilities for future research in generating secure ABAC policies. Our candidate rule scoring metric, $C_{score}$, can be expanded to consider policy complexity (WSC), or our method can be combined with those which minimize policy complexity only. Additional attributes may be incorporated from sources other than just audit logs such as HR databases of user attributes, or by introspecting the application environment and extracting attribute information about existing resources. As the number of attributes grows, so does the importance of feature selection for selecting highly relevant attributes that can help improve the security of the generated policies without greatly increasing the runtime and memory required by a mining algorithm.
CHAPTER 5
CONCLUSION

As access controls have evolved to cover the complex and various use cases of modern computing, the burden of defining access control policies has also increased, often exceeding the human ability to define policies that implement the Principle of Least Privilege. Along with increasing complexity the commoditization of computing power, such as cloud computing, has made it easier than ever for organizations to rapidly deploy computing resources with minimal effort (or training), thus increasing the risks and damages that may be caused as a result of poor access control policies. The research presented in this thesis demonstrates the effectiveness of several automated methods for creating access control policies that achieve the principle of least privilege with quantitatively evaluations of their performance at reducing under-privilege and over-privilege on real world datasets. More specifically, the individual projects that comprise this thesis have made the following contributions to advance the state of access control research:

1. We explored the challenges and benefits of implementing an automated least privileges approach for third party web services using real world data. We also presented a concrete implementation of a framework for generating least privilege policies from audit log data, and presented metrics and methodology for quantifying the effectiveness of least privilege policies in an RBAC environment.

2. We formally defined the Privilege Error Minimization Problem (PEMP) which described the problem of creating complete and secure RBAC privilege policies. Using our previously defined metrics and policy generation framework we presented a methodology for training and validating one naive and two machine learning based algorithms. Again using real world data, we present evaluation results for our presented algorithms.
3. We presented an association rule mining based algorithm to address the problem of automatically creating ABAC policies. We also presented feature selection, scalability, and performance optimization methods for processing the large privilege spaces that are inherent to the ABAC environment. Using metrics adapted from our previous work to better suit ABAC policies, we presented a quantitative analysis of the performance of our mining algorithm using a real-world dataset and a comparison of our automatically generated ABAC policies created by our mining algorithm with automatically generated RBAC based policies.
REFERENCES CITED


