Explanation-Based Auditing

by

Daniel Fabbri

A dissertation submitted in partial fulfillment
of the requirements for the degree of
Doctor of Philosophy
(Computer Science and Engineering)
in The University of Michigan
2013

Doctoral Committee:

Assistant Professor Kristen LeFevre, Chair
Assistant Professor Michael J. Cafarella
Professor Margaret Hedstrom
Assistant Professor David A. Hanauer
Professor H.V. Jagadish

© Daniel Fabbri 2013
All Rights Reserved

To My Mother.

ACKNOWLEDGEMENTS

The PhD process is a long and arduous journey with many ups and downs. This type of journey cannot be completed alone. I would like to thank my advisor, Kristen LeFevre, for her assistance. She taught me many skills that I will carry for my lifetime. I learned how to analyze and break down complex problems, search for their basic principles and express my solutions clearly. While this process drained many red pens, her comments and insights kept me going and provided direction when I was lost. I attribute a large proportion of my current opportunities to her tutelage. So, Kristen — thank you.

The database research group provided the tight-knit community I required to explore. Their feedback immensely improved my initial ideas and forced me to reconsider my assumptions and think deeply. I am thankful for their mentoring and support: paper reviews, late-night office company and life guidance. I may have quit without some of them. Because of the graduate mentors before me, I made it a priority to try to mentor the younger students in the group. I hope I was as helpful as my predecessors.

The faculty and staff in the Computer Science and Engineering Department were exceptionally helpful and supportive. Steve, Lauri, Bebe, Dawn and many others made my job easier and for that I am thankful.

I would like to also thank the many friends and colleagues who supported me over the years. Our time together provided the necessary breaks from graduate work and allowed me to re-energize. In particular, I have been lucky enough to play on sports teams with two great groups of friends for many years. Our camaraderie and post-game outings provide lasting memories. Additionally, I have lived with fantastic housemates. They were my Ann Arbor family. Thank you for dealing with me when I was down and pushing me forward when I needed it. The friends I have made here will last me the rest of my life, no matter where in the world we may be.

Finally, I would like to thank my family. My father always provided the guidance and support I needed. He taught me to make decisions for myself and manage challenging situations. My sister inspires me to keep pushing myself and live life with energy. My grandfather, through his simple enjoyment of life, reminds me to appreciate the moments we have, even during stressful times. Lastly, I would like to thank my mother. Even though she is no longer with us, she taught me many things. Most importantly, she instilled within me the ideas that I am capable, and do not always need be perfect to be successful. While she may have joked that she wanted me to be the ‘correct’ type of doctor, I am sure she would be proud of this accomplishment.

This work was supported by National Science Foundation grants CNS-0915782 and IGERT-0903629.

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

ABSTRACT

Explanation-Based Auditing

by
Daniel Fabbri

Chair: Kristen LeFevre

Recent U.S. legislation such as the Affordable Care Act, HIPAA and HITECH outline rules governing the appropriate use of personal health information (PHI). Unfortunately, current technologies do not meet the security requirements of these regulations. In particular, while electronic medical records (EMR) systems maintain detailed audit logs that record each access to PHI, the logs contain too many accesses for compliance officers to practically monitor, putting PHI at risk. This thesis presents the explanation-based auditing system, which aims to filter appropriate accesses from the audit log so compliance officers can focus their efforts on suspicious behavior. The main observation of the system is that most appropriate accesses to medical records occur for valid clinical or operational reasons in the process of treating a patient, while inappropriate accesses do not. This thesis discusses how explanations for accesses (1) capture these clinical and operational reasons, (2) can be mined directly from the EMR database, (3) can be enhanced by filling-in frequently missing types of data, and (4) can drastically reduce the auditing burden.

CHAPTER I

Introduction

1.1 Overview

Database systems increasingly manage volumes of valuable and sensitive information. These systems allow employees to query, view and retrieve data from a single location quickly and efficiently. In certain environments, employees are free to access and analyze all data stored in the database. However, in other environments, there are restrictions on who can access what data. Ideally, the database system would deploy the necessary technology to restrict access when necessary. Unfortunately, modern technologies cannot always secure the sensitive data while still permitting normal operations. As a result, limited security systems are deployed, putting the sensitive data at risk of misuse.

One environment that requires this type of access restriction is the medical domain. In recent years, there has been a migration from paper medical records into their digital form. Electronic medical records (EMR) systems have the potential to drastically improve patient care by aggregating clinical information, improving caregiver communication and reducing medical errors. However, because of their increased accessibility, the security and appropriate use of the personal health information (PHI) contained in medical records is at greater risk. For example, as reported in the news, various celebrities and dignitaries from Britney Spears [62] to Congresswoman Gabrielle Giffords [41] have had their PHI breached. While these inappropriate accesses were detected and publicized in the news, many likely go unnoticed.

What is interesting about these breaches is that often these employees are not malicious but are rather simply curious. As stated by a healthcare employee, "'It's pretty damn common' for medical professionals to peek at files for unwarranted reasons" [21]. Moreover, after analyzing many breaches, one compliance officer noted, "Most of the time, the motivation for the snooping is curiosity or concern about a coworker, family member or neighbor" [57]. However, simply looking at a patient's medical record out of curiosity is not legally allowed.

Recent U.S. legislation such as the Affordable Care Act, the Health Insurance Portability and Accountability Act (HIPAA) and the Health Information Technology for Economic and Clinical Health (HITECH) Act outline rules governing the appropriate use of personal health information. These laws provide guidance on how PHI can be used, requirements for monitoring accesses to PHI, and penalties and fines for breaches. Unfortunately, current technologies fail to meet the security requirements of the regulations.

One possible way to secure electronic medical records is with fine-grained access controls. Fine-grained access controls limit which patients' medical records an employee can view. The problem with this technology is that these access control policies must be specified proactively. However, the medical environment is dynamic, and it is difficult to predict who will treat a patient. As a result, fine-grained access controls cannot meet the data needs of employees. Moreover, in emergency situations, it is irresponsible to prevent access if needed. As a result, EMR systems typically do not deploy fine-grained access controls [32]. Instead, once a hospital employee is authenticated by the EMR system, the employee can access any patient's medical record, which makes PHI susceptible to curious behavior.

A slightly more relaxed security mechanism still deploys access controls, but allows employees to escalate their permissions (or "break-the-glass" [43]) to access restricted patient data. Before an employee accesses the data, a large warning is displayed, and if they choose to continue, the escalated accesses are specially logged and monitored. Unfortunately, studies have show this type of escalation can be overused. Specifically, because of the dynamics of patient care, employees escalated their permissions to access approximately 50% of the patients' medical records in the study, which makes an escalated access the norm [69].

Instead of these proactive security mechanisms, hospitals maintain audit logs that record all accesses to PHI. Compliance officers can retrospectively review accesses in the log to detect breaches. Ideally, this type of security approach would act as a deterrent to inappropriate behavior because employees are aware that their accesses are logged and analyzed. (In fact, hospitals are required by law to maintain audit logs, analyze accesses for inappropriate use and report breaches [23].)

Unfortunately, analyzing the audit log to detect inappropriate accesses presents many challenges. First, most accesses are appropriate. Therefore, finding the few inappropriate accesses is similar to finding a needle in a haystack. Second, medical records systems from large institutions record millions of accesses a week. Therefore it is infeasible for compliance officers to manually review all accesses in the limited time that they have budgeted. In practice, compliance officers monitor accesses to VIPs or patients who have registered a complaint, but no additional monitoring is provided to the general public. Third, it is not immediately apparent how to determine if an access is inappropriate from the information stored in the log, which makes automating the auditing process difficult.

In summary, current technologies do not provide adequate accountability and security for EMR systems because of the complexities of the work environment, where there are no fine-grained access controls, and the raw number of accesses that occur, which does not scale with manual analyses.

1.2 Objective

Observing the challenges of the medical work environment and the limitations of current security technologies, the objective of this thesis is to *provide compliance officers the ability to quickly and accurately find inappropriate accesses from the audit log.*

In addition to this main objective, this thesis also addresses a secondary objective. Recent Health and Human Services proposed rules allow patients to request access reports detailing all accesses to their medical records. One concern about these reports is that they will create more problems than benefits because patients will not know many of the employees listed (e.g., the nurses or pharmacists). Thus, the secondary objective of the thesis is to *develop security systems that provide interpretable results that both compliance officers and patients can comprehend.* If this objective is met, the concerns about the access reports may be mitigated (if the rules are put in place).

1.3 Approach

This thesis presents the explanation-based auditing system (EBAS) as a means to quickly and efficiently find inappropriate accesses from audit logs, while providing interpretable results. The remainder of this thesis will discuss the basic observations

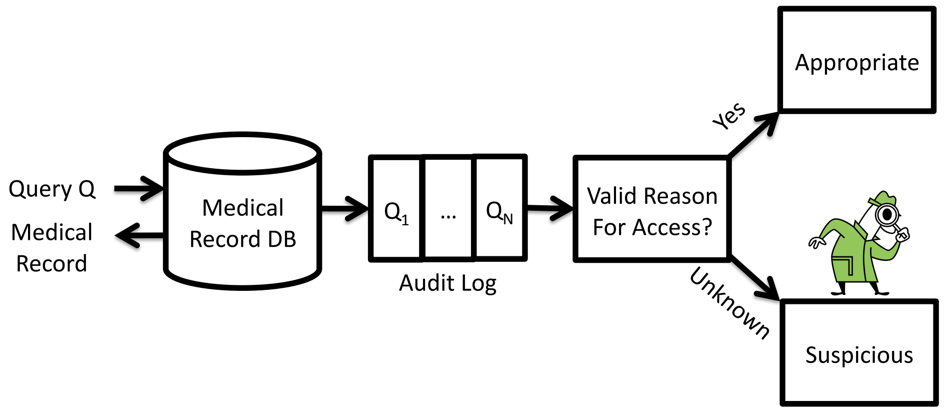


Figure 1.1: The explanation-based auditing system attempts to find a valid clinical or operational reason for each access in the audit log. Accesses without a reason are suspicious and manually reviewed by the compliance officer.

and assumptions of the systems, the mechanisms used and evaluation results using real hospital data from the University of Michigan Health System.

The main insight of this thesis is that *most appropriate accesses to medical records occur for valid clinical or operational reasons in the process of treating a patient, while inappropriate accesses do not.* Moreover, if a valid reason for access can be determined, the access is most likely appropriate and does not need to be reviewed by the compliance officer. This insight is exemplified by the University of Michigan Health System's screen saver: "Authorized access is limited to those with the need to know for purposes of patient care, billing, medical record review, or quality assurance" [77].

As Figure 1.1 shows, for each audit log entry, the explanation-based auditing system attempts to find the clinical or operation reason for access. If a valid reason is found, the access is classified as appropriate and does not need to be analyzed further. If no reason is found, the access is marked as suspicious and reviewed by the compliance officer. Intuitively, the EBAS serves as a filter on the audit log so that compliance officer only must review a subset of the log.

The challenge for the EBAS is to detect valid reasons for access that are also interpretable. There are several classes of reasons that logically explain why accesses occur. The main observation of this research is that *EMR databases store information describing the process by which a patient is treated, which can be used to explain why accesses occur.* For example, the EMR database stores appointment information, medication orders and radiological findings, all of which can be used as evidence to infer the reason for an access based on who participated in the appointment, administered the medication or reviewed the X-ray. The auditing system can use the EMR database to construct valid reasons for access that can then be used to filter appropriate accesses from the audit log.

This thesis explores various issues related to determining the reasons for access (or explanation) and filtering appropriate accesses from the audit log.

- **Defining Explanations:** The first goal of this thesis is to define an explanation that captures the reason for access. Conceptually, an explanation is logically represented as a path connecting the data that are accessed (i.e., the patient's medical record) to the employee accessing the data, possibly using data stored in the EMR database. The thesis discusses various examples and demonstrates how this model captures the desired semantics of appropriate use (Chapter V).
- **Mining Explanations:** Before explanations can be used, they must be specified. In the naive case, the compliance officer would specify all explanations. However, this manual process is burdensome for large databases. Therefore, this thesis presents algorithms to automatically mine explanations from the database (Chapter V). The hypothesis is that the same clinical and operational reasons for access are common across many employees and patients, and therefore can be discovered by analyzing frequently occurring connections between employ-

ees and patients in the database. Variations of a basic mining algorithm are presented that improve system performance.

- **Dealing With Missing Data:** Unfortunately, databases are not perfectly curated and are often missing data. For example, patients make appointments with doctors, not nurses. Therefore, while a doctor's access to the patient's medical record can be explained using the appointment information, no connection exists between the nurse and the patient. To improve the auditing system, Chapter VI presents two algorithms to fill-in missing types of data such as (i) employee collaborative working relationships and (ii) information describing the medical diagnoses each department is responsible for treating. With this added information, the auditing system is able to mine new explanations and filter more appropriate accesses from the audit log.
- **Ordering Accesses By Suspiciousness:** The explanation-base auditing system is able to filter over 94% of the accesses from a University of Michigan Health System audit log. Unfortunately, the number of unexplained accesses is still too large for compliance officers to manually review. Interestingly, most unexplained accesses originate from general departments in the hospital such as Central Staffing Nurses, Pharmacy and Radiology. To best utilize the compliance officers' time, this thesis presents techniques to order the remaining unexplained accesses by suspiciousness, so that compliance officers can review the most suspicious accesses first (Chapter VII). The observation here is that these general departments' accesses are temporally related to events occurring around the hospital. For example, a pharmacist's access is temporally dependent on when a medication order is placed. Using this observation, the EBAS can order accesses by the likelihood that a department will access a patient's

record at a given point in time.

- **Developing A Prototype Auditing System**: To close-the-loop and verify that the explanation-based auditing system is accurate and improves compliance officer efficiency, Chapter VIII presents the design and development of a prototype auditing system. The prototype allows compliance officers to extract data from a commercial EMR system, load the data into the prototype, create and mine explanations, and search for inappropriate behavior. The University of Michigan Health System's Compliance Office will evaluate the effectiveness of the prototype with known instances of inappropriate behavior.

- **Auditing Ad-Hoc SQL Query Logs:** Typically, employees access EMR databases through restrictive user interfaces that restrict access to a single patient's medical record at a time. In these cases, the audit log records the employee who is accessing the database, the patient's medical record that is accessed and the time. However, in certain situations, employees query patient information using ad-hoc SQL queries. For example, a doctor may pose the question: *How many pediatric patients had asthma over the last year grouped by month?* In this latter case, the database constructs an audit log that stores the raw SQL query. These SQL audit logs present some challenges for the EBAS because they do not explicitly record which employees accessed which patients' records. This thesis presents algorithms to audit SQL logs to determine which medical records each employee accesses (Chapter IX).

The explanation-based auditing system was evaluated using a de-identified temporal snapshot of the University of Michigan Health System's EMR database (Chapter III). The snapshot was acquired over one year after numerous meetings with Medical Center Information Technology (MCIT) staff members. The data set contains an audit log with over four million accesses by 12K hospital employees (inpatient and outpatient) to 120K patients. In addition to the audit log, information describing the process by which a patient was treated was also acquired (e.g., appointment, medication and diagnosis information). The auditing system was evaluated using this temporal snapshot. While the specific results presented are derived from the University of Michigan Health System, the approaches are generally applicable to other health care institutions.

The explanation-based auditing system makes two simplifying assumptions (Figure 1.2). First, it addresses the threat posed by a curious employee who only accesses PHI; it does not address the potential threat posed by a malicious employee who modifies database information or the threat posed by a malicious outsider who compromises the authentication system. Second, it assumes that all appropriate accesses occur for a valid clinical or operational reason to treat a patient. However, some appropriate accesses occur for alternative reasons such as retrospective research. As a result, the EBAS can produce false positives and negatives. These assumptions strike a reasonable balance between addressing a real and meaningful threat to PHI and addressing the practical aspects of the problem, but if these assumptions do not hold, the system reverts back to the case where compliance officers still must manually analyze all accesses.

It is important to note that while the techniques presented in this thesis are developed and evaluated on medical data, the approaches are applicable to other environments. Specifically, the methods presented are applicable for domains where accesses occur for operational reasons, and information describing these operational reasons are stored in the database. For instance, the explanation-based auditing system could be used to audit accesses by passport officials who inappropriately

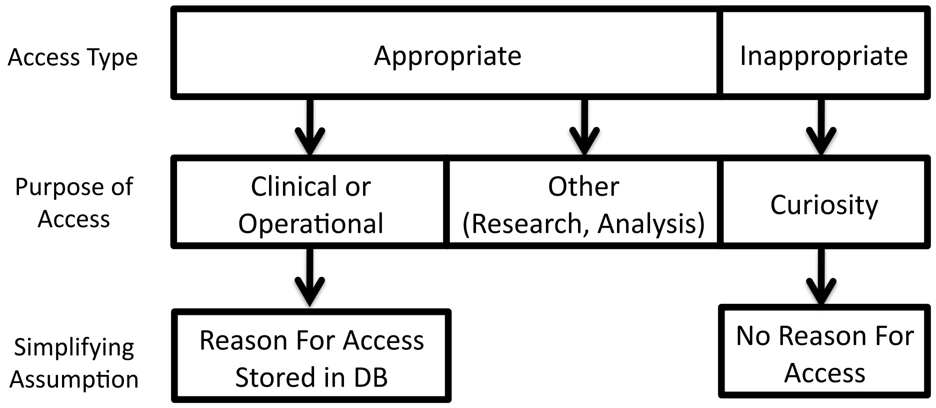


Figure 1.2: Problem Taxonomy: This thesis considers the problem of explaining the clinical or operational reason for appropriate accesses in the audit log using data stored in the EMR database. Inappropriate accesses that occur out of curiosity are not explained, but reviewed by the compliance officer. The auditing system produces a false positive if no reason can be found for an appropriate access, while a false negative occurs if a reason for access is mistakenly found for a curious access.

review the travel history of specific individuals (e.g., Obama's passport file [46]). The database information describing which passport officials are assigned to review each individual's travel history could be used to mine and explain accesses.

In summary, this thesis presents the explanation-based auditing system, a tool for compliance officers to quickly and accurately find inappropriate (curious) accesses from audit logs by filtering appropriate accesses that occur for clinical or operational reasons. These reasons for access are represented as paths connecting the employee accessing the patient's medical record to the patient whose record is accessed. Moreover, these connections are constructed using information stored in the EMR database that describes how the patient was treated. Because EMR databases are often missing important information such as doctor and nurse working relationships, this thesis also presents techniques to fill-in missing data to improve the quality of the auditing system. For compliance officers to efficiently process the remaining unexplained accesses, the auditing system orders these accesses by their suspiciousness so the most suspicious accesses are reviewed first. The quality of the auditing system is measured using data from the University of Michigan Health System.

This thesis describes the main components of the explanation-based auditing system, but many other extensions are possible in the future. In particular, this thesis utilized de-identified data for its analysis. However, if patient privacy can be ensured, there are opportunities to examine if clinical notes can be used to fill-in additional types of missing information such as patient referrals. Additionally, as the number of accesses to EMR systems increase, the auditing system must scale out with its usage. Future work is needed to analyze how distributed computation architectures can be leveraged to parallelize the auditing system.

CHAPTER II

Background

This chapter provides an overview of past work in data auditing and security.

2.1 Access Controls

One common mechanism used to secure sensitive data stored in a database are access controls. In the simplest case, administrators can configure access control policies such that each user (or employee) can only access specific database tables. While these types of broadly-specified access control polices are useful in many environments, they are often too coarse for tables that aggregate large amounts of diverse data. Instead, fine-grained access controls (or row-level access controls) can be configured to restrict users to a subset of a table. Databases enforce these access control policies by rewriting queries as they are executed (e.g., [60, 68]). For example, a pediatrician's query for all patients' medical records could be rewritten to return only pediatrics records by appending the condition $age < 18$ onto the query's selection condition. Unfortunately, deploying access controls for many users in large databases is difficult. As a result, misconfigurations in the access control policies can occur, allowing users to access data inappropriately.

Instead of specifying access control policies for each employee, role-based access controls restrict access to employees based on their roles and responsibilities [73].

Role-based access controls have many practical benefits for the medical domain; for example, secretaries can be restricted to patient contact information, while physicians can access contact information and clinical notes. However, these roles are often too general, allowing employees that are not involved in a patient's care to access medical information. Team-based access controls provide administrators greater flexibility to specify access control policies than role-based access controls, which is useful in collaborative environments like the medical domain where many employees from different roles treat patients together [36, 76]. However, team-based approaches are limited because it is often difficult to determine the treatment team *a priori*. Moreover, employees are constantly rotating between services, making the task of updating these teams extremely challenging and error prone.

To better manage dynamic collaborative work environments, context-aware access controls extend previous approaches by granting and adapting permissions based on the current context [82]. For example, if the access control system knows a patient has an outstanding medication order, then the pharmacist can access the patient's medical record. Similarly, the context can also include relationships between the data that are accessed and the user accessing the data [10]. For example, Salazar-Kish et al. use pre-existing relationships between the user and patient (that are already stored in the database) to allow access for a patient's primary care physician, scheduled provider and referring provider [72]. Interestingly, the authors explore how long (temporally) after an appointment a user should be able to access a patient's record. The results show that the relationship provides minimal additional benefits after four months have passed. Additionally, Russelo et al. propose an access control framework for e-Health that adapts access rights to the actual tasks employees have to fulfill [71].

The challenge with context-based systems is identifying what information should be used to make an access control decision. Moreover, in some cases, the information needed for the access control decision may be missing from the database at the time of access (e.g., the medication order is ordered verbally at first, but electronically entered later). As a result, legitimate accesses may be prevented. Chapter V adapts some of these ideas (i.e., (i) finding relationships between the data and the user accessing the data, and (ii) using the database context) to model and mine *explanations* for access.

Deploying fine-grained access controls in the medical domain is challenging because of the dynamics of patient care and the associated difficulty of predicting who will access a patient's medical record. As a result, any specified access control policy is likely to prevent access when necessary. Even worse, access controls could unnecessarily restrict access in an emergency, potentially resulting in patient harm. To provide exceptions to access control policies, previous work evaluated if employees should be able to escalate their permissions (or "break-the-glass" [43]) to access restricted data. In these cases, the EMR database warns the employee about his or her access, and if the employee chooses to continue, the access is specially logged and monitored by the compliance office. Unfortunately, in the study, employees escalated permissions for approximately 50% of the patients' medical records [69], making an escalation the norm rather than an exception. As a result of these challenges (i.e., many escalations and the difficultly of predicting who will access what data), fine-grained access controls are irregularly deployed in EMR databases [32]. Instead, once employees log in and are authenticated, they can access any patient's medical record.

The explanation-based auditing system is designed for retrospective auditing rather than proactive access prevention. However, the techniques presented in this thesis could be leveraged to enhance fine-grained access control systems. In particular, the explanation-based auditing system can be modeled as a type of context-based access control system, where an access is allowed if the system can determine the clinical or operational reason for access. Moreover, the access control decision can be improved by filling-in missing types of data (e.g., diagnosis responsibility information) as discussed in Chapter VI. One limitation of this application is that the information needed to determine the reason for access may not be available (i.e., stored in the database) at the time of access.

2.2 Audit Logs

To comply with data compliance requirements, most commercial databases have the ability to maintain audit logs, which record a history of accesses to the database [47, 60, 79]. Each audit log record typically stores information about who accessed the data, when it was accessed and what data was accessed. The audit log is stored in a separate, secure, and tamper-proof database. A compliance officer can review the audit log to detect inappropriate accesses.

Generally, audit logs are designed for specific levels of abstraction. *Application-level* audit logs store information detailing the exact data objects accessed by a user (e.g., Dr. Dave accessed Alice's record). With this type of log, it is trivial to answer the question: *Who accessed the sensitive data?* In contrast, a recent body of work has focused on the related problem of SQL log auditing. In this case, logs are collected at the level of the Database Management System (DBMS), recording the text of all SQL queries and updates; most commercial DBMSs now support this form of logging [47, 60, 79]. In contrast to application-level auditing, it is non-trivial to determine which logged queries accessed particular portions of the database from these raw SQL audit logs (or database-level audit logs), and used these records in non-trivial ways.

2.3 Auditing SQL Audit Logs

Various models and systems have been proposed to address the problem of auditing SQL audit logs [6, 30, 31, 45, 59]. Here, the goal of an auditing system is to determine which users accessed sensitive data (sensitive data are pre-specified by the compliance officer). The challenge is then to determine what it means for a query to “access” sensitive data.

Broadly, there are two approaches used to determine if a SQL query accessed sensitive data. An *instance independent* auditing approach marks a query as *suspicious* of accessing the sensitive data if there exists *some* database instance in which the query’s result changes when sensitive data are removed from the database instance [59]. Alternatively, an *instance dependent* auditing approach marks a query as suspicious if the query’s result changes when sensitive data are removed from the *current* database instance [6]. The current database instance refers to the database state that existed at the time when the query was originally executed. As a result, databases that use an instance dependent auditing system need to be able to reconstruct past database states (e.g., temporal databases [44] or “point-in-time” recover APIs). Ideally, the auditing system would be able to analyze the SQL audit log and construct an application-level audit log that specifies exactly the data each user accessed.

In some ways, these auditing semantics are similar to those described in provenance research. At a high-level, provenance work attempts to answer questions like: *How was this row produced, what information contributed to this query result*, or *why was this row not in a result* [13, 16, 19]? To answer these questions, additional flags or bits are often added to the data to track the propagation of information. While useful, this line of work is not immediately applicable to the auditing problem.

2.4 Inappropriate Access Detection

Anomaly detection systems have been studied extensively in the past to detect inappropriate access [15]. One line of research considers the user (or employee) as the unit of suspiciousness and attempts to determine whether or not a user is behaving in an unexpected way. For EMR systems, Chen et al. study the extent to which user access patterns deviate from their peers to detect inappropriate use [17, 18]. In contrast, this thesis considers individual accesses as the unit of suspiciousness, and tries to explain why each access occurs. This approach is more suitable if, for example, hospital employees are generally well behaved, but in some isolated cases inappropriately access information (e.g., the Britney Spears case [62]).

Recent work has also considered finding individual inappropriate accesses, and therefore are the closest related works to this thesis. Boxwala et al. extract patient-employee relationships (e.g., the user is the patient's provider), patient characteristics (e.g., the patient's address and name), and employee characteristics (e.g., the employee's address and care unit) that are used as features in classifiers to detect suspicious accesses [12]. Experiments demonstrate good prediction quality (sensitivity > 0.75). Interestingly, the output probabilities from these classifiers can be used to order accesses for review, so compliance officers can analyze the most suspicious accesses first. This ordering approach is similar to the the work presented in Chapter VII that orders accesses using temporal relationships between hospital events.

Similarly, other approaches use patient and employee characteristics to detect inappropriate access. Asaro et al. analyze patient characteristics (e.g., the patient is a V.I.P. or recently had a drug test), patient-employee interactions and access session characteristics (e.g., large numbers of accesses at a time) to create a taxonomy of indicators designating when a breach may occur [8]. While these related works and this thesis focuses on data available within the EMR database, other work considers aggregating external data sources to improve auditing [42].

There are obvious similarities between these related works and the work presented in this thesis. Specifically, they leverage patient and employee information already stored in the database to detect inappropriate access. One challenge of deploying these related systems is that compliance officers must manually engineer features to be used as input to the classifier, which can be time consuming. To reduce the burden for compliance officers, Chapter V presents methods to mine frequently occurring reasons for access from the database so that compliance officers only need to approve those mined explanations instead of generating them by hand. The standardized structure of an explanation template (i.e., a path on the schema graph connecting the patient and user) allows the system to easily incorporate new types of data and construct interpretable access reports. For example, Chapter VI describes techniques to fill-in frequent types of missing data that can be easily used to produce new explanations.

Other auditing systems have attempted to understand and mine the workflows of organizations to detect inappropriate access [78]. The experience-based access management system attempts to understand medical record usage under a probabilistic framework to construct relational access control policies to better predict user data needs [37, 51, 56]. Using these relationships, for example, the system can determine the probability with which a patient's medical record will be accessed by a surgeon after an emergency room physician accesses it. Similarly, Li et al. characterize workflows on a per-object basis using hidden Markov models, and if an extracted workflow deviates from the expected, the access is marked as suspicious [52].

Anomaly detection systems have been studied in a variety of other domains besides electronic medical records access. For example, researchers have developed techniques for detecting anomalous traffic to web servers [48] and finding irregularities in process system calls [33]. Often these anomaly detection systems aim to find behavior that deviates from 'normal' activity to serve as evidence for malicious activity. Moreover, rather than requiring administrators to manually specify the input features for these anomaly detection systems, researchers have proposed techniques to learn anomaly detection parameters from the data [49]. In some sense, the approaches presented in Chapter V (i.e., mining explanations from data) are similar to this concept of mining anomaly detection parameters in that the explanation template can be thought of as a frequently occurring pattern contained within the data that signifies appropriate access. One major difference between these previous approaches and this work is that the explanation-based auditing system attempts to find appropriate rather than inappropriate access behavior. Moreover, these traditional anomaly detection approaches do not always produce easily interpretable results (e.g., a numeric value describing the difference from the norm, rather than a logical boolean statement), which can make it difficult to produce access reports.

Observing these differences, there appears to be two extreme approaches for building an inappropriate access detection system. One option is to explicitly search for inappropriate accesses [8, 12]. These systems are designed with the hypothesis that inappropriate accesses have distinguishable characteristics that make them stand out from all the other accesses in the audit log. Such a proactive anomaly system naturally fits in an online security infrastructure that attempts to warn users as the access occurs, which has some parallels with access controls. Alternatively, the system can first find all appropriate accesses and then mark the remaining accesses as suspicious [27, 37, 56]. In some cases, this approach is preferable if the appropriate reasons for access are easily enumerable. This retrospective security approach naturally fits in an offline security infrastructure that analyzes accesses after the fact.

2.5 Threats

There are many possible threats to sensitive data. This thesis considers the threat of a curious user who inappropriately accesses sensitive data without a valid clinical or operational reason. This threat represents many of the real instances of inappropriate behavior publicized in the press (e.g., [46, 62]) and 'snooping' employees are even referenced in Health and Human Services regulations [23]. Recent articles specifically focus on the risk of curious users [21]: "Even though these professionals are trained in this, for whatever reason – their curiosity or snoopiness, or sometimes something more malignant – prevails over their ethics and what they know they shouldn't do." Additionally, the research community has also looked at the threat of curious users and administrators in other environments (sometimes referred to as the 'honest-but-curious' adversary model) [58, 64].

There are other threat models that this thesis does not consider and leaves for future work. One possible threat is from malicious users who inappropriately access sensitive data and attempt to subvert the security system to prevent detection. Specifically, a malicious user could create fake appointments to cover-up inappropriate accesses. Related work on adversarial data mining addresses a variation of this problem where a malicious user misrepresents data such that a data mining algorithm produces incorrect results [22]. Another type of threat is from malicious outsiders who attempt to subvert the authentication system to gain admission to the database. While this threat has been publicized in the press, the authentication problem is outside the scope of this work. Instead, we assume a working authentication system that verifies that each user is who they say they are.

2.6 EMR Meta-Data

EMR databases store a wide range of meta-data that can be used for auditing. Hospital-designated departments serve as a natural grouping of employees based on shared roles and responsibilities. The approximately 300 departments at the University of Michigan Health System include Pediatrics, Nursing-Pediatrics, Pharmacy and Central Staffing Nurses, among others. It is important to note that employees that frequently work together (e.g., Pediatrics and Nursing-Pediatrics employees) are often assigned to different groups. Moreover, the task of assigning thousands of employees to departments is difficult. To automate this process, recent work has examined predicting an employee's department or role using EMR audit logs [83]. These roles have the potential to be used in a role-based access control system if highly accurate [73].

Additionally, EMR databases record ICD-9 diagnosis codes for billing. It is assumed that these codes are correctly entered. However, errors are possible. Recent work has proposed techniques to analyze medical record text to predict the patient's ICD-9 code in order to detect errors [81]. Moreover, patients are often diagnosed with many related conditions. Patnaik et al. mine temporal sequences of diagnoses to better understand their correlation and temporal relationships [63].

2.7 Broader Impact

This thesis presents the explanation-based auditing system as a tool to improve the accountability and security of electronic medical records. While the methods presented here are developed and evaluated on medical data in response to health care legislation [23], these techniques are applicable to other environments. For example, financial institutions and military departments store sensitive data that are protected with access controls. However, because of the difficultly in specifying fine-grained access control policies, administrators provide greater access to the data than necessary (often with coarse security designations such as ‘secret’ or ‘top secret’), so employees can complete their normal business operations. Observing the potential abuse of these access rights, various pieces of regulation from Sarbanes-Oxley [2] to the Department of Homeland Security’s *Handbook for Safeguarding Sensitive Personally Identifiable Information* [24] outline rules for when employees in financial organization or military positions can access sensitive data, respectively. The regulations often cite an employee’s ‘need to know’ when determining if an access is appropriate or not (i.e., access is required for the performance of official duties). However, like in the medical domain, it is difficult to determine if an access occurred for normal operational reasons or curiosity. Therefore, inappropriate accesses are difficult detect except in the case of large-scale abuse (e.g., the disclosure of government files to WikiLeaks [75]).

The explanation-based auditing system can serve as a complementary security tool for a wide range of organizations that retrospectively monitor accesses to sensitive data because their access control policies are too broadly specified. For example, financial institutions and government agencies can deploy the system to better ensure that employees only access the data they require for their job. The auditing system is applicable for those domains where accesses occur for operational reasons, and the reason describing why the access occurred is stored in the database as part of normal operations. Moreover, because databases are often missing information, some of the techniques presented in this thesis can be leverage to fill-in data to explain additional accesses.

CHAPTER III

Data Overview

3.1 Data

We conducted an extensive experimental study using data from the University of Michigan Health System. The data set contains a de-identified audit log consisting of 4.5M accesses during a week by 12K employees from 291 departments to 124K patients' medical records. The number of distinct user-patient pairs is approximately 500K, which gives a user-patient density of $\frac{|user-patient\ pairs|}{|users| \times |patients|} = 0.0003$. We were also given 291 descriptive codes describing which users worked in which departments such as *Pediatrics* and *Nursing-Pediatrics*. The accesses in the log can be divided into two groups: **first accesses** (i.e., accesses where an employee accesses a patient's medical record for the first time in the log), and **repeat accesses**.

The data set also contains evidence describing the reason for access such as diagnosis, appointment, visit, and medication information, among others listed in Table 3.1. It is important to note that our data set is a temporal snapshot of the EMR database from the weeks around the accesses in the audit log. As a result, information such as diagnoses and appointments outside this snapshot were not reported for our study. This directly impacts the number of diagnoses that we were provided (we have diagnoses for approximately 47% of the patients).

Log Characteristic	**Value**
# Employees	12K
# Departments	291
# Patients	124K
# Patients with Diagnoses	58K

Evidence	**# Records**
Audit Log Accesses	4.5M
Audit Log First Accesses	509K
Appointments	51K
Diagnosis Instances (ICD-9)	248K
Documents	76K
Inpatient Medications	122K
Labs	45K
Outpatient Medications	120K
Radiology Orders	17K
Services/Orders	106K
Visits	3K

Table 3.1: Data Overview

When we started the study, we initially requested the *Appointments*, *Visits*, and *Documents* tables (sometimes referred to as data set A), in addition to the log. However, after some preliminary analysis, we discovered that a large proportion of the unexplained accesses were by users who worked in departments that provide *consultation* services throughout the hospital (e.g., radiology, pathology, and pharmacy). Users in these departments often do not have appointments with patients. However, there is often an explicit request recorded in the database. Therefore, we expanded the study to also include the *Labs*, *Medications*, *Diagnosis* and *Radiology* tables (data set B), which maintain a record for each such event.

3.2 Basic Usage

In this section, we use the access log to provide a basic characterization of CareWeb usage patterns.

3.2.1 Log Characteristics

First, we examined the general properties of the access log. Figure 3.1 shows the number of accesses per hour over the week (December 13-19, 2010). Clearly, the

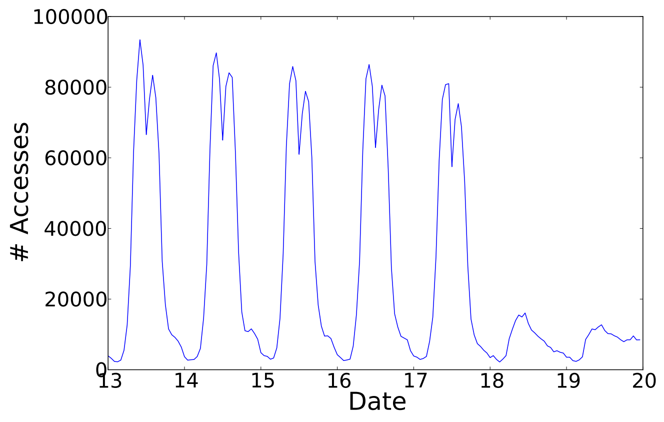


Figure 3.1: Accesses Per Hour (12/2010)

accesses follow a diurnal pattern, where the number of accesses is low in the early morning, increases through the day, drops during lunch, increases in the afternoon, and then declines again in the evening. Additionally, fewer accesses occur on the weekend (December 18-19) than during the week. Intuitively, the number of accesses appears to be proportional to the number of people working in the hospital system at a given point of time.

While the raw number of accesses can be useful, we are also interested in the types of actions that are executed. Along with each access, CareWeb records a description of the action that was performed. There are 124 distinct action types. Figure 3.2 shows the eight most frequent actions, which account for over 65% of all accesses. Unsurprisingly, the most common action is to view a list of a patient's documents, and the second most common action is to view a patient's document (e.g., doctor's note).

Finally, we examined the number of actions that read data vs. write data. Figure 3.3 shows that most accesses only read EHR data, while a smaller fraction modify the data.

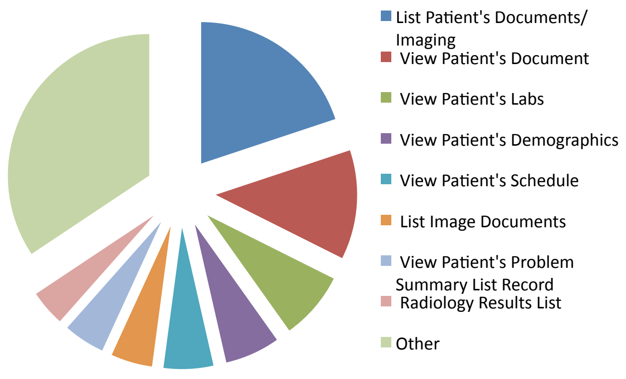


Figure 3.2: Actions

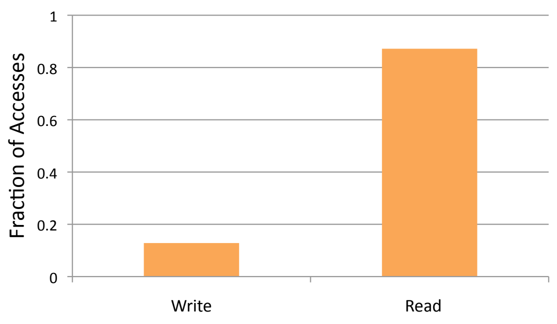
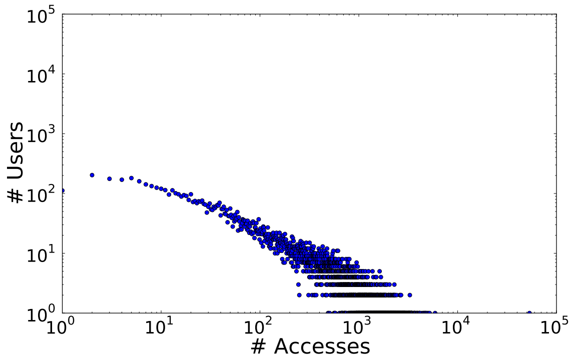


Figure 3.3: Read-Write Distribution



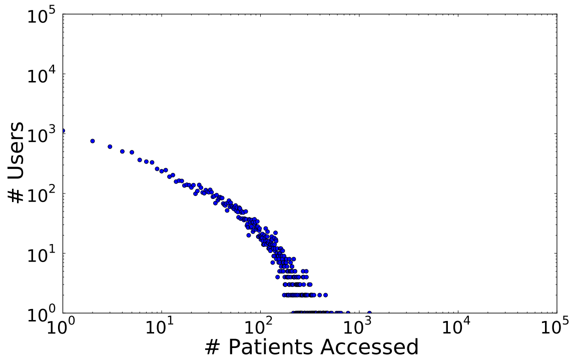


Figure 3.4: Accesses Per User

Figure 3.5: Patients Per User

3.2.2 User and Patient Characteristics

Due to the different types of jobs within the hospital system, EHR users access patient records in different ways. For instance, some users repeatedly access the same patient's record, while others access many different patients' records. Similarly, patients have their medical records accessed in different ways. For some patients, only their primary care physician accesses their medical record; other patients have large teams of doctors.

To better understand the types of users and patients in our data set, we analyzed the log of accesses based on four quantitative measures:

1. Total number of accesses per user (accesses/user)
2. Number of distinct patients accessed per user (patients/user)
3. Total number of accesses to a patient's medical record (accesses/patient)
4. Number of distinct users that access a patient's medical record (users/patient)

Figures 3.4, 3.5, 3.6 and 3.7 show the distribution of each measure using a log-log scale. We find that all four measures have heavily-skewed (”long-tail”) distributions. While the average user performed 358 accesses (39 distinct patients) during the week, most users accessed very few records. Similarly, the average patient's record

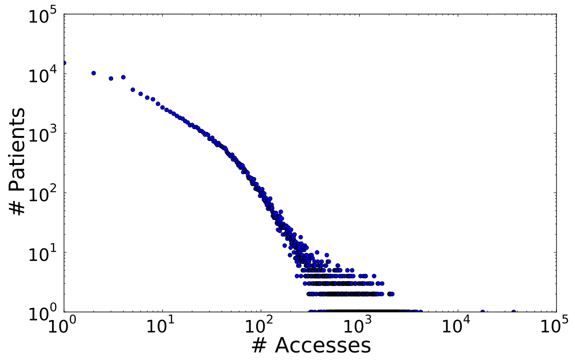


Figure 3.6: Accesses Per Patient

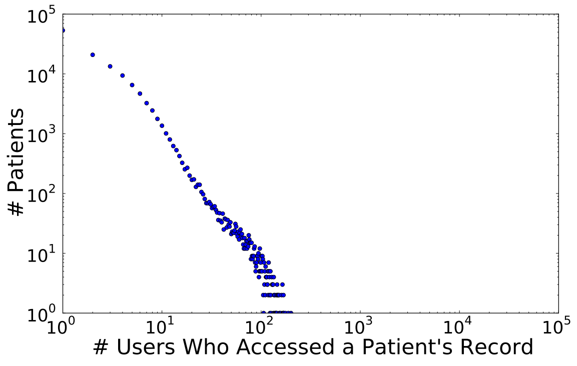


Figure 3.7: Users Per Patient

Accesses/User	Patients/User	Accesses/Patient	Users/Patient
UMHS Int Med - General Medicine (Physicians)	UMHS Int Med - General Medicine (Physicians)	Cancer Center	Radiology
UMHS Internal Medicine (Physicians)	Central Staffing Resources	UMHS Int Med - General Medicine (Physicians)	Central Staffing Resources
Cancer Center	Int Med - Outpatient Svcs - Taubman	UMHS Internal Medicine (Physicians)	Pathology
Health Information Management	Physician Services	Medical Students	Operating Rooms/PACU
UMHS Pediatrics - General Medicine (Physicians)	Radiology	Health Information Management	Cancer Center
Medical Students	Cancer Center	Central Staffing Resources	Physician Services
UMHS Family Medicine (Physicians)	Health Information Management	Radiology	Medical Students

Figure 3.8: Top departments by access behavior. For example, Internal Medicine physicians access the most medical records and the most unique patients.

was accessed 36 times (4 distinct users), but most patients' records were accessed very few times.

For both patients and users, there is a small subset of "heavy" users and patients. To further understand who these people are, we examined the users and patients in the tails of the distribution. For example, for accesses per user, we identified the top 10% of users (i.e., those with the most accesses). Among these users, the most frequent department codes are shown in Figure 3.8. Notice that internal medicine physicians perform the most accesses, and they access the most unique patients. In general, physicians tend to be among the heaviest users.

We also looked at the top 10% of patients (i.e., those with the most accesses to their records). Because we were working with de-identified data, we were not able to examine the patients' characteristics directly. Instead, we looked at the department codes of the users who accessed the patients' records, which gives us a good idea of

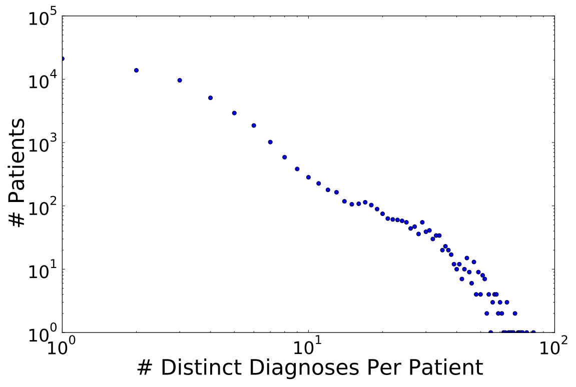


Figure 3.9: Diagnoses Per Patient: Patients on average have three diagnoses, but the distribution has a long tail.

where the patients were being treated. Figure 3.8 shows the common department codes of the users who treated this group of patients. Interestingly, and perhaps unsurprisingly, many of the patients were treated by users from the Cancer Center. When we look at the patients whose records were accessed by the most distinct users, we find that the users are from radiology, pathology and the operating room. These departments make sense for heavily accessed patients, since patients with x-rays, labs or who are in the operating room are typically treated by teams of hospital employees.

3.2.3 Diagnosis Characteristics

To provide more insight into the diagnosis information, Figure 3.9 shows that patients on average have three distinct diagnoses, but the distribution is heavily skewed ("long-tail"). When a patient has multiple diagnoses, the auditing system does not know which accesses correspond to each diagnosis. Figure 3.10 shows that the number of distinct accesses to a patient's medical record (i.e., distinct patient-employee pairs in the real audit log) is positively correlated with the number of distinct diagnoses for the patient (Pearson's correlation coefficient of 0.74).

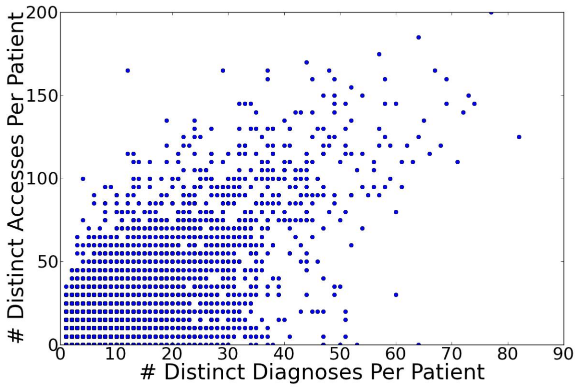


Figure 3.10: Access-Diagnosis Correlation: The number of distinct accesses for a patient is correlated with the number of distinct diagnoses for the patient.

CHAPTER IV

Experimental Methodology

The goal of the explanation-based auditing system is to determine if a given access occurred for a valid clinical or operational reason to treat a patient. Previous work (e.g., [17]) focused on finding individual employees who consistently displayed anomalous behavior across many accesses (e.g., downloading large amounts of patient data). However, in the medical domain, there is another threat: individual employees who are generally well-behaved, but who have occasional instances of curiosity that lead to inappropriate accesses. Thus, our work focuses on discovering individual inappropriate accesses, rather than finding users who consistently display anomalous behavior.

We first describe what information is available to the auditing system to determine if an access is appropriate. An access is stored in the audit log as an audit log entry that describes the employee who accessed the EMR system, the patient whose record was accessed, and the time of the access (i.e., Audit Log(employee, patient, timestamp)). The EMR database additionally stores patient information that can be used as evidence to infer the reason for an access, such as appointment and medication information. A **reason for access** is conceptually represented as a logical expression using information from an audit log entry and the evidence. If satisfied, then the access is **explained** by the reason and is deemed to be **appropriate**. For example, *an employee accessing a patient's record because of an appointment* can be represented as a logical condition on the audit log entry and appointment evidence [27].

The auditing system works as follows: The auditing system is given as input an audit log entry, the evidence and a set of valid reasons that are approved by the compliance officer. The system outputs whether the access is **appropriate** or **suspicious**. If the access is appropriate, the system also outputs the set of reasons that explain why it is appropriate.

Ideally, the auditing system would correctly classify each access such that the suspicious set of accesses contains all inappropriate accesses and no appropriate accesses. However, in this environment, it is difficult to determine the *gold standard label* for an access (i.e., is the access really appropriate or inappropriate) because of the volume of accesses (4.5M per week), the compliance officer's limited time to manually label data and restrictions on the data that researchers can view (we were only provided with de-identified data and therefore could not manually label the data ourselves). Instead, for the purpose of evaluating our techniques, we construct a **fake (random) audit log** as a proxy for inappropriate ("curious") behavior. The fake log is constructed by copying the **real** audit log, and for each audit log entry, replacing the employee with a randomly selected employee from the hospital system. The resulting fake audit log is of equal size and has the same distribution of diagnoses as the real audit log. This randomness is intended to mimic curious access patterns. Previous work has used similar approaches of inserting random accesses to detect inappropriate behavior [17]. We acknowledge that the lack of manually labeled data is a limitation of this work, but believe the methodology allows for the effective study of a difficult problem.

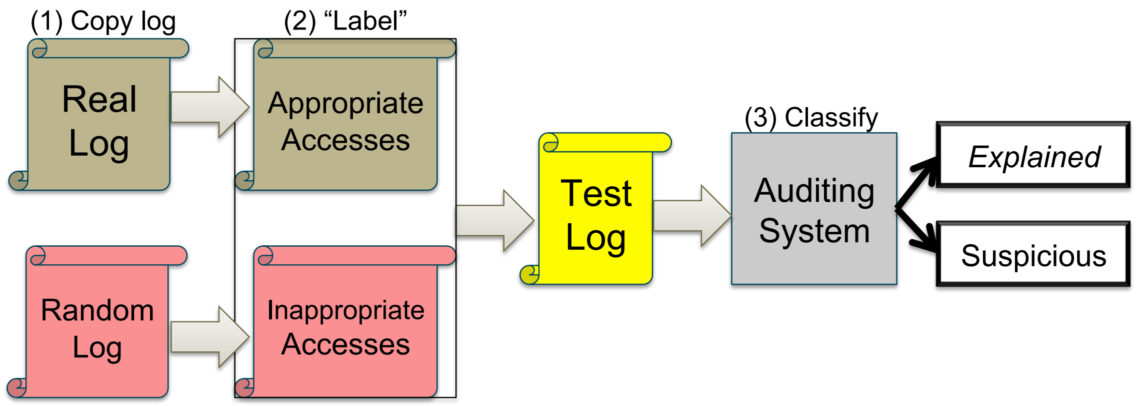


Figure 4.1: Experimental Methodology Overview: First, a fake (random) audit log is created by copying the real log and replacing each employee with a randomly selected employee. Second, accesses in the real log are labelled as appropriate and accesses in the fake (random) log are labelled as inappropriate. Third, the combined test log is given as input to the auditing system, which then classifies each access as explained or suspicious. The quality of the auditing system is determined by analyzing the accuracy of this classification.

Due to the lack of labeled data, the quality of the auditing system's classification is measured by analyzing which real and fake accesses can be explained (i.e., labeled as appropriate) as a proxy for correctly labeling appropriate and inappropriate accesses. While this methodology requires the assumption that all accesses in the real log are appropriate and all accesses in the fake log are inappropriate (even though there may be a few inappropriate accesses in the real log or visa versa), it is a reasonable strategy given that employees are generally well-behaved.

More formally, this objective can be expressed in terms of recall, precision and F-measure metrics. The recall for a set of reasons R, evidence E and audit log L is the fraction of real accesses explained from the entire real log (a recall of 1.0 implies that all real accesses are explained).

$$\mathsf{Recall}(L, E, R) = \frac{|access \in L \ explained \ by \ reasons \ R \ using \ evidence \ E|}{|L|}$$

The precision for a set of reasons R, evidence E, audit log L and fake audit log F is the fraction of real accesses explained versus real and fake accesses explained (a precision of 1.0 implies that no fake access is spuriously explained).

$$\mathsf{Precision}(L, F, E, R) = \frac{|access \in L \text{ } explained \text{ } by \text{ } reasons \text{ } R \text{ } using \text{ } evidence \text{ } E|}{|access \in L \cup F \text{ } explained \text{ } by \text{ } reasons \text{ } R \text{ } using \text{ } evidence \text{ } E|}$$

F-measure takes into account the system's ability to correctly classify accesses versus its ability to find all appropriate accesses.

$$\mathsf{F{-}Measure}(L, F, E, R) = 2 \times \frac{\mathsf{Precision}(L, F, E, R) \times \mathsf{Recall}(L, E, R)}{\mathsf{Precision}(L, F, E, R) + \mathsf{Recall}(L, E, R)}$$

The auditing system's objective is to select the reasons R to maximize the F-measure. In our case, a high F-measure is not sufficient to ensure the auditing system functions properly. That is, if the precision of the system is low, then even though many real accesses are explained, the utility of the system is minimal. Therefore, we also add an additional constraint that the precision must be greater than a minimum threshold (e.g., precision ≥ 0.85).

For a fair analysis, the size of the fake log was explicitly set to the size of the real log. While the number of inappropriate accesses is smaller than the number of appropriate accesses in practice, it would be unfair to measure the quality of the auditing system using a fake log that is significantly smaller than the real log. This configuration is unfair because the auditing system could simply classify each real and fake access as appropriate, even though the explanation is meaningless, and still attain near perfect precision and recall. Instead, with equally sized logs, the auditing system is forced to produce explanations that correctly explain real accesses without spuriously explaining fake accesses.

CHAPTER V

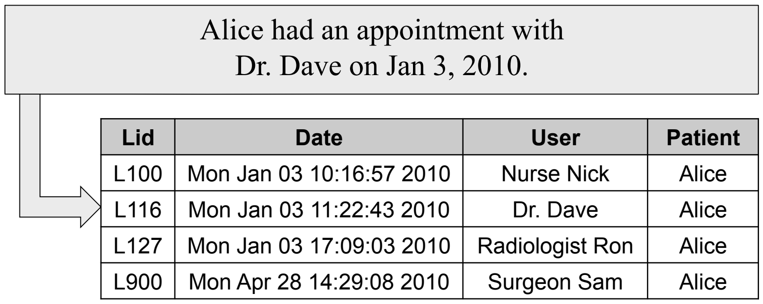
Explaining Accesses

5.1 Introduction

In recent years, laws and regulations have imposed a number of new requirements governing the responsible management of personal and private data. For example, in the United States, the Health Insurance Portability and Accountability Act (HIPAA) stipulates that individuals have the right to request an accounting of the disclosures of their protected health information (PHI) by hospitals and other healthcare providers (so-called "covered entities"). Recently, the U.S. Department of Health and Human Services proposed an expansion of this rule, which would require covered entities to provide individuals with detailed *access reports*, including the names of all people who have accessed their electronic PHI.[1]

Most modern electronic health records systems (EHRs) collect *access logs* automatically. For example, the University of Michigan Health System has built and deployed a web-based clinical EHR system called CareWeb[2]. To support regulatory compliance, each time an employee accesses a medical record via CareWeb, a record is added to the access log. While the precise format can vary among EHR systems, it is typically quite simple. CareWeb access logs contain four main attributes: *Times-*

[1]HHS Press Release, May 31, 2011.
http://www.hhs.gov/news/press/2011pres/05/20110531c.html
[2]http://www.med.umich.edu/mcit/carewebwe/help/overview.html



Lid	Date	User	Patient
L100	Mon Jan 03 10:16:57 2010	Nurse Nick	Alice
L116	Mon Jan 03 11:22:43 2010	Dr. Dave	Alice
L127	Mon Jan 03 17:09:03 2010	Radiologist Ron	Alice
L900	Mon Apr 28 14:29:08 2010	Surgeon Sam	Alice

Figure 5.1: Sample access log and explanation

tamp, *User_ID*, *Patient_ID*, and a coded description of the *Action* performed (e.g., *viewed lab reports*, or *updated history*).

One promising approach to providing access reports, and improving overall transparency, is the idea of *user-centric auditing*. Basically, the idea is to construct a portal where individual patients can login and view a list of all accesses to their medical records. When the underlying access logs are of the form described above, this is relatively straightforward. Unfortunately, the resulting access histories are often long and hard to analyze. Worse, the list of accesses often includes accesses by many people the patient does not know. (For example, the patient probably knows the name of his primary care physician, but he is not likely to recognize the name of the intake nurse or the radiologist who read his x-ray.)

In this chapter, we observe that in addition to asking who has accessed their medical records, patients will want to understand *why* these people accessed their records.

Example V.1. Consider a patient Alice who is using a user-centric auditing system. She logs into the patient portal and requests a log of all accesses to her medical record. The resulting log is shown in Figure 5.1, and includes accesses by four different hospital employees.

Looking at this log, Alice would like to understand the reason for each of these accesses. Ideally, we would like to provide an explanation for each access; if Alice clicks on a log record, she should be presented with a short snippet of text:

- **L100** Nurse Nick works with Dr. Dave, and Alice had an appointment with Dr. Dave.
- **L116** Alice had an appointment with Dr. Dave.
- **L127** Radiologist Ron reviewed Alice's x-rays for Dr. Dave.
- **L900** Surgeon Sam performed a surgery for Alice after Dr. Dave referred Alice to Sam.

One approach to providing explanations would require the user (e.g., the doctor) to enter a reason each time he accesses a medical record. While some systems may require this (e.g., [14]), it places a large burden on users.

Another approach would identify the access control rule(s) that allowed access to the medical record. (For example, users with a clinical appointment may be granted access to patient records.) Unfortunately, in environments like hospitals, it is very difficult to specify and maintain detailed access control policies [1]. (For example, residents and medical students change departments as often as once per week.) Further, overly restrictive policies can have disastrous consequences, interfering with patient care. As a result, it is typical for many more users to be granted access to a particular medical record than have a legitimate clinical or operational reason for accessing the record.

Instead, we would like to develop a technique to automatically produce informative explanations. Of course, there may be accesses for which we are not able to generate explanations. In these cases, if the access appears suspicious, the patient has the right to report the access to the hospital compliance office, and to request an investigation. However, developing a system to generate explanations automatically is useful both for the purpose of informing patients how their medical records are being used and for reducing the burden on the compliance office in handling complaints.

Interestingly, this also suggests a secondary application of explanations for the purpose of automated *misuse detection*. Because of the difficulties in expressing and maintaining access control policies up-front, rather than preventing data access, hospitals often err on the side of maintaining an access log in hopes of detecting misuse after the fact. Unfortunately, there are few technical tools for proactively detecting misuse from the access log. Common approaches often involve manual analysis in response to a complaint, or monitoring accesses to the medical records of VIPs (high-profile people).[3] Of course, manual analysis does not scale to the access logs collected by modern hospitals. (For example, in just one week, the University of Michigan Health System collected over 4 million access log records via CareWeb.) On the other hand, if we are able to automatically construct explanations for why accesses occurred, we can conceivably use this information to reduce the set of accesses that must be examined to those that are unexplained. While we are not likely to be able to explain every access, this process significantly reduces the set of records that are potentially suspicious.

5.1.1 Contributions

In this chapter, we study the novel problem of automatically explaining individual log records (accesses) in an access log. Our work is inspired by a fundamental observation: *For certain classes of databases, including those used to store EHR data, there is typically a clear reason for each access. Further, this reason can often be gleaned from information stored elsewhere in the database.* We provide an extensive empirical study in Section 5.4 using a large access log and EHR data from the

[3]For example, in 2008, hospital employees inappropriately accessed Britney Spears' medical record [62]. Also, in 2008, U.S. State Department employees were fired for inappropriately accessing President Obama's passport file [46].

Michigan Health System (CareWeb), which validates our hypothesis. Based in part on this observation, we make the following important contributions:

- In Section 5.2 we define a novel approach to modeling explanations. Intuitively, an explanation can be viewed as a connection from the data accessed (e.g., the Patient), through the database, and back to the user who accessed the data (e.g., the User).
- Before explanations can be used, they must be generated or specified. Our empirical study indicates that most accesses can actually be explained using a limited number of explanation types, or *templates*. For example, the fact that a patient had an appointment with the user who accessed his record is a general explanation type that can explain many different accesses.
- Nonetheless, we would like to remove some of the burden from the administrator in specifying explanation templates. Thus, in Section 5.3 we propose algorithms for automatically discovering templates that occur frequently in a given database (i.e., that explain a large number of accesses).
- Finally, in Section 5.4, we describe an extensive empirical study and experimental evaluation using data from CareWeb, which contains over 4.5 million accesses as well as records of appointments, visits, documents produced, and other information. Our experiments confirm the hypothesis that there is a reason for most accesses in our log, and that these accesses can be explained using data located elsewhere in the database.

5.2 Explaining Accesses

Given an entry in an access log, which describes both the data that was accessed (e.g., the patient's medical record) and the user who accessed the data, our goal is to construct a simple *explanation* describing the reason for the access. In addition, an explanation should satisfy the following basic properties:

- **Human Interpretable:** The reason why the access occurred should be easily understood. Among other things, we argue that an explanation should be logical and boolean (either it explains the access or not). In contrast, systems that provide probability distributions or other ambiguity are difficult to interpret.
- **General:** Explanations should take on a general form whereby a single explanation type or *template* explains many accesses by many users. For example, a patient having an appointment with the doctor who accesses his medical record is a common explanation template that can be used to explain many different accesses in the log.
- **Concise:** The explanation should be represented concisely.
- **Easy to produce/calculate:** Given a particular access, it should be easy to compute the explanation(s) for the access.

5.2.1 Explanation Templates

We begin by formalizing the structure of explanations, which can be used to describe why individual accesses occurred. We model an explanation based on the hypothesis that for every legitimate data access, there is a reason for the access, and in most cases the reason can be gleaned from information stored elsewhere in the database.

Example V.2. Consider the patients Alice and Bob, who log into the patient portal of their healthcare provider's electronic medical records system. To support transparency, the portal allows the patients to view a log of hospital employees who have accessed their records. Among others, the patients observe that an employee named Dr. Dave accessed their medical records. While this information may itself be useful, oftentimes it is important to provide further details, explaining why Dr. Dave accessed the record. Consider the following possible explanations:

A. Dr. Dave accessed Alice's medical record because Alice had an appointment with Dr. Dave on 1/1/2010.

B. Dr. Dave accessed Bob's medical record because Bob had an appointment with Dr. Mike on 2/2/10, and Dr. Dave and Dr. Mike work together in the Pediatrics department.

C. Dr. Dave accessed Alice's medical record because Dr. Dave previously accessed Alice's record.

D. Dr. Dave accessed Alice's medical record because Alice had an appointment with someone else.

Intuitively, an explanation should connect the user who accessed the data with the data itself (i.e., the patient's medical record). In examples (A-C), notice that there is a connection from the user who accessed the data (Dr. Dave), through the data in the database (appointment and department information), back to the data that was accessed (Alice or Bob's medical record). In contrast, the final explanation does not provide a connection between the user and the data. Consequently, the final explanation does not provide a meaningful description of why Dr. Dave in particular accessed Alice's record.

To capture this intuition more formally, we can model the explanation as a *path* through the database, beginning and ending in the log. We assume that the database stores a log of accesses, which records the time of the access, the user who accessed the data (Log.User) and a reference to the data that was accessed (Log.Patient). An *explanation template* is a tool that can be used to explain many individual accesses.

Definition V.3 (Explanation Template)**.** An explanation template is a stylized query on the database and log.

Consider a query Q of the following form, where $T_1, ..., T_n$ are (not necessarily distinct) tables in the database, and each C_i is an attribute comparison condition of the form $A_1 \theta A_2$ where $\theta \in \{<, \leq, =, \geq, >\}$.

```
SELECT Log.Lid, A_1, ..., A_m
FROM Log, T_1, ..., T_n
WHERE C_1 AND ... AND C_j
```

Let G be a graph, where each attribute in $Log, T_1, ..., T_n$ is a node. Let there be an edge from attribute A_1 to A_2 in G if (i) A_1 and A_2 are in the same tuple variable (i.e., $Log, T_1, ..., T_n$) or (ii) Q imposes a comparison condition between A_1 and A_2.

Query Q is an explanation template if there is a *path* P on G that starts at the data that was accessed (Log.Patient) and terminates at the user who accessed the data (Log.User), touching at least one attribute from each tuple variable mentioned in the query, and where no edge is traversed more than once.

Because an explanation template is a query, it can be used to explain why many different data accesses occurred. We refer to these data-specific descriptions (query results) as *explanation instances.* Notice that instances of a particular explanation template can be easily converted to natural language by providing a parameterized *description string.*

Example V.4. Consider the database and log provided in Figure 5.3. Explanations like (A) from Example V.2 can be derived from the following explanation template:

```
SELECT L.Lid, L.Patient, L.User, A.Date
```

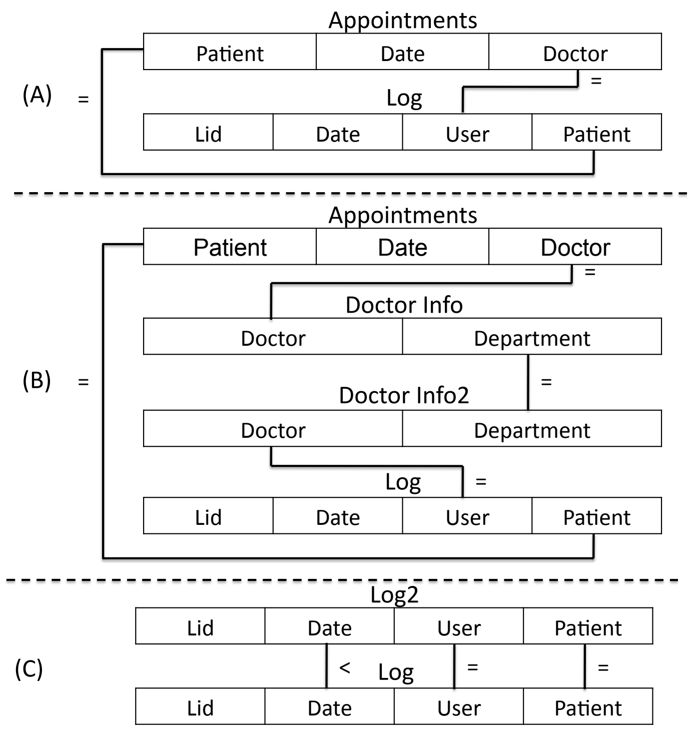


Figure 5.2: Paths through the explanation graph

Patient	**Date**	**Doctor**
Alice	1/1/2010	Dave
Bob	2/2/2010	Mike

(a) Appointments

Doctor	**Dept.**
Mike	Pediatrics
Dave	Pediatrics

(b) Doctor Info

Lid	**Date**	**User**	**Patient**
L1	1/1/2010	Dave	Alice
L2	2/2/2010	Dave	Bob

(c) Log

Figure 5.3: Example hospital database and log of accesses

```
FROM Log L, Appointments A
WHERE L.Patient = A.Patient
   AND A.Doctor = L.User
```

Figure 5.2 (A) shows the graph G associated with this explanation template. Notice that there is a path P that starts at $Log.Patient$ and terminates at $Log.User$. The edges between attributes in the same tuple variable are implicit.

Instances of this explanation template can easily be converted to natural language using a simple description string: "[L.Patient] had an appointment with [L.User] on

[A.Date].” For example, log record $L1$ can be explained using the description “Alice had an appointment with Dave on 1/1/2010.”

Explanations like example (B) can be derived from the following explanation template:

```
SELECT L.Lid, L.Patient, L.User, A.Doctor,
   A.Date, I1.Department
FROM Log L, Appointments A, Doctor_Info I1,
   Doctor_Info I2
WHERE L.Patient = A.Patient
   AND A.Doctor = I1.Doctor
   AND I1.Department = I2.Department
   AND I2.Doctor = L.User
```

Figure 5.2 (B) shows the graph associated with this explanation template. Instances of this explanation are easily expressed in natural language: “[L.Patient] had an appointment with [A.Doctor] on [A.Date], and [L.User] and [A.Doctor] work together in the [I1.Department] department.”

Notice that a single log record may have multiple explanation instances, generated from one or more explanation templates. For example, the query implementing explanation (A) would produce multiple results with Lid = L1 if Alice had multiple appointments with Dr. Dave. We consider each of these instances to be a valuable source of information; in practice, when there are multiple explanation instances for a given log record, we convert each to natural language and rank the explanations in ascending order of path length.

It is useful to draw a further distinction between what we will call *simple* explanation templates and their more complex *decorated* counterparts.

Definition V.5 (Simple Explanation Template)**.** Consider an explanation template and its associated graph G and path P. The explanation template is *simple* if it is not possible to remove any set of selection condition edges from G and still have a path P' from Log.Patient to Log.User.

Intuitively, a simple explanation provides a minimal connection between the data and the user who accessed it. Notice that explanation templates (A) and (B) in Example V.4 are both simple.

At the same time, simple explanations may not always be sufficient to express the desired semantics. For example, suppose we want to express the idea that an access occurred because the same user previously accessed the data (e.g., explanation (C) in Example V.2). A simple explanation template could partially capture the desired semantics as follows:

```
SELECT L1.Lid, L1.Patient, L1.User
FROM Log L1, Log L2
WHERE L1.Patient = L2.Patient
   AND L2.User = L1.User
```

However, to express the temporal aspect of the explanation, we need the additional selection condition `L1.Date > L2.Date`. Figure 5.2 (C) shows the graph associated with this explanation template. As a result, this *decorated* explanation always explains a subset of the accesses that are explained by the corresponding simple explanation.

Definition V.6 (Decorated Explanation Template)**.** A decorated explanation template is a simple explanation template with additional selection conditions added.

Finally, for modern databases with large schemas, the number and complexity of explanations can be very large, even if we only consider simple explanations. At the same time, we hypothesize that most explanations only require information from a few tables in the database. (We verify this hypothesis in Section 5.4.) For this reason, we may restrict the number of tables that a path can reference to an administrator-specified value T.

Definition V.7 (Restricted Explanation Template)**.** A restricted simple explanation template is a simple explanation template that only refers to at most T tables.

5.3 Mining Explanations

Before explanations can be used in any particular database, the appropriate explanation templates must be specified. One approach would require the security or database administrator to specify explanation templates manually. However, this can be a tedious process. Worse, due to the complexity of modern database schemas, a single administrator may not have complete knowledge of all the different reasons that data accesses occur.

While it is important to keep the administrator in the loop, we argue that the system should reduce the administrator's burden by automatically suggesting templates from the data. In this section, we describe our approach to mining templates from a given database. The administrator can then review the suggested set of templates before applying them.

The goal of the mining algorithms is to find the set of *frequent* explanation templates, or those that can be used to explain many accesses. Intuitively, this reduces the possibility of spurious results. The problem of mining frequent explanation templates is related to previous work on frequent pattern mining [4]. Indeed, our algorithms take a bottom-up pruning approach inspired by algorithms like *a priori*. At the same time, there are several important differences between the template mining problem and frequent pattern mining that prevent us from directly applying existing algorithms: First, we are mining connected paths between a start and end attribute in the schema. Second, our measure of frequency (support) is different; for explanation templates, frequency is determined by the number of accesses in the log that are explained by the template, so every path we consider must reference the log. Finally, the data is stored across multiple tables in the database, rather than in a single large file of transactions.

5.3.1 Problem Statement

Our goal is to find the set of explanation templates that occur frequently in a given database instance. We define *support* to be the number of accesses in the log that are explained by the template.

An extremely naive approach would enumerate all possible templates of the form described in Definition V.3. However, the number of possible templates is unbounded. Even if we restrict ourselves to simple templates without self-joins, the number of possible templates is still exponential in terms of the number of attributes in the schema.

To reduce the space, we make some practical simplifying assumptions: (1) We only consider simple explanation templates. (2) We only consider equi-joins between two tables if there exists a key-foreign key relationship, or if another relationship between two attributes is explicitly provided by the administrator. (3) An attribute and table can only be used in a self-join if the administrator explicitly allows the attribute to be used in a self-join. (4) We restrict the path length to M and restrict the number of tables referenced to T. We leave the task of developing algorithms for

Algorithm 1 One-Way Template Mining Algorithm

Input: Start attribute (Log.Patient), end attribute (Log.User), support (S), max path length (M), restricted number of tables referenced (T), the set of edges from the schema (Edges) and the database instance (D).

Output: Set of supported explanation templates (up to the max length).

```
 1: Length = 1
 2: Paths = {Edges that begin with the start attribute}
 3: Explanations = {}
 4: while Length ≤ M do
 5:    New Paths = {}
 6:    for Path $p \in Paths$ do
 7:       for Edge $e \in Edges$ do
 8:          if areConnected(p, e) then
 9:             Candidate Path = $p.append(e)$
10:             if isARestrictedSimplePath(Candidate Path) then
11:                if Support(Candidate Path, D) ≥ S then
12:                   New Paths.add(Candidate Path)
13:                   if isAnExplanation(Candidate Path) then
14:                      Explanations.add(Candidate Path)
15:    Paths = New Paths
16:    Length += 1
17: Return Explanations
```

mining more complex (decorated) explanation templates to future work.

Definition V.8 (Explanation Mining)**.** Given a database D and a log of accesses L, return those explanation templates of length at most M, that reference at most T tables and that explain (support) at least $s\%$ of the accesses in the log, where the edges in the path are restricted to attributes from the same tuple variable, key relationships, specified self-joins, or administrator-specified relationships.

Example V.9. Continuing with Example V.2 and the database in Figure 5.3, template (A) has support of 50% (from access L1), and template (B) has support of 100% (from accesses L1 and L2).

5.3.2 One-Way Algorithm

We begin by describing a basic algorithm. (Details are provided in Algorithm 1.) The administrator provides the start attribute Log.Patient (the data that is accessed), the end attribute Log.User (the user who accessed the data), the minimum support S, the maximum length M, the maximum number of tables referenced T, the schema, and the database. We restrict the set of edges (denoted as *Edges*) that can be used in explanations as described in Section 5.3.1. An initial set of paths of length one are created by taking the set of edges that begin with the start attribute Log.Patient. The goal of the algorithm is to find the set of *supported* explanation templates, which are those templates that explain at least $s\%$ of the accesses.

The algorithm finds the set of supported templates as follows: First, for each path at the current length, and for each edge, the algorithm tests if the two are connected. Intuitively, the path and edge are connected if the last attribute in the path is the same attribute as the first attribute in the edge. Second, for those connected paths and edges, the path and edge are combined by appending the edge to the right end of the path. Third, the algorithm checks if this *candidate* path is a restricted simple path. Intuitively, the candidate path is simple if it begins at the log and continues to join with previously untraversed tables until the log is reached (the path traverses each node at most once and at most two nodes per table). The candidate path is a restricted simple path if it references no more than T tables (a path that references a table and a self-join for that table is counted as a single reference). Next, the candidate path is converted to SQL and evaluated on the database to calculate the path's support. We calculate the support using the following query:

```
SELECT COUNT(DISTINCT Log.Lid)
FROM Log, T_1, ..., T_N
WHERE C
```

If the support is greater than or equal to $S = |Log| \times s\%$, then the path is added to the set of new paths that will be used in the next iteration of the algorithm.

Furthermore, if the path has the appropriate start and end attributes, then the path is also an explanation template, and is marked accordingly. The algorithm repeats for paths of increasing length until the maximum path length is reached.

Example V.10. Consider the database shown in Figure 5.3; the one-way algorithm works as follows: The input set of edges includes key-foreign key equi-joins such as {Log.Patient = Appointments.Patient, Appointments.Patient = Log.Patient, Log.User = Appointments.Doctor, Appointments.Doctor =Log.User} and the administrator-provided self-join {Doctor_Info.Department = Doctor_Info2.Department}. The initial set of paths is: {Log.Patient = Appointments.Patient}. This first path is converted into SQL and has the selection condition `Log.Patient = Appointments.Patient` and is evaluated on the database. The path has support of 100%.

Next, connected edges are appended onto the path. For example, one candidate path has the selection condition: `Log.Patient = Appointments.Patient AND Appointments.Doctor = Log.User`. This candidate path is also an explanation since it has the correct start and end attributes. The explanation has support of 50%.

The one-way algorithm works in a bottom-up manner to find the supported explanation templates. We observe several important properties of the algorithm: First, the paths must always include an attribute from the Log in order to calculate the support for the path; if there was no Log attribute, then it would be impossible to count the number of log entries explained.

Second, the support function is monotonic. If a path P of length $\ell - 1$ does not have the necessary support (i.e., does not explain $\geq s\%$ of the accesses in the log), then adding additional edges to the path will never produce an explanation template with the necessary support. Thus, the bottom-up algorithm is able to prune certain paths that are guaranteed not to have the necessary support.

The mining algorithms presented in this section have some similarities to previous work on pattern mining [4, 5, 39, 74], although existing algorithms do not solve our problem directly. The main differences between our problem and classical frequent pattern mining are as follows: First, we are mining connected paths between a start and end attributes in the schema, where the classical problem mines item sets. Second, our metric for frequency (support) is determined by the number of accesses in the log that are explained by the template. Therefore, every path we consider must reference the log. Additionally, the structure of the patterns that are mined and where the data is stored differs from the classical problem. For instance, the templates represent logical expressions that data in the database must satisfy. In contrast, the classical problem learns relationships between actual values. Lastly, the data is stored across multiple tables in the database, rather than in a single file of transactions.

Performance Optimizations

We apply three performance optimizations for the algorithm:

Caching Selection Conditions and Support Values: We observe that multiple paths may have the same selection conditions, even though the paths traverse the explanation graph in different orders. Since the order in which the selection conditions are applied does not change the result, these paths are guaranteed to have the same support (i.e., $R.attr = T.attr$ is equivalent to $T.attr = R.attr$). Thus, a simple optimization is to cache the support of each path that has already been tested. Then, before the next path's support is calculated, the algorithm checks if some variation of the path (with an equivalent selection condition) has already been tested. If so, the algorithm does not need to evaluate the query on the database and can use the previously recorded support value instead.

Reducing Result Multiplicity: The multiplicity of data in the database can impact performance. For example, from Example V.4, if Alice had three appointments with Dr. Dave, then there would be three instances of explanation (A) for the same log id. These additional rows in the output make computing the support (i.e., the distinct set of log ids) more costly. Therefore, since it does not matter how many times a given log id is in the result, the performance can be improved by reducing the number of rows in the result. To remove duplicates from each table, we use a subquery to extract the distinct set of rows from the table, while only projecting those attributes needed for the path. For example, the query from Example V.4 can be rewritten as follows:

```
SELECT COUNT(DISTINCT L.Lid)
FROM Log L,
   (SELECT DISTINCT Patient, Doctor
   FROM Appointments) A
WHERE L.Patient = A.Patient
   AND A.Doctor = L.User
```

Skipping Non-Selective Paths: For many (short) paths, the selection conditions are not selective and return most of the log. Computing the support for these paths wastes time because these non-selective paths typically have sufficient support and are not pruned. Therefore, the algorithm's performance can be improved by passing these non-selective paths directly to the next iteration of the algorithm, instead of calculating their support. We determine if a path is likely to have sufficient support by asking the database optimizer for the number of log ids it expects to be in the result of the query. If the value is greater than the desired support $S \times c$ (where c is a constant like 10), the system skips this path and adds it to the set of paths to try in the next iteration of the algorithm. In the special case when the path is also an explanation, the path is not skipped. The constant c is used to account for the optimizer's estimation error. Using this optimization, the system trades off pruning some paths in order to not have to calculate the support of the non-selective paths. Even in the worst case when the database optimizer significantly errs with its estimation, the output set of explanation templates does not change because paths are not discarded; rather, they are tested in the next iteration of the algorithm.

5.3.3 Two-Way Algorithm

Intuitively, the two-way algorithm constructs paths in two directions: from the start to the end, and from the end to the start. The two-way algorithm is initiated with the edges that begin with the start attribute and the edges that terminate with the end attribute. The paths that begin with the start attribute are extended to the right with connected edges until the end attribute is reached (i.e., the one-way algorithm), while the paths that terminate with the end attribute are extended to the left with connected edges until the start attribute is reached. Therefore, an optimized algorithm would have them meet in the middle.

Bridging Paths

The one-way and two-way algorithms explore all paths that have the desired support. However, the goal of the algorithms is to find supported explanation templates. Therefore, by enforcing the constraint that paths must start and end with particular attributes, we can restrict the set of paths the algorithms must consider. Moreover, since we have paths extending from the start and end attributes, we can combine,

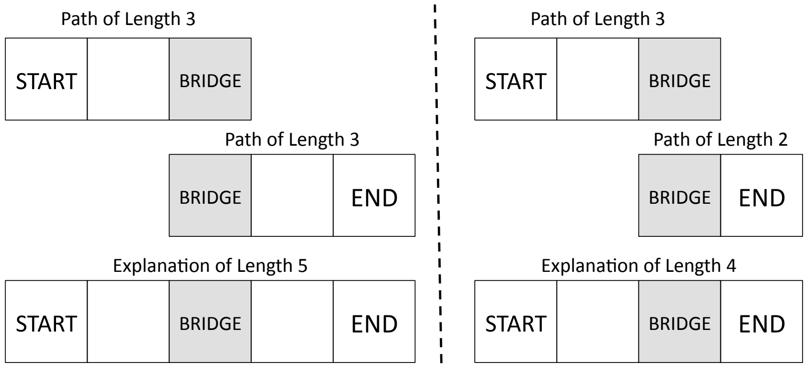


Figure 5.4: Bridging paths to create explanations

or *bridge*, these paths.

Consider the case where the two-way algorithm has executed and produced all supported paths up to length ℓ. The algorithm can use these paths to easily construct the set of candidate explanation templates up to length $2\ell - 1$ (the candidate templates are a superset of those templates that have the necessary support). These candidate templates can be produced by connecting those paths that begin with the start attribute to those paths that terminate with the end attribute as shown in Figure 5.4. The remaining paths that do not start or end with one of these attributes can be ignored.

More concretely, candidate templates of length n ($2 \leq \ell < n \leq 2\ell - 1$) can be produced by taking paths of length ℓ that begin with the start attribute and connecting them to paths of length $n - \ell + 1$ that terminate with the end attribute. We say the paths are *bridged* because the algorithm requires that the edges where the two paths are connected (the bridge edge) are equivalent. As a result, the length of the combined path is one less than the sum of the individual path lengths. Once the set of candidate templates is produced, the support for each candidate is tested.

Example V.11. Template (B) from Example V.2 can be created by bridging the following two paths:

```
SELECT COUNT(DISTINCT L.Lid)
FROM Log L, Appointments A, Dept_Info I1,
   Dept_Info I2
WHERE L.Patient = A.Patient
   AND A.Doctor = I1.Doctor
   AND I1.Department = I2.Department
```

```
SELECT COUNT(DISTINCT L.Lid)
FROM Log L, Dept_Info I1, Dept_Info I2
WHERE I1.Department = I2.Department
   AND I2.Doctor = L.User
```

Notice that the combined path has the appropriate start and end attributes, and the condition I1.Department = I2.Department can be used to bridge the paths.

When the length of the desired path is greater than or equal to 2ℓ, the candidates cannot be constructed from the paths that have been found thus far. While the algorithm can still use the paths to restrict the ends that the candidate template can take, the algorithm does not have knowledge about which edges should be included in the middle of the explanation. Thus, the algorithm must consider all combinations of edges from the schema to bridge these paths.

Bridging paths is beneficial because it can greatly reduce the space of candidate templates to test. In general, since the algorithm's performance is proportional to the number of candidates that must be tested, bridging improves performance because the start and end attribute constraints are pushed down in the algorithm. However, if only short paths are mined, but long explanation templates are desired (i.e., $n > 2\ell$), then the number of candidates exponentially increases with the length. Thus, for some length n, it is then no longer beneficial to bridge paths.

5.4 Experimental Evaluation

To test our ideas, we conducted an extensive experimental study using a real access log and database from the CareWeb system at the University of Michigan Health System. Our experiments aim to answer the following questions:

- *Do explanations (as we described them in Section 5.2.1) exist in real databases?* We find that explanations like those described in Example V.2 occur in the real hospital database and can explain over 94% of the accesses in the log.
- *Can we mine explanation templates efficiently?* We measure the performance of the one-way, two-way and bridged algorithms from Section 5.3 and find they are able to discover explanation templates automatically and efficiently. Moreover, the bridging optimization can improve performance in particular cases.
- *How effective are the mined explanation templates at correctly classifying future accesses?* We measure the precision and recall of the mined explanations and find that shorter explanations provide the best precision, but moderate recall. Longer explanations, including those that use the group information, can be used to improve recall.
- *Is the same set of explanation templates mined over time?* We find that the set of explanation templates discovered by the mining algorithms is relatively stable across time.

5.4.1 Implementation & Environment

Our system is a Python layer on top of PostgreSQL[4]. This layer constructs paths from the schema and executes queries on the database to determine an explanation's

[4]http://www.postgresql.org

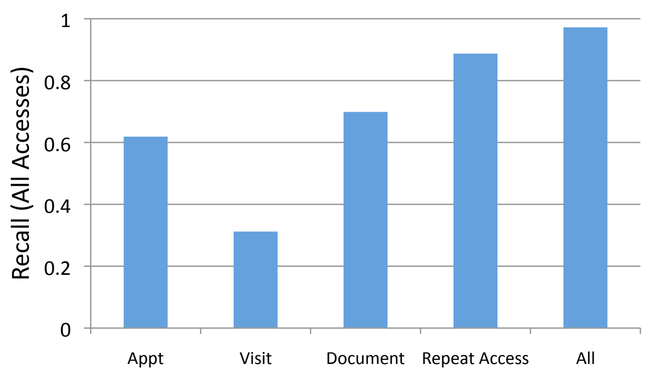


Figure 5.5: Frequency of events in the database for all accesses.

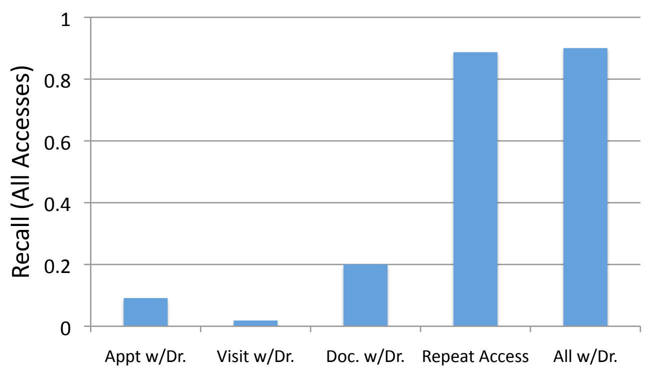


Figure 5.6: Hand-crafted explanations' recall for all accesses.

support. Clustering was performed with a Java implementation of the graph modularity algorithm. The experiments were executed on a dual core CPU with 4 GB of RAM, running Red Hat Linux.

5.4.2 Results

Explanations in a Real Data Set

Our first set of experiments tests the fundamental hypothesis that accesses in the log can be explained using data stored elsewhere in the database.

We began by measuring the proportion of accesses in the log pertaining to a patient such that the patient had some type of *event* recorded elsewhere in the database. In particular, we started by measuring the proportion of patients who had an appointment (Appt), visit, or document produced (Document). Figure 5.5 shows the

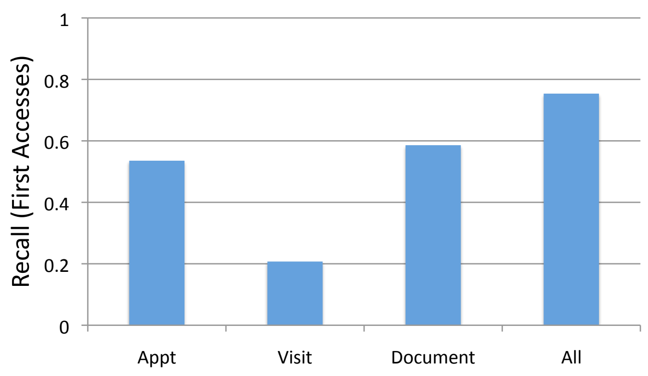


Figure 5.7: Frequency of events in the database for first accesses.

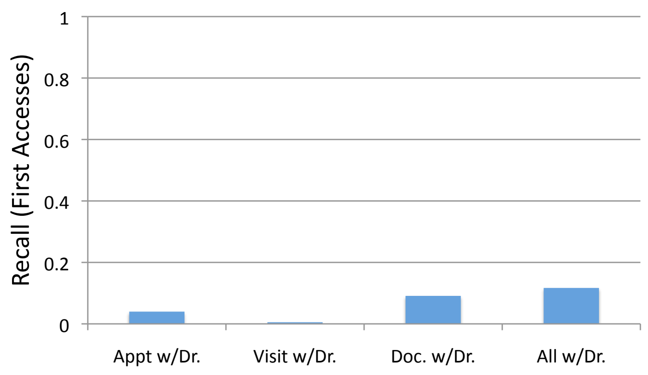


Figure 5.8: Hand-crafted explanations' recall for first accesses.

frequency of these events in the log. (The recall of Appointment would be 1.0 if every patient whose record was accessed also had an appointment with someone listed in the database.) As expected, many patients had an appointment with someone or had a document produced (e.g., a doctor's note added to the file) by someone. Additionally, a majority of the accesses can be categorized as repeat accesses, meaning that the same user accessed the same patient's record for an additional time. When we combined all these events together, approximately 97% of all accesses corresponded to a patient who had some type of event in the database. Interestingly, a small percentage of the accesses did not correspond to a patient who experienced some type of event. We suspect that this is largely due to the incomplete data set. For example, appointments outside of the study's timeframe were not considered.

Of course, these events do not constitute explanations since they do not necessarily connect the patient whose record was accessed to the specific user who accessed the record. (For example, a patient may have an appointment listed, but it may not be with the person who accessed her record.) To measure the proportion of accesses that can be explained using the approach described in Section 5.2.1, we hand-crafted a simple set of explanation templates, based on common reasons for medical records to be accessed, that test if the patient: (i) had an appointment with the specific doctor who accessed the record (Appt w/Dr.), (ii) had a visit with the doctor (Visit w/Dr.), (iii) had a document produced by the doctor (Doc. w/Dr.), or (iv) the access was a repeat access.

Figure 5.6 shows the recall for the explanations (i.e., proportion of the log records explained). While the repeat accesses can still explain a majority of the accesses, the recall of the other explanations is lower. This result is expected because the appointments, visits and documents produced typically only reference the primary doctor in charge of the patient's care. Therefore, using these basic explanation templates, we cannot explain why a nurse accesses a medical record. Even with this lower recall, these explanation templates can still explain 90% of the accesses.

Although repeat accesses make up a majority of the log, it is more challenging and interesting to explain why a user accesses a record for the first time. To do this, we analyzed all of the *first accesses* in the log, where a user accesses a patient's medical record for the first time. (Notice that since we only have a subset of the log, some accesses that are actually repeat accesses appear to be first accesses due to truncation.)

Figures 5.7 and 5.8 show the recall for the events and explanation templates among only the first accesses. When combined, the explanation templates for appointments, visits, and documents produced explain approximately 11% of first accesses (see the *All w/Dr.* bar in the chart). Ideally, we should be able to explain approximately 75% of the first accesses because 75% of the patients have some corresponding event in the database (see Figure 5.7). For the remaining 25% of the patients, we have no corresponding event. We attribute this result in large part to the incomplete data set.

In the next sections, we will show that it is possible to improve recall by adding missing data, and also by mining additional explanation templates.

Mining Explanations

Our next set of experiments measured the performance of the mining algorithms presented in Section 5.3. We ran the algorithms on the first accesses from the first six days of the log, with the combined data sets A and B, and the added group information (see Section 6.2 for details on how these groups are constructed). Based on an initial study, we set the support threshold to 1%. (A support threshold of 1% was sufficient to produce all of the explanation templates that we constructed by hand except one template where a doctor has a visit with a patient, which had a very small support.) We restricted the size of templates to $T = 3$ tables. We allowed self-joins on the Groups.Group_id attribute and the department code attribute. The algorithms utilized the optimizations described in Section 5.3.2. Due to how the data was extracted, data set B identifies users with a key *audit_id*, and data set A identifies users with a *caregiver_id*. We used a *mapping table* to switch from one identifier to the other. Thus, to deal with the slight difference in how the data was extracted, we did not count this added mapping table against the number of tables used.

We evaluated the algorithms based on their performance and their ability to find

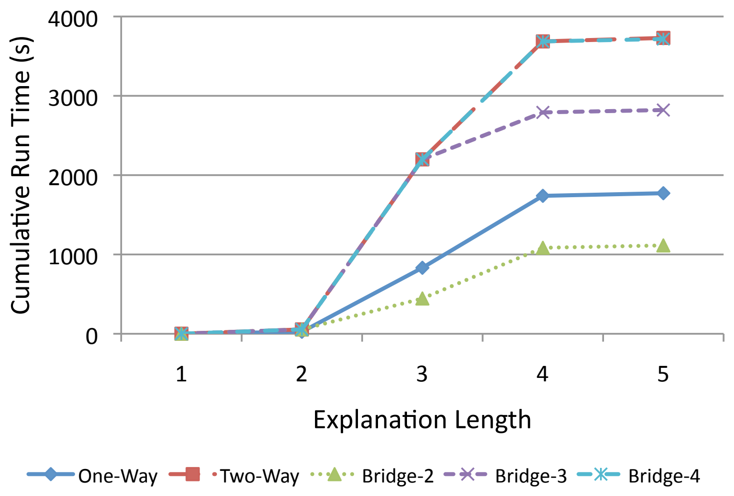


Figure 5.9: Mining performance (Data sets A & B, log days 1-6, T = 3, s = 1%)

the hand-crafted explanation templates we previously constructed. Figure 5.9 shows the cumulative run time of the various algorithms by path length (the *length* corresponds to the number of joins in the path). The algorithms mined explanations up to a length of five when the path included a self-join and the mapping table. Bridge-ℓ indicates that we used paths up to length ℓ for bridging. For our experimental setup, the Bridge-2 algorithm was the most efficient because it pushes the start and end constraints down in the algorithm. The one-way algorithm was faster than the two-way algorithm because the two-way algorithm considers more initial edges. Without the optimizations described in Section 5.3.2, the run time increases by many hours.

Each algorithm produced the same set of explanation templates. Moreover, it is worth noting that our mining algorithms were able to discover all the supported hand-crafted explanation templates we described in the chapter such as appointments with doctors, appointments with users that work in the same department, and appointments with users that are in the same group.

It is important to note that this chapter is not intended to be a full performance study. Rather, we intend this as a proof of concept, demonstrating that explanation

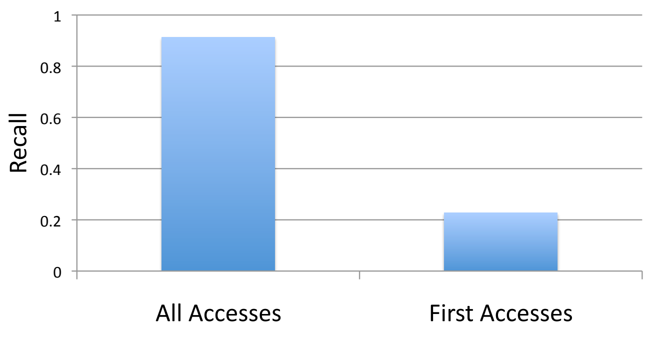


Figure 5.10: Recall for direct explanations.

templates can be mined automatically from a real data set. Therefore, the administrator's time can be saved if algorithms can find these explanation templates.

Predictive Power Of Direct Explanations

Next, we evaluated the predictive power of the explanation templates that were produced by the mining algorithm. To do this, we split the log and other data (data sets A and B) into two sets (training and testing). Each patient was assigned to one of the two sets. We then ran the mining algorithm on the training set and extracted direct explanation templates, where a direct explanation template has a path of length two. The mined templates included explanations such as the patient had an appointment with the user, the user administered a medication for the patient or the decorated template that a user previously accessed a patient's record. The mined explanation templates were then evaluated on the testing set. We measured the precision and recall using the combined real and fake logs.

Figure 5.10 shows the recall for the mined direct explanation templates. When all access in the testing set are included, the templates can explain approximately 90% of the accesses. However, a large proportion of the accesses are explained with the repeat access explanation. If we instead look at only first accesses, we find that only 22% of the accesses are explained. It is important to note that the precision

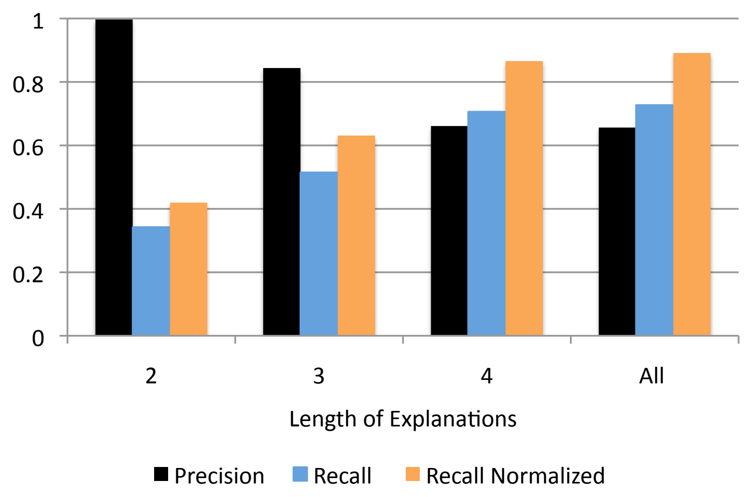


Figure 5.11: Mined explanations predictive power for first accesses (Data sets A & B, trained on days 1-6, tested on day 7)

for these direct explanation templates is 99% because it is unlikely that a randomly generated access corresponds to a real appointment.

Predictive Power Of Explanations

Using the explanation templates mined from the first six days of accesses from Section 5.4.2 on data sets A and B, we tested the predictive power of the explanation templates on the seventh day of accesses using the same fake log that was described in Section 6.2.2. The goal is to determine if the mined explanation templates can correctly explain real accesses, while not spuriously explaining fake accesses. Figure 5.11 shows the results for explanations of various lengths, and the results when all of the explanations are tested together (All) for first accesses.

Explanation templates of length two have the best precision, while the recall is approximately 34% (42% normalized). These short explanations are like explanation (A) from Example V.2, where the doctor has an appointment with the patient. The precision is high for these explanations because it is very unlikely that, for example, a fake access corresponds to an actual appointment. We believe this is a result of the user-patient density being so small.

It is also important to point out that the recall for these length-two explanation templates is higher when data set A and B are combined compared to when only data set A is included in Figure 6.4 (the recall increases from 13% to 34% when analyzing the first accesses for day seven). This change in recall shows that as more data is added to the database, we can explain additional accesses. With a complete data set, we argue that we can explain more accesses.

As explanation template paths get longer, the recall increases while the precision drops. Explanations of length three, which typically combine event information of two types (e.g., appointments and medications) have a recall of 51% (65% normalized). Explanation templates of length four, which use group information, increase the recall to 73% (89% when normalized). The precision drops since it is more likely that the user from a fake access corresponds, for example, to an appointment with another user that is in the same group. When all the explanations are combined and tested together, we find the recall and precision only change slightly from the length-four explanation templates because the longer explanations typically are more general versions of the shorter explanations. Therefore, the longer explanation templates explain most of the accesses that the shorter templates explain. For example, template (B) from Example V.2 explains all those accesses explained by template (A).

We analyzed the department codes for which we could not explain the largest number of accesses. The top four departments were: Nursing-Vascular Access Service, Anesthesiology, Health Information Management, and Paging & Information Services. The users in the vascular access service department typically assist with IVs. Therefore, since our data set does not explicitly record why each nurse treated

	# Explanation Templates				
Length	Days 1-6	Day 1	Day 3	Day 7	Common Templates
2	11	11	11	12	11
3	241	257	231	268	217
4	25	25	25	27	25

Table 5.1: Number of explanations mined

a patient and these nurses assist many different departments, it makes sense that the mined explanation templates could not explain their accesses.

For the evaluation, we used group information from any depth in the hierarchy (see Section 6.2 for details on how these groups are constructed). However, we observe that not every event type should use the same depth. For example, when only data set A was used, we had a precision of approximately 93% for depth 1, however when data set B was included, the precision dropped to 66%. Therefore, group information at one depth may be sufficient to explain an access with an appointment, but group information at another depth may be necessary to explain accesses with medication information to attain a desired level of precision. In the future, we will consider how to mine decorated explanation templates that restrict the groups that can be used to better control precision.

Stability of Explanations

Next, we measured the stability of the explanation templates that were mined over different time periods to determine if there exists a set of consistently occurring explanation templates. To do this, we ran the mining algorithms on different subsets of the log: days 1-6, day 1, day 3 and day 7. Table 5.1 shows the number of explanation templates produced per time period. For our data sets, the number of explanations that are produced is small enough for an administrator to manually analyze and approve those semantically correct explanations. Moreover, there is a set of *common explanation templates* that occur in every time period. Therefore, we believe the explanation templates mined represent generic reasons why medical records are accessed.

We did observe a small difference in the explanations mined across time periods. For example, on the seventh day, a twelfth length-two explanation template was added because there were more accesses corresponding to visits. We found larger variability in length-three explanations. This variation occurred from those explanation templates that connected two event types. For example, the path through radiology information to medication information occurred frequently on some days, but did not occur frequently during others.

5.5 Summary

In this chapter, we outlined the main components of the explanation-based auditing system. Many systems, including EHRs, collect access logs. While this information is sufficient to explain who has accessed a particular piece of data (e.g., a patient's medical record), it is not usually enough to explain *why*. To address this problem, we introduced a framework that generates explanations automatically. Our work is based on the fundamental observation that accesses in specific classes of databases occur for a reason, and the reason can be inferred from data in the database. Thus, we model an explanation as a path that connects the data accessed to the user who accessed it, by way of data elsewhere in the database. Producing explanations for a large database can be time consuming. Instead, we provided algorithms to automatically mine explanation templates from the data. We evaluated our system on a real log and data set from the University of Michigan Health System. Using our model, we can explain over 90% of the accesses with high precision.

CHAPTER VI

Dealing With Missing Data

6.1 Overview

So far we have only considered explanations that can be expressed solely in terms of the data stored in the database. Unfortunately, real databases are typically not perfectly curated. Information may be missing from the database, or relationships may not be recorded. For example, consider a nurse in a hospital who works directly with a doctor. When a patient has an appointment with the doctor, the appointment is recorded in the database, and we can use explanations of the type described in Section V to explain the doctor's accesses. Unfortunately, appointments are typically only scheduled with the doctor, not with the nurse. Thus, we cannot explain why the nurse accessed the patient's record, even though the access is appropriate.

To explain these types of accesses, we must deal with "missing" data in the database. One common type of missing data are the relationships between users of the database. While a database may store information such as the department each user works in, our evaluation shows that additional information is still needed to explain accesses. Moreover, as we found in our data set, nurses and doctors are assigned to different departments. We hypothesize that information used to explain an access such as an appointment often is stored in the database with a reference to a single user, but that information can be used to explain why many other users access the data. Thus, if the database stored relationships among users, additional accesses could be explained.

Another common type of missing data is information describing the diagnoses each department is responsible for treating [28]. Specifically, EMR databases store codes from the International Statistical Classification of Diseases and Related Health Problems, revision 9 (ICD-9) that describe over 6K diagnoses, diseases and procedures (e.g., acne, breast cancer, encounter of chemotherapy), which are later used for billing purposes (for conciseness, this thesis uses the term diagnosis to refer to the ICD-9 diagnoses, diseases and procedures). Our hypothesis is that specific hospital employees are responsible for treating each diagnosis (i.e., ICD-9 code) and that a group of employees treat a diagnosis together. Thus, if the database stored this diagnosis responsibility information, additional accesses could be explained such as: *Dr. Carl accessed Alice's record because Dr. Carl works in the Oncology Department and Alice has cancer.* The challenge is to determine which employees are responsible for each diagnosis.

After these types of missing data are added to the database, the mining algorithms can automatically use the data to discover new explanations. We evaluate if adding missing data improves the quality of the auditing system. However, it is important to note that by adding data, the auditing system can potentially produce false positive explanations, where an access is explained through a spurious connection. We evaluate the trade-offs of including the missing data at the end of this chapter.

6.2 Collaborative Groups

A natural method to determine relationships between users of a database is to analyze user access patterns [17, 56]. In general, users who work together often access the same data. Using the log of accesses, we can automatically discover *collaborative groups* of users who access the same data often and use these groups to explain more accesses. For example, an explanation for why the nurse accessed the patient's medical record could be described as follows: the nurse accessed the patient's medical record because the nurse works with the doctor and the doctor had an appointment with the patient.

Next, we outline one possible approach to construct collaborative groups that we found to be effective for our data set. However, we note that there has been extensive work on clustering [34, 80], and alternative approaches are possible. In general though, we treat these algorithms as a black box that produces a set of relationships between users of the database. Once this data is plugged into the database, our explanation mining algorithms can incorporate the information to find additional supported templates.

6.2.1 Extracting Collaborative Groups

Given an access log, we can model the relationships between database users using a graphical structure. We use a method similar to that presented by Chen et al. [17]. Let a node in the graph represent a user. An edge exists between two users if the users access the same data. We assign weights to the edges to signify the strength of the users' relationship. To do this for a log of accesses that occur between some start and end time that has m patients and n users, we construct an $m \times n$ matrix A. The index $A[i, j]$ represents the inverse of the number of users (including user j)

that accessed patient i's record. More formally, if user j does not access i's record, then $A[i,j] = 0$, else:

$$A[i,j] = \frac{1}{\#\ users\ who\ accessed\ patient\ i\text{'}s\ record}$$

The weight of an edge between user u_1 and user u_2 can be found in $W[u_1, u_2]$ where $W = A^T A$. Intuitively, $W[u_1, u_2]$ represents the similarity of two users' access patterns, relative to how often a particular record is accessed. Our current approach does not adjust the weight depending on the number of times a user accesses a specific record, but rather it only considers if a user accesses the record. A node's weight is defined as the sum of the connected edges' weights.

Given the graph structure, we can directly apply weighted graph clustering algorithms. Specifically, we use an algorithm that attempts to maximize the graph modularity measure [61]. Intuitively, optimizing for the graph modularity measure attempts to maximize the connections (and weights) for nodes within a cluster and minimize the connections between nodes that reside in different clusters. The algorithm is also parameter-free in the sense that it selects the number of clusters automatically.

After running the clustering algorithm once, the algorithm outputs a set of clusters and an assignment of users to clusters. We can recursively apply the clustering algorithm on each cluster to produce a hierarchical clustering. Intuitively, clusters produced at the lower levels of the hierarchy will be more connected than clusters produced at higher levels. In Section 6.2.2 we show how this affects the precision and recall of explanations.

Example VI.1. Consider the log of accesses in Figure 6.1 that lists which users have accessed which patient's medical records. From the log, we can construct the

Patient	User Ids
A	0, 1, 2
B	0, 2
C	1, 2
D	2, 3

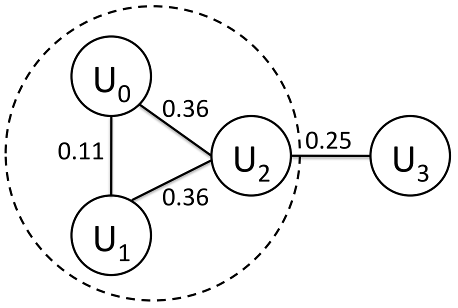


Figure 6.1: (i) Example log of accesses (per patient), and (ii) the associated graphical representation with clustering.

matrix A. For example,

$A[patient\ A, user\ 0] = \frac{1}{3}$ since three users accessed patient A's record. After evaluating $W = A^T A$, we find the edge weights that are labeled on the graphical representation. After running the clustering algorithm, users 0, 1 and 2 are assigned to the same cluster.

After clustering, the table *Groups(Group_Depth, Group_id, User)* is added to the database. By applying a self-join on this table, the mining algorithms can use these groups to explain additional accesses.

Example VI.2. Nurse Nick's access of Alice's record in Figure 5.1 occurred because Nick works with Dr. Dave, and Dr. Dave had an appointment with Alice. The corresponding explanation template is expressed as follows:

```
SELECT L.Lid,L.Patient,L.User,A.Date,G1.User
FROM Log L, Appointments A,
   Groups G1, Groups G2
WHERE L.Patient = A.Patient
   AND A.Doctor = G1.User
   AND G1.Group_id = G2.Group_id
   AND G2.User = L.User
```

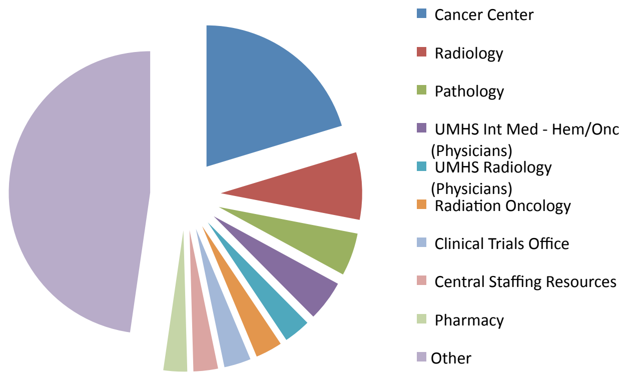


Figure 6.2: Collaborative Group I (Cancer Center)

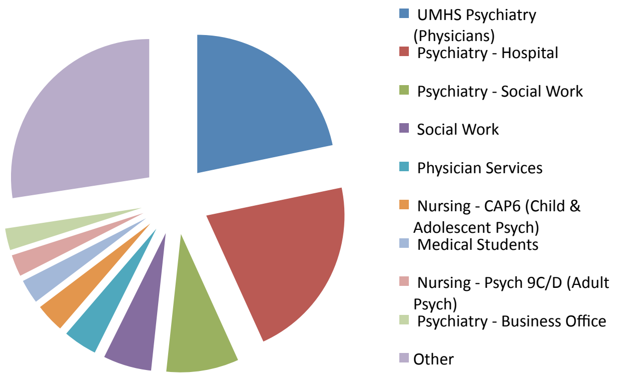


Figure 6.3: Collaborative Group II (Psychiatric Care)

6.2.2 Results

When a patient has an appointment, the appointment is scheduled with the doctor. However, the nurses who work with the doctor also typically access the patient's medical record. For this reason, we could only explain 11% of the first accesses, even though 75% of these patients have some associated event (e.g., an appointment with someone). To improve recall, we applied the algorithm described in Section 6.2.1 to cluster users who access similar medical records using the first six days of accesses in the log, and we added these collaborative groups to the database.

Since we were working with de-identified data, it was impossible to systematically verify the correctness of the resulting groups. However, a manual inspection of

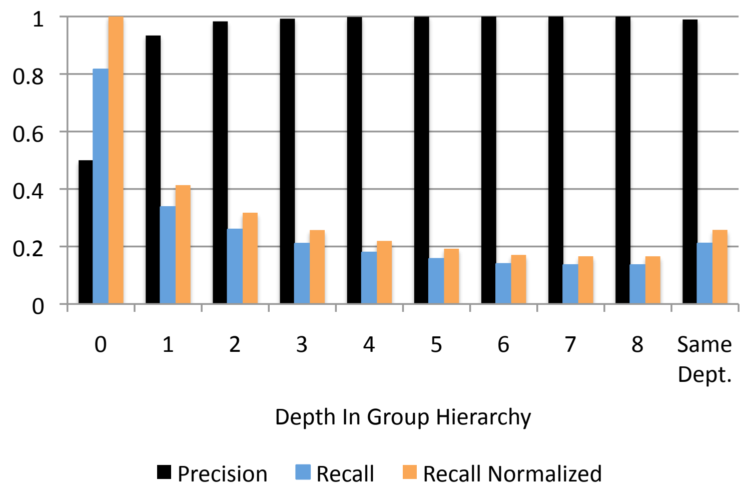


Figure 6.4: Group predictive power for first accesses (Data set A). Collaborative groups were trained using the first 6 days of the log; precision and recall were tested using the seventh day.

the groups suggests that the process was successful. For example, we studied the department codes present from the users in each group. Figures 6.2 and 6.3 show the department codes present in two of the 33 top-level groups. The first group clearly contains users who work in the Cancer Center, and the second group contains users who work in psychiatric care.

Interestingly, department codes themselves do not directly coincide with collaborative groups. For example, the *Medical Students* department code appears in the psychiatric care collaborative group. This makes sense because certain medical students were rotating through psychiatric care during the week when our log was collected, and they accessed the associated patients' medical records. However, medical students change rotations on a regular basis. This indicates that it would be incorrect to consider all medical students as their own collaborative group. It also indicates that we must update the collaborative groups from time to time in order to capture dynamic collaboration patterns.

Our goal in extracting collaborative groups is to improve explanation recall (i.e., the number of accesses that can be explained). As a baseline, we could assign all users to a single group; doing this, we are able to explain the 75% of first accesses where the patient has an event (see Figure 5.7). However, this approach has the consequence of potentially providing spurious *false positive* explanations if two users are not actually part of a collaborative group.

To measure the trade-off between adding collaborative groups to improve recall and introducing false positives, we performed a simple experiment. We constructed a *fake* log that contains the same number of accesses as the real log. We generated each access in the fake log by selecting a user and a patient uniformly at random from the set of users and patients in the database. (Because the user-patient density in the log is so low, it is unlikely that we will generate many fake accesses that "look" real.) We then combined the real and fake logs, and evaluated the explanation templates on the combined log.

We define *recall* to be the proportion of real accesses returned by an explanation template from the set of all real accesses ($Recall = \frac{|Real\ Accesses\ Explained|}{|Real\ Log|}$). We define *precision* to be the proportion of real accesses that are in the set of all accesses returned ($Precision = \frac{|Real\ Accesses\ Explained|}{|Real+Fake\ Accesses\ Explained|}$). The *normalized recall* is the proportion of real accesses returned by an explanation template from the set of accesses we have information on

($Normalized\ Recall = \frac{|Real\ Accesses\ Explained|}{|Real\ Accesses\ With\ Events|}$). The normalized recall takes into account the fact we have a partial data set. In an ideal world, our explanation templates would observe precision and recall values close to 1.0.

We ended up with an 8-level hierarchy of collaborative groups, and we created the Groups table as described in Section 6.2.1. Using hand-crafted explanation templates that incorporate the groups (e.g., Example VI.2), we measured the precision, recall and normalized recall. Figure 6.4 shows the results for the groups at different levels of the hierarchy, measured using the first accesses from the seventh day of the log. Depth 0 refers to the naive approach of placing every user in a single group. Additionally, we included the hand-crafted explanation template that captures the idea that a user accesses a medical record because another user with the same department code has an appointment, visit or produced a document with the patient (e.g., explanation (B) from Example V.2).

As expected, the top-level groups in the hierarchy (depths 0 and 1) result in higher recall, but lower precision. On the seventh day, the depth 0 group explains 81% of the first accesses. We also found that explanations based on collaborative groups outperformed explanations based on department codes, because users from different departments (e.g., *Pediatrics and Nursing-Pediatrics*) often work together.

In practice, depth 1 collaborative groups appear to strike a reasonable balance of high precision (>90%) and improved recall. For day seven in the log, if we consider explanations based on appointments, visits, documents produced, and repeat accesses (e.g., Figure 5.6), and we also include collaborative groups at depth 1, we are able to explain over 94% of all accesses.

6.3 Diagnosis Responsibility Information

Another common type of missing data is information describing the diagnoses each department is responsible for treating. EMR databases record ICD-9 codes for billing purposes that detail each patient's diagnoses. However, this information currently cannot be used to explain why accesses occur. Our hypothesis is that specific hospital employees are *responsible* for treating a diagnosis. Moreover, instead of acting independently, employees treat patients collaboratively. We leverage these observations to explain why accesses to medical records occur. That is, if the auditing system is confident that an employee is responsible for treating a diagnosis, then the associated accesses are appropriate. For example, consider the patient Alice who requires chemotherapy, and Dr. Carl who works in the Hematology/Oncology (Hem/Onc) Department. The auditing system can explain why Dr. Carl accessed Alice's medical record: *Dr. Carl accessed Alice's medical record because Hem/Onc employees are responsible for chemotherapy patients.*

The challenge is then to determine which employees are responsible for each diagnosis and to quantify the confidence in this decision. The first part of this section examines this problem when departments treat a diagnosis, and the second part considers clustering similarly treated diagnoses.

6.3.1 Method

Given a diagnosis, the objective is to determine the set of employees that are responsible for treating it. While it is possible to determine, at the granularity of individual employees, who is responsible for a diagnosis, we observe several limitations with this fine granularity. First, many employees treat the same diagnosis. For example, another doctor in the Hem/Onc Department could also adequately treat Alice. Second, hospitals typically schedule employees in shifts and rotations, where different employees are responsible for the same patients at different times. Therefore, analyzing treatment behavior at the granularity of individual employees suffers from noise.

Instead, we determine diagnosis responsibility information at the granularity of hospital-designated departments. Intuitively, these departments serve as a grouping of employees with similar responsibilities. Moreover, we can assume these groupings are relatively accurate since EMR administrators manage them. Interestingly, the specificity of the diagnoses a department treats varies greatly. In some cases, like the Hem/Onc Department, employees are responsible for specific types of diagnoses. In contrast, departments such as the Central Staffing Department (which contains nurses who rotate throughout the hospital based on need) treat a wide range of diagnoses. Hospital-designated departments are limited because they cannot be used to detect fine-grained inappropriate behavior such as when one employee from a department should be able to access a patient's record while another employee in the same department should not. However, given that employees in the same department often speak about patients during rounds, this level of abstraction seems to be a reasonable compromise between the typical flow of information in a hospital and the expressivity of security policies. While previous work has used access patterns to construct collaborative groups to more accurately capture working relationships [17], these hospital-designated departments provide a natural starting point for defining employee groups that are interpretable and effective.

The objective can then be restated as: Given a diagnosis, find the set of departments that are responsible for treating it. A simple metric is the likelihood that an employee from a department accesses the medical records of patients with the given diagnosis. The intuition is that if a department is responsible for a diagnosis, then it is highly likely that someone from the department will access the associated patients' records. For example, if we consider a chemotherapy patient, it is more likely that the Hem/Onc Department will access the patient's record than the Dermatology Department.

Definition VI.3 (Access Probability)**.** The access probability for a diagnosis c and a department d is defined as the probability that an employee in department d will access a patient's medical record, given that the patient has diagnosis c:

$$AP(\text{Department d}|\text{Diagnosis c}) = \frac{|\text{Patients with diagnosis c accessed by department d }|}{|\text{Patients with diagnosis c}|}$$

One might consider using the access probability alone to conclude that a department d is responsible for diagnosis c. (I.e., if the access probability is greater than a given threshold, the auditing system would conclude that all accesses by employees of department d to the records of patients having diagnosis c are appropriate.) However, this ignores an important problem: accesses by employees in general departments (e.g., Central Staffing) would nearly always be deemed appropriate because employees in these departments are involved in broad aspects of patient care. While it is likely that someone from Central Staffing accesses each chemotherapy patient's record, it would be a mistake to conclude that Central Staffing is *responsible* for treating chemotherapy patients.

Observing these limitations, an additional metric is required to differentiate auxiliary accesses from accesses of employees that are responsible for treating a diagnosis. To this end, the *treatment probability* measures how likely it is that a patient has a specific diagnosis when employees from a given department access his medical record. The basic assumption here is that departments responsible for a diagnosis will have higher treatment probabilities because these departments treat a smaller and more narrow set of diagnoses than auxiliary departments (e.g., Hem/Onc employees treat a smaller set of diagnoses than Central Staffing employees).

Definition VI.4 (Treatment Probability)**.** The treatment probability for a diagnosis c and department d is defined as the probability that an employee in department d is accessing a patient's medical record to treat diagnosis c:

$$TP(\text{Diagnosis c}|\text{Department d}) = \frac{|\text{Patients with diagnosis c accessed by department d }|}{|\text{Patients accessed by department d}|}$$

A higher treatment probability indicates that the department is responsible for the diagnosis. For example, the treatment probability for the Hem/Onc Department to access a chemotherapy patient's medical record is much higher than the Central Staffing Department, even though Central Staffing employees frequently access chemotherapy patients.

The auditing system combines the probabilities to determine if a department is responsible for a diagnosis. Specifically if the access and treatment probabilities are above a threshold, the department is *responsible* for the diagnosis. Here, a responsible department implies that the department is specifically tasked to treat the diagnosis. It is important to note that while general diagnoses (e.g., hypertension) are treated throughout the hospital, they are often not the responsibility of specific departments.

Ideally, a single pair of thresholds could be set to determine the responsible departments for all diagnoses. However, because the probabilities for each diagnosis have their own means and variances, responsibility information can be missed. For example, given TP(Chemotherapy | Hem/Onc) = 0.12, TP(Chemotherapy | Central Staffing) = 0.05, and TP(Strep Throat| Pediatrics) = 0.01, selecting a single threshold such that the Hem/Onc Department, but not Central Staffing, is responsible for chemotherapy, while the Pediatrics Department is responsible for strep throat, is not possible. Instead, the probabilities are normalized (denoted AP_n and TP_n) to a mean of zero and unit variance. [1] Thresholding the normalized access and treatment probabilities allows the auditing system to find responsible departments

[1]The access probability is normalized as follows: Let $V_c = \{AP(d_1|c), \ldots, AP(d_N|c)\}$, m = mean($V_c$), std = stdev($V_c$), then Normalized($V_c$) = $\{(AP(d_1|c) - m)/std, \ldots, (AP(d_N|c) - m)/std\}$.

across all diagnoses with a single pair of thresholds. Moreover, thresholding the normalized probabilities provides for a relative comparison between the departments, so that responsible departments can be found even if the probabilities are small. As a result, this methodology finds responsible departments that have relatively larger access and treatment probabilities than other departments in the hospital.

More formally, responsibility is defined as follows:

Definition VI.5 (Responsibility)**.** Given a diagnosis c, department d is *responsible* for the diagnosis if both of the following are true:

- The department frequently treats the diagnosis: $AP_n(d \mid c) \geq s$
- The department is directly involved in the diagnosis' treatment: $TP_n(c \mid d) \geq t$

Where s and t are thresholds specified by the compliance officer.

Recall that the explanation-based auditing system is provided with a set of reasons for access as input. For our purposes, the set of reasons that are input to the auditing system include departmental responsibility information and direct explanations from the EMR database (e.g., patient appointments, visits, medications, etc.) [29]. Alternative clinical and operational reasons are possible and we plan to study them in future work.

Definition VI.6 (Reasons For Access)**.** Consider an audit log entry describing the employee that accessed the medical record and the patient whose record was accessed. The patient is diagnosed with a set of diagnoses C. The EMR database also stores direct explanations using appointment, visit and medication information, among others. The access is appropriate if one of the following is true:

- The employee is assigned to a department that is responsible for treating *one* of the patient's diagnoses.

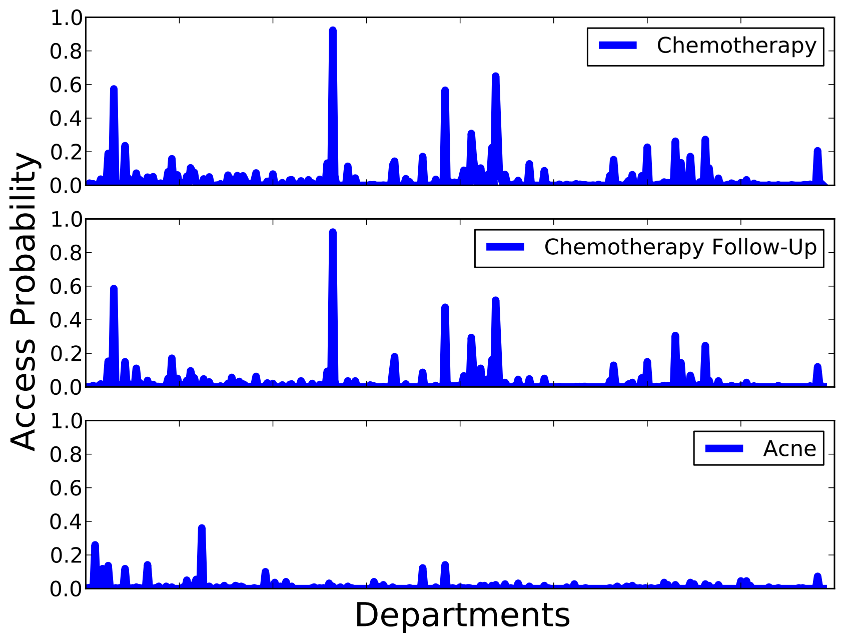


Figure 6.5: Access patterns for the diagnoses *chemotherapy* and *follow-up chemotherapy* are highly correlated compared to the access patterns for *acne*. The most likely department to access chemotherapy patients is the Cancer Center, while the most likely department for acne patients is Dermatology.

- The EMR database stores direct explanations describing the reason for access.

6.3.2 Overfitting Diagnoses

As described above, the auditing system attempts to determine which departments are responsible for a given diagnosis. However, in many cases, multiple diagnoses (with unique ICD-9 codes) require the same or similar treatment. For example, "encounter for antineoplastic chemotherapy" and "follow-up examination following chemotherapy" (with ICD-9 codes V58.11 and V67.2, respectively) require similar treatments. Due to these similarities, the auditing system can potentially overfit the responsibility information.

Our observation is that if diagnoses with similar treatments are clustered, and the access and treatment probabilities are calculated using all diagnoses in the cluster, then the resulting responsibility information is more representative of actual clinical processes. More formally, we construct diagnosis clusters in the following manner: Let the patient vector V_p be a boolean vector of length N, where N is the number of departments. Index i is set to 1 (i.e., $V_p[i] = 1$) if an employee from department i accessed patient p's medical record, and 0 otherwise. A diagnosis vector V_c is the average of all patient vectors where the patient has diagnosis c. The resulting values represent the access probabilities for each department. We say two diagnoses c_i and c_j are treated similarly if their vectors' Pearson's correlation coefficient is greater than a provided threshold. More formally, two diagnoses are clustered if $corr(V_{c_i}, V_{c_j}) \geq v$, where $v = 1.0$ implies no clustering and smaller values of v increase the number of diagnoses clustered. As Figure 6.5 shows, chemotherapy and follow-up chemotherapy have highly correlated treatments (Pearson's correlation coefficient of 0.97) compared to acne.

6.3.3 Results & Discussion

For the evaluation, we divide the patients for which we have diagnoses into two sets: a **training set** and a **testing set**. The training set and testing set each contain approximately 29K patients and 200K real and fake first accesses to these patients' medical records. The access and treatment probabilities are calculated using only real first accesses for patients in the training set to determine the responsible departments. We only consider diagnoses that occur at least 30 times. The reasons for access are then evaluated on real and fake first accesses for patients in the testing set to calculate the recall, precision and F-measure.

The first question we aim to answer is whether the auditing system produces semantically correct responsibility information for a given diagnosis. Table 6.1 shows the system's output for selected diagnoses and departments. For example, the departments that are likely to treat patients who require a kidney transplant are the

Chemotherapy			Kidney Transplant		
Department	AP_N (AP)	TP_n (TP)	**Department**	AP_n (AP)	TP_n (TP)
Cancer Center	9.19 (0.91)	1.00 (0.12)	Transplant	8.14 (0.56)	5.61 (0.20)
Pharmacy	6.52 (0.66)	1.38 (0.16)	Nephrology	6.21 (0.43)	6.22 (0.23)
Central Staff.	5.54 (0.56)	0.24 (0.06)	Pharmacy	2.78 (0.21)	0.20 (0.02)
Hem/Onc	5.54 (0.56)	2.07 (0.21)	Dialysis	1.11 (0.10)	2.91 (0.11)
Nursing-Hem/Onc	2.98 (0.32)	2.20 (0.23)	Nursing-Renal	1.02 (0.09)	2.04 (0.08)

Table 6.1: Normalized access and treatment probabilities for selected departments and diagnoses. The probabilities are normalized to a zero mean and unit variance so that values are comparable across diagnoses.

Strep Throat		
Department	AP_N (AP)	TP_n (TP)
Pediatrics	14.04 (0.64)	6.25 (0.019)
Central Staff.	4.41 (0.21)	0.94 (0.004)
Health Cntr	3.17 (0.11)	2.69 (0.009)
Nursing-ER	1.30 (0.07)	1.75 (0.006)
Family Med.	0.99 (0.05)	0.27 (0.002)

Table 6.2: Normalized access and treatment probabilities (continued).

Nephrology Department and Transplant Center. The Cancer Center and Hem/Onc Department are responsible for chemotherapy patients. In contrast, even though the Central Staffing Department accesses a majority of chemotherapy patients' records, the normalized treatment probability is low because the department treats a wide range of diagnoses.

Next, we analyze the impact of the access and treatment probability thresholds on precision and recall for a few selected diagnoses. Figure 6.7 shows the results when accesses are explained using the responsibility information for a given diagnosis, without direct explanations, as the normalized treatment probability threshold is varied and the normalized access probability threshold is held constant. For these figures, recall for a single diagnosis is 1.0 when all accesses for patients with the diagnosis are explained, and recall increases as more departments are determined to be responsible for a diagnosis. As expected, higher treatment thresholds produce better precision at the cost of lower recall. The precision drops to 60% for low treatment probability thresholds, but plateaus because of the minimum access probability required (i.e., Figure 6.7). Interestingly, general diagnoses such as hypertension have few if not any

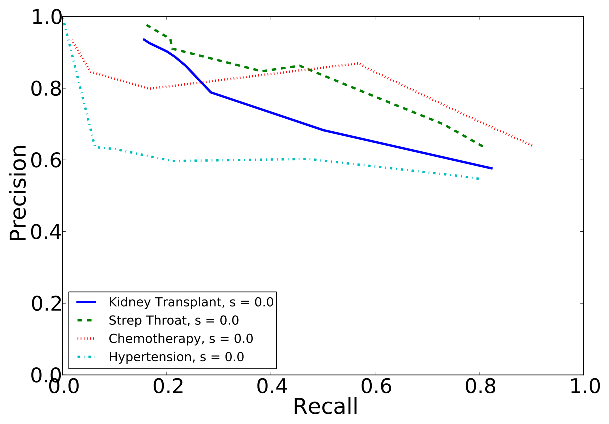


Figure 6.6: Precision vs recall for specific diagnoses (normalized thresholds: s = 0.0 and varied t).

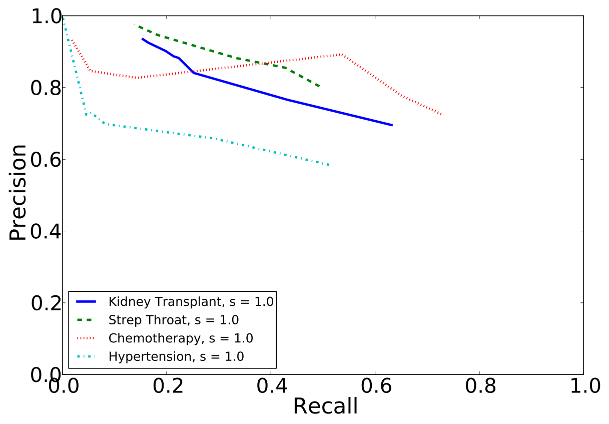


Figure 6.7: Precision vs Recall for specific diagnoses (normalized thresholds: s = 1.0 and varied t).

responsible departments due to their low normalized treatment probabilities since these diagnoses occur across every department. For example, hypertension has no responsible departments for thresholds larger than s=1.0 and t=2.0, which produces a recall of zero for the diagnosis (i.e., Figure 6.7). As a result, for higher thresholds, general diagnoses are not as effective at explaining why accesses occur compared to more specific diagnoses such as kidney transplant. Future work is needed to explain accesses for diagnoses that are encountered throughout the hospital.

Table 6.3 shows the recall, precision and F-measure for various thresholds when all reasons for access are aggregated (i.e., the set of accesses explained by the dis-

Description	**s**	**t**	**v**	**Recall**	**Precision**	**F-Measure**
Evidence + Responsibility	1.0	1.0	1.0	0.534	0.830	0.650
Evidence + Responsibility + Clustering	1.0	1.0	0.95	0.540	0.842	0.658
Evidence + Responsibility + Clustering	1.0	1.0	0.9	0.529	0.856	0.654
Evidence + Responsibility	1.0	2.0	1.0	0.411	0.904	0.565
Evidence + Responsibility + Clustering	1.0	2.0	0.95	0.419	0.915	0.574
Evidence + Responsibility + Clustering	1.0	2.0	0.9	0.411	0.920	0.568
Evidence + Responsibility	1.0	3.0	1.0	0.344	0.945	0.504
Evidence + Responsibility + Clustering	1.0	3.0	0.95	0.353	0.953	0.515
Evidence + Responsibility + Clustering	1.0	3.0	0.9	0.353	0.957	0.515
Evidence Only	- - -	- - -	- - -	0.229	**0.997**	0.372

Table 6.3: Aggregate results for all diagnoses with varying thresholds (s = Normalized access probability threshold, t = Normalized treatment probability threshold, v = Clustering threshold).

junction of reasons). As shown in previous work, direct explanations have a recall of 22% with near perfect precision [29]. The high precision is due to the sparsity of employee-patient relationships, which makes it is unlikely that a fake access corresponds to actual evidence. As responsibility information is included, two times as many first accesses are explained at the cost of lower precision (e.g., 41% recall with 90% precision for $s = 1.0$ and $t = 2.0$). Figure 6.8 shows the trend from this table without the clustering data (the thresholds are decreasing from left to right). In addition to the recall, precision and F-measure, Table 6.4 further breaks down the results by the number of real and fake accesses that are explained and unexplained, respectively. The responsibility information introduces a drop in precision because it is at the granularity of a department instead of an individual employee. Clustering similarly treated diagnoses focuses the auditing system on the departments that are responsible for a diagnosis and provides a slight improvement in precision. For example, after clustering, various nursing departments are no longer thought to be responsible for chemotherapy. In summary, the explanation-based auditing system filters a large subset of the first accesses and over 94% of all accesses (when repeat accesses are filtered).

Lastly, Table 6.5 lists which departments account for the largest proportion of

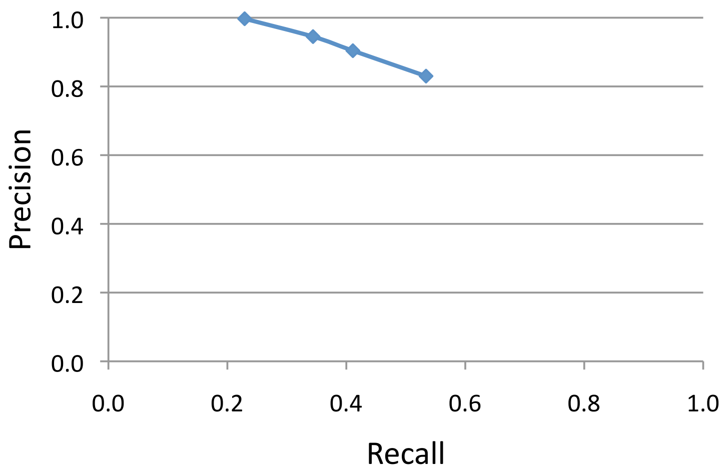


Figure 6.8: Precision vs recall for all diagnoses at various thresholds from Table 6.3 (thresholds decrease from left to right).

	Explained	**Unexplained**
Real First Accesses	80K	115K
Fake First Accesses	8K	187K

Table 6.4: Confusion matrix (for $s = 1.0, t = 2.0, v = 1.0$) shows the breakdown of first accesses explained.

unexplained first accesses, and the rate at which a department's accesses are unexplained. As expected, departments that provide wide ranges of services throughout the hospital (e.g., Radiology and Central Staffing) are often unexplained, even though these general departments access most patient's medical records. In contrast, departments that provide narrow sets of services such as the Allergy, Neonatology, Dermatology, Nephrology and Thoracic Surgery Departments have unexplained rates that are equal to or less than 0.05. This is not to say that Central Staffing employees are more likely to inappropriately access records. However, we cannot use diagnosis information to explain why their accesses occur. Future work is needed to explore alternative types of explanations to remedy this gap in the auditing system for general departments.

Department	**Unexplained Rate**	**Proportion of All Unexplained Accesses**
Cancer Center	0.78	0.08
Radiology	0.90	0.07
Central Staffing	0.86	0.05
Physician Services	1.00	0.04
General Medicine	0.21	0.04
Outpatient Services	0.82	0.04
Pharmacy	0.88	0.04
Health Info. Management	0.98	0.04

Table 6.5: Departments with the most unexplained first accesses (for $s = 1.0$, $t = 2.0$, $v = 1.0$).

CHAPTER VII

Ordering Accesses By Suspiciousness

7.1 Overview

As the previous chapters demonstrated, the explanation-based auditing system is able to filter a large number of the appropriate accesses from the audit log. Unfortunately, it is still impractical for compliance officers to review the remaining suspicious accesses manually (e.g., 5% of 4M accesses is 200K accesses). In practice, compliance officers allocate an *auditing budget*, which specifies the number of accesses (or the amount of time) that can be reviewed in a given period of time. Ideally, the compliance officers would find all inappropriate accesses by the time they have exhausted their budget.

There are many possible strategies for selecting which access the compliance officers should review next. In the simplest case, compliance officers could be presented with the suspicious accesses in temporal order. If an inappropriate access occurs at the beginning, then the access will be detected. Otherwise the breach will go undetected, putting personal health information at risk. Instead, of this simple approach, we argue that the auditing system should order the suspicious accesses such that the most suspicious access are reviewed first, resulting in the best utilization of the auditing budget.

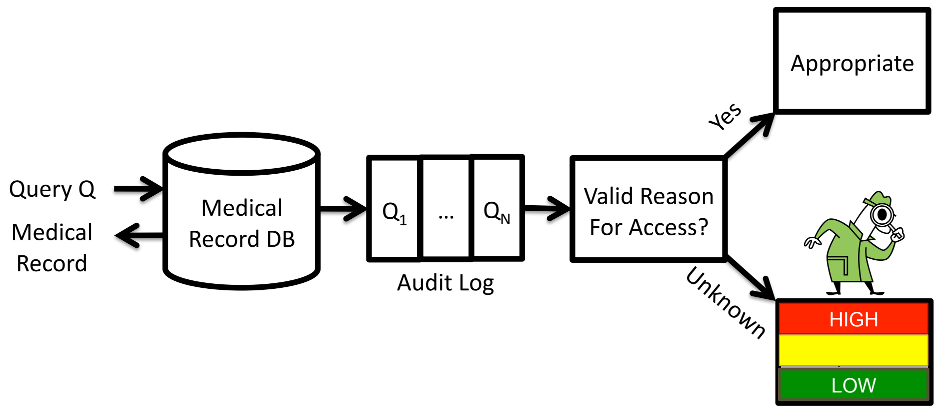


Figure 7.1: Accesses are ordered by their suspiciousness so that compliance officers review the most suspicious accesses first.

This chapter considers the problem of ordering suspicious accesses from the audit log by their *suspiciousness* (Figure 7.1). We define what it means for an access to be suspicious and its associated metrics. The main observation is that *accesses to medical records are not temporally independent, but rather are temporally dependent on other events occurring in the hospital system* such as medication orders, appointments or even other employees' accesses. For example, a pharmacist is more likely to access a patient's record after a medication order is placed. Using this observation, we then formulate a standard machine learning problem where the inputs are the events occurring in the hospital and the output is the probability that an employee should access the patient's record at a specified time. This *probability for access* is then used to order the suspicious accesses for review. Lastly, we measure the quality of the ordering using data from the University of Michigan Health System.

7.2 Access Suspiciousness

Given a suspicious access in the audit log (that cannot be explained with a clinical or operational reason), the objective is to determine its suspiciousness. The suspiciousness of an access should satisfy the following desired properties.

- **Interpretable:** The metrics associated with suspiciousness should be easily understandable by both compliance officers and patients, and the reasons for why the risk is high or low should be easily associated with the access.
- **Comparable:** Given two accesses, the auditing system should be able to determine which has a higher risk of being inappropriate.
- **Easily Computable:** Given the large number of accesses in the audit log, determining the suspiciousness of an access should require simple computation.

One way to think about the suspiciousness of an access is: *Should the employee access the patient's medical record (at any time)?* Previous work found the clinical or operational reason why an employee accessed a patient's record [27], and therefore implicitly assumed that the employee could access that patient's record at any time (within a month-long time range of an appointment, for example). Unfortunately, not all accesses can be explained with a clinical or operational reason. In particular, broadly functional departments (e.g., pharmacy) disproportionally suffer from unexplained accesses because there is no specific connection between pharmacists and patients.

Instead of considering if an employee should access a patient's record at any time, we consider the question: *Should the employee access the patient's medical record at a given time t?* We consider this temporal aspect for two main reasons. First, considering if an access should or should not occur at a specific time allows for a fine-grained analysis of an employee's accesses. Second, accesses are often temporally dependent on other events occurring throughout the hospital, which can be leveraged to determine when accesses are expected.

This chapter leverages a probabilistic model to determine if an employee should access a patient's record at a specific time. This model is used because patient care is complex and dynamic (meaning that it is difficult to predict when an employee will access a patient's medical record). For example, a pharmacist may access a patient's record a few hours later than expected if many medication orders are placed simultaneously. The probabilistic model is preferred over a boolean approach that predicts if an access will occur or not at a given time because of the ambiguity in the problem. Interestingly, this probabilistic model naturally allows for an interpretable, comparable and easily computable metric that can be used to compute the suspiciousness of an access because each access is associated with a probability. More formally, suspiciousness is defined as follows.

Definition VII.1. The suspiciousness of an access by an employee to a patient's medical record at time t with respect to other events occurring in the hospital ($Events$) is defined as one minus the probability of access:

$$Suspiciousness = 1 - P(access \mid employee,\ patient,\ t,\ Events)$$

Suspiciousness ranges from zero to one where a value of one implies the highest risk. Accesses are ordered by suspiciousness in descending order.

Example VII.2. Consider the patient Alice who has cancer. Alice has an appointment, where an oncologist analyzes her medical record and creates a medication order. Later, a pharmacist accesses Alice's medical record to fulfill the prescription. Figure 7.2 shows a timeline of these events in the hospital as well as the expected probability of access of the pharmacist. In this example, the pharmacist's access at time 11 is more suspicious than the access at time eight because of its temporal relationship to the medication order.

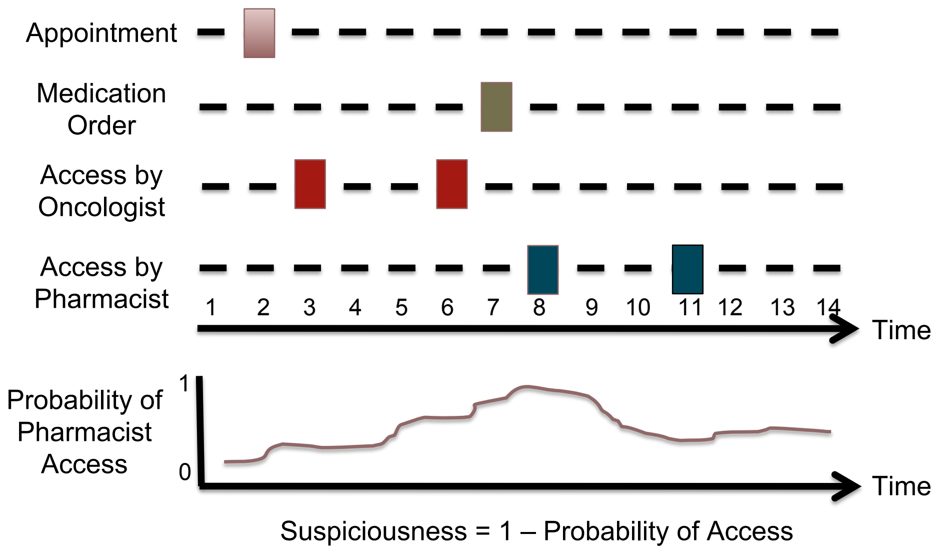


Figure 7.2: An example set of events (appointments, medications, accesses by oncologists and accesses by pharmacists) over time for a patient. Events are denoted by colored boxes. The auditing system determines the suspiciousness of the pharmacist's accesses for all possible times. Ideally, actual pharmacist accesses correspond to times with the lowest suspiciousness levels.

7.3 Method To Calculate Suspiciousness

This section presents one method to efficiently calculate the suspiciousness of an access. Recall that the main observation of this work is that employees' accesses are temporally dependent on other events occurring in the hospital. Consider the environment where all accesses and hospital events are binned by hour. In this setting, one natural method to predict if an access should occur at hour t is to look at the hospital events that also occur in that hour. For example, if a medication order is placed at hour t for a given patient, then the it is likely that the pharmacist will access the patient's record in the hour. Such a model can be represented with a linear logistic classifier. The inputs to the classifier are the events occurring at hour t and the output is the probability of access, which can be used to compute an access's suspiciousness.

One question concerning using such a classifier is: *At what granularity should the* *classifier be constructed?* At one extreme, a single classifier could be constructed per hospital employee to predict when his or her access will occur. This approach would require the construction of thousands of classifiers, which are likely over-fitted to the employee. In the opposite extreme, one classifier could be constructed for all employees. However, this approach does not take into account that each employee's accesses are best predicted by different hospital events (e.g., a pharmacist's accesses are predicted by medication orders, while a radiologist's accesses are predicted by x-ray orders). Observing these extremes, this work builds one classifier per hospital department (Pharmacy, Radiology, Central Staffing, etc.), under the assumption that employees assigned to the same department perform similar tasks and their accesses are dependent on similar types of events. However, it is important to note that employees within departments may have different responsibilities and this department-level model may miss these fine-grained differences.

The machine learning problem to calculate suspiciousness is formulated as follows.

- For each department D, a single classifier is constructed, C_D. A matrix is constructed as input to the classifier as shown in Figure 7.3.
- The columns (or features) of the matrix represent the set of hospital events: such as employee accesses grouped by department (i.e., oncologists' access to a patient's medical record), appointments, medications, and many others. Additionally, features are added to record which department is *associated with* an event (e.g., an oncologist had an appointment with the patient). We initially considered adding one feature per (event-type, department) pair, but this approach added too many features. Instead one feature is added per department, which records which departments are involved at the given time, but not which specific event they are associated with if multiple events occur in the same hour.

The event corresponding to accesses by the department that is being predicted is excluded from the input matrix.

- The rows of the matrix represent a specific hour in time for a given patient. If there are F features, P patients and H hours to analyze (for example, one week has 24 $hours/day \times 7$ $days/week = 168$ $hours/week$), then the matrix is of size $P \times H$ rows by F columns. Row i in the input matrix corresponds to patient $\lfloor i/P \rfloor$ at hour i % H. (where % is the modulus operator).

- Each entry in the matrix stores a boolean value, where the value is one if the event occurred at the given time for the specific patient, and zero otherwise.

- The output of the classifier is a size $P \times H$ vector that specifies the probability of access for any employee in the classifier's department at the given time for the specific patient.

It important to note that the resulting matrix is sparse (i.e., contains mostly zeroes) and therefore optimizations allow for compressed storage of the matrix.

Given this problem formulation, standard techniques can be used to train the classifier. The same input matrix is used as described above, but each entry in the output vector is a boolean value depending on if an access did or did not occur at the given time by the department for the specific patient. It is easy to extract a logistic classifier's significant features by analyzing each feature's weight.

After one classifier is trained for each department (there are approximately 300 departments at the University of Michigan Health System), they are used to order suspicious accesses. That is, for each hour t when a suspicious access occurs, the employee's department's classifier is retrieved and used to determine the probability of access (and its suspiciousness). This testing procedure is performed for all suspi-

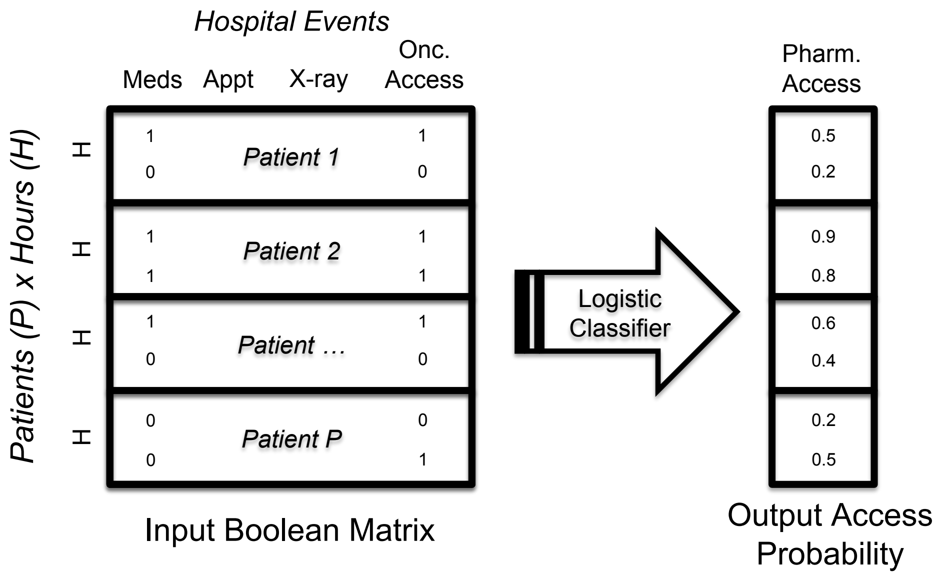


Figure 7.3: Overview of the machine learning problem to predict when pharmacists will access medical records. A boolean matrix describes which events occur in the hospital such as, but not limited to, medication orders, x-ray orders, appointments and oncology department accesses for a given patient at a specific time (each row corresponds to one hour in time). Given this input, the logistic classifier can predict the probability of access at each point in time.

cious accesses, but does not need to be performed for other times (which reduces the size of the input matrix compared to the training phase because the training phase analyzes times when accesses do not occur).

Obviously, the above solution is limited because it only considers events that occur at the hour t to predict if an access should also occur at that time. However, as noted previously, clinical care is complex and therefore predicting exactly when an access will occur is difficult. Moreover, events that predict when an access will occur may occur earlier or later than the access. Therefore, to manage this ambiguity, the input set of features is expanded to include those events that occur at hours $t+1, ..., t+W, t-1, ..., t-W$ where W is a pre-specified window size. As a result, medications at hour $t-1$ can be incorporated into the classifier to increase the probability of a pharmacist's access at time t. Due to this feature expansion, the number of columns in the input matrix increases from F to $F \times (2W + 1)$.

There are several limitations of the problem formulation as it is specified. First, the model considers boolean input features rather than the frequency in which an event occurs at a specific time. Second, all employees are grouped by their department, which then makes it impossible to distinguish between multiple employees' accesses that work in the same department. However, even with these limitations, this simple problem formulation is a natural starting point.

7.4 Selecting The Next Access To Audit

In the basic case, the auditing system would analyze all suspicious accesses, order them by suspiciousness and then present them to compliance officers (referred to as the *all-accesses approach*). However, this approach does not take into account the relationship between an employee's many accesses. That is, if the auditing system knows an employee previously accessed a patient's record appropriately, then other accesses to the same patient are more likely to be also appropriate (relative to the situation where there is no prior information about the employee's access). However, the all-accesses approach treats each access independently and does not consider these relationships.

Instead, the auditing system can order accesses in two different ways to better utilize the auditing budget. First, once the compliance officer determines that one of an employee's accesses are appropriate for a given patient, then the remaining accesses by the employee to the patient's record no longer need to be reviewed and can be removed from the list (here the assumption is that if one access is appropriate, then all accesses are appropriate). Second, given the most suspicious access, we argue that the auditing system should not present the compliance officer with that specific access; rather the auditing should select the least suspicious access from the set of accesses made by the same employee to the same patient's record. The idea is that this *best-of-the-worst* access will allow the compliance officer to more quickly determine if the accesses is appropriate or not, because this access will be better associated with events in the hospital than the most suspicious access.

It is important to note that this best-of-the-worst approach can be abused. For example, if a curious employee accesses a patient's record every hour, then it is likely that one of the many accesses will appear to be appropriate. Therefore, the compliance officer must be mindful of the number of accesses an employee makes when determining if an access is appropriate or not.

7.5 Evaluation Methodology

The objective of this work is to order accesses such that the most suspicious accesses are reviewed first. Ideally, labeled (ordered) accesses would be available to test and evaluate the methods presented in the work. Unfortunately, ground truth is difficult to acquire in the medical domain because of the large number of accesses that are logged and the compliance officers's limited time to label data. Instead, simplifying assumptions are required to analyze the quality of the auditing system's ordering. Specifically, *the quality of the ordering is measured by analyzing how well the models predict when an access will occur as a proxy for ordering accesses by their suspiciousness*. (This evaluation attempts to predict when all accesses in the log occur, but other approaches are possible that predict when an employee's first access to a patient's record will occur.) The assumption here is that if an access occurs when the system predicts one will occur, then the access is likely appropriate. In contrast, if an access occurs when no access is predicted, then it is suspicious and should be reviewed first.

In this setting, one pair of metrics that can capture the quality of the ordering system is sensitivity and specificity. *Sensitivity* (or recall) represents the auditing system's ability to predict when an access will occur. *Specificity* represents the auditing system's ability to predict when an access will not occur. Let a *true positive* be an hour where an access occurs and the probability of access is greater than a threshold τ, a *false positive* be an hour where an access does not occur and the probability of access is greater than a threshold τ, a *true negative* be an hour where an access does not occur and the probability of access is less than a threshold τ, and a *false negative* be an hour where an access occurs and the probability of access is less than a threshold τ.

Definition VII.3. The sensitivity and specificity of the auditing system are defined as:

$$\begin{aligned} \textit{Sensitivity} &= \frac{\#\ \textit{True Positives}}{\#\ \textit{True Positives} + \#\ \textit{False Negatives}} \\ \textit{Specificity} &= \frac{\#\ \textit{True Negatives}}{\#\ \textit{True Negatives} + \#\ \textit{False Positives}} \end{aligned}$$

As the threshold τ is varied, there is a trade-off between the number of accesses that are predicted and the accuracy of the prediction, which can be measured using ROC curves. Specifically, the plots compare the *true positive rate* (or sensitivity) against the *false positive rate* (or one minus the specificity). The area under the curve (AUC) allows for easy comparison between classifiers.

For this study, 40K patients were randomly selected and evenly divided into testing and training sets. For each department, a classifier was trained as described in Section 7.3 using a week of accesses to patients' medical records from the training set. Accesses and other hospital events were binned by hour so that each patient

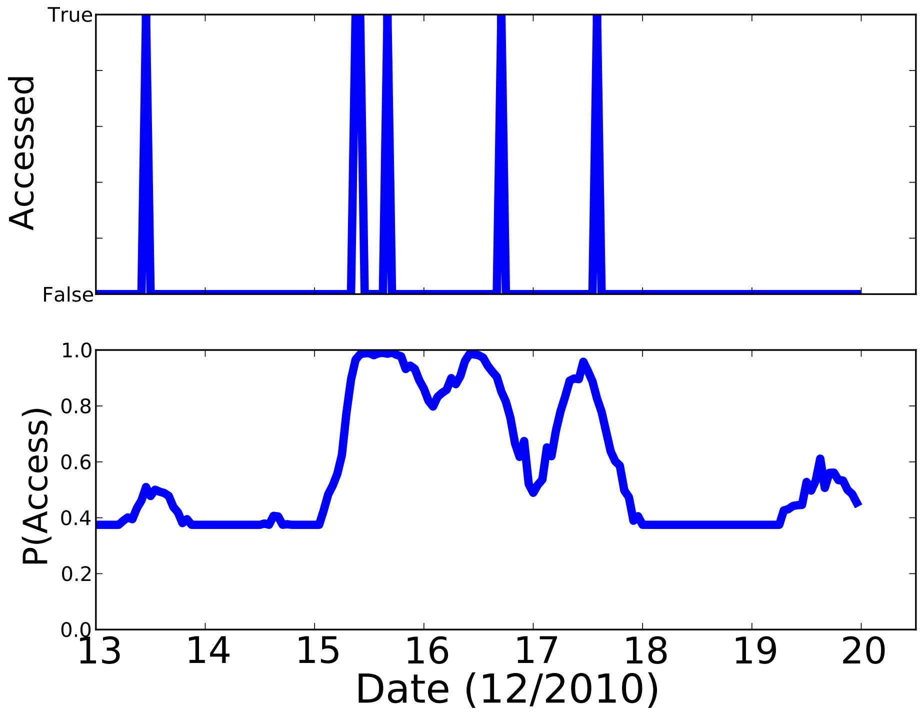


Figure 7.4: Accesses to a single patient's medical record by central staffing employees over time along with the probability of access at each hour.

accounted for 168 rows in each input matrix (the window size W was set to 8). Next, the classifiers were used to determine the probability of access (and suspiciousness) at each hour in the week for the patient's in the testing set. These probabilities and the actual accesses that occurred were then compared.

7.6 Results

First, we analyzed if the models described in Section 7.3 accurately predict accesses. Figure 7.4 and Figure 7.5 show the (i) history of accesses and (ii) probability of access for a single patient by central staffing and radiology employees, respectively. Interestingly, the probability of access is often high when accesses occur, supporting the hypothesis that accesses are temporally dependent on hospital events. However, false positives occur when the probability of access is high, but no access occurs (e.g., Figure 7.5), demonstrating the dynamics of patient care.

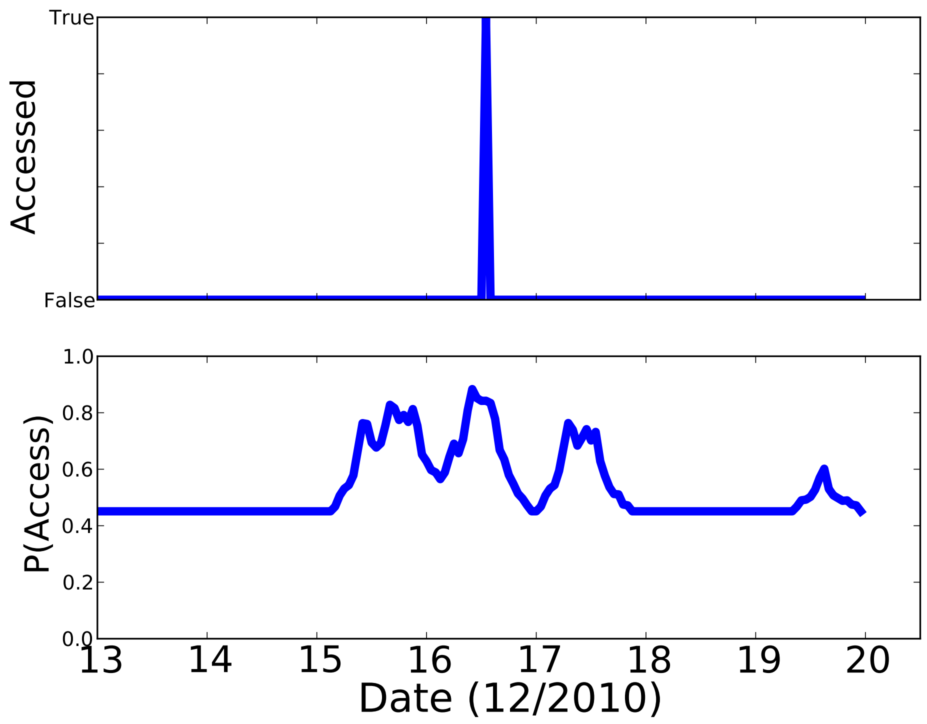


Figure 7.5: Accesses to a single patient's medical record by radiology employees over time along with the probability of access at each hour.

Next, we analyzed how well the system predicts accesses for all patients in the testing set. Figure 7.6 shows ROC curves for selected departments. Additionally, we compared the results to a *baseline* classifier that never predicts an access will occur. Interestingly, departments such as Pharmacy, Central Staffing and Radiology are accurately predicted and have area under the curve values of 0.88 and above. Table 7.1 lists the significant features of the departments' classifiers. For example, pharmacists' accesses at time t are best predicted by *orders that are placed* at the same, previous or later times. The Cancer Center had slightly poorer results (AUC = 0.80) because the department contains many employees with varying responsibilities. Cancer Center employeess' accesses are best predicted by appointments.

Lastly, we examined the quality of the predictions across all departments. To this end, the area under the curve (AUC) was measured for all departments that had at

Pharmacy	Radiology	Cancer Center
Order Entered at time t	Order Entered at time t	Appointment at time t
Order Entered at time t+1	Order Entered at time t-1	Appointment at time t+1
Order Entered at time t-1	Radiology Event Began at time t	Appointment at time t+2
Order Entered at time t+2	Radiology Event Updated at time t	Central Staffing Access at time t

Table 7.1: The highest-weighted features to predict accesses at time t for selected departments.

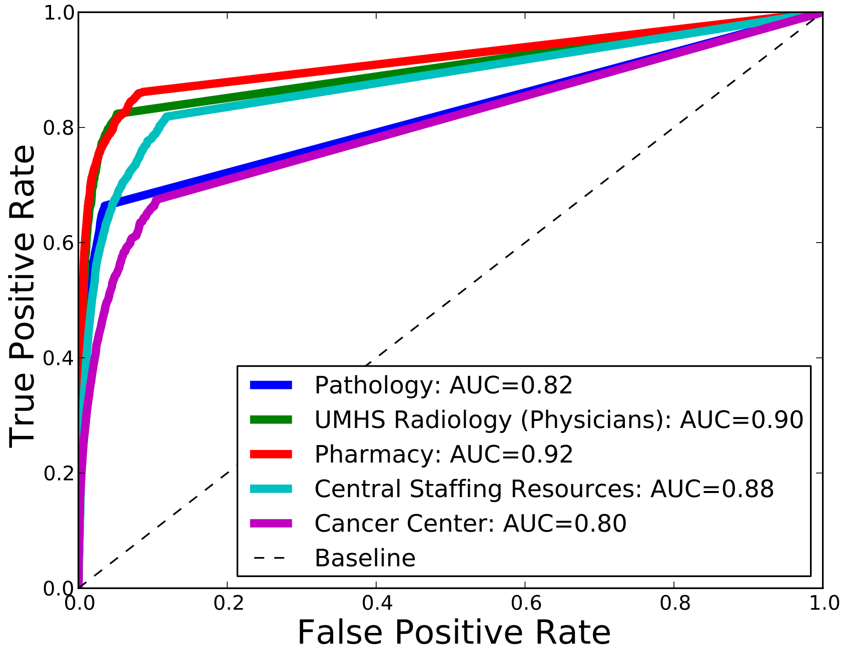


Figure 7.6: Accuracy of predicting when accesses will occur for selected departments.

least 10K accesses in the log (approximately 115 of the 300 departments that account for 86% of all accesses), which was done so the classifier had sufficient training data. Figure 7.7 shows the number of departments with given AUC values (binned in 0.05 ranges) as well as the cumulative distribution of AUC values. Interestingly, over 50% of the departments measured had AUC values greater than 0.75. However, some departments produced poor predictions. For example, the Physician Services Department had an AUC of 0.64, while the Information Technology (IT) Department had an AUC of 0.5. However, this last poor prediction is expected because IT accesses are not associated with patient care.

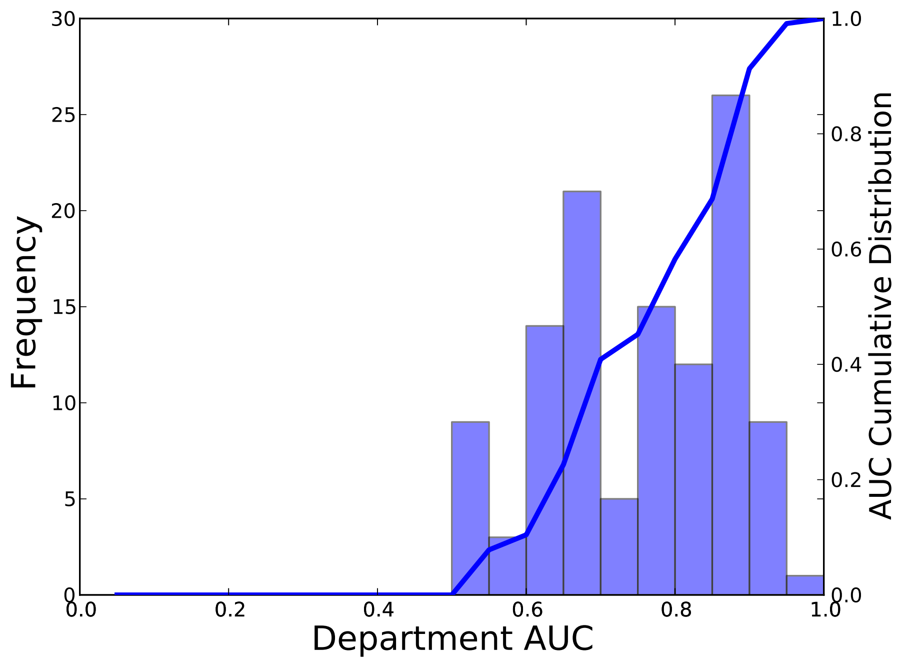


Figure 7.7: Distribution of the area under the curve (AUC) across departments.

7.7 Discussion

The auditing system's ability to accurately predict when accesses will occur for many departments implies that appropriate accesses are likely to be ranked lower than curious accesses. Therefore, compliance officers will review curious accesses first. For example, Figure 7.6 demonstrates that pharmacy employees' accesses are accurately predicted 80% of the time with less than a 10% chance of false positives. However, not all departments' accesses can be predicted accurately.

There are many possible directions for future work to improve the ordering quality. First, this work treats all accesses from a department equally. However, often employees within a department are responsible for different aspects of patient care at different stages during the treatment process (for example cancer patients are diagnosed, undergo surgery, and then are treated with chemotherapy). Moreover, the set of events that predict when an access will occur can vary as the patient's treatment proceeds. Therefore, the prediction can be improved if smaller functional groups can be discovered within departments that better capture these treatment variations. Second, the approaches presented in this work assumed a simple linear model where each hospital event is independent of each other; however, it may be possible to utilize conditional random fields or other similar approaches that take into account the dependency of events to determine the probability of access.

It is important to note that there are ways to abuse the auditing models presented here. First, the auditing system assumes the threat of a curious employee that simply accesses patient data. However, malicious employees that create fake appointments can trick the system to believe an access is appropriate. Therefore, future versions of the auditing should not only look at when events occur, but also who created the event (however, this also can be abused by malicious teams). That being said, discussions with the compliance office and recent Health and Human Services rules confirm the importance of threats by curious or snooping employees.

CHAPTER VIII

Prototype Auditing System

8.1 Introduction

Ensuring that hospital employees' appropriately use electronic medical records (EMR) is increasingly important. Recent legislation such as the Affordable Care Act, HITECH and HIPAA outlines guidelines for the appropriate use of patient data. To provide accountability, these regulations require that hospitals record all accesses to medical records in secure, tamper-proof audit logs, which can be retrospectively reviewed by compliance officers. While it is the compliance officer's responsibility to monitor these logs, manual analyses do not scale (e.g., the University of Michigan Health System records millions of accesses per week). Even analyzing accesses for the smaller subset of patients that file complaints is time-consuming and requires teams of tens of compliance officers.

One tool that can be used to improve compliance officers' efficiency is the explanation-based auditing system [27, 28]. The explanation-based auditing system improves compliance officers' efficiency by filtering appropriate accesses from the log that occur for clinical or operational reasons so compliance officers can focus their efforts on suspicious behavior. Previous studies demonstrated that over 90% of accesses in a hospital's audit log could be filtered. However, because the system was a research tool rather than a deployable application, various questions remained about the system's effectiveness. First, how much time does a compliance officer save when using the auditing tool compared to a manual analysis? Second, how accurately does the tool classify *actual* inappropriate and appropriate accesses? (Previous studies measured the accuracy of the system using randomly generated 'curious' accesses as a proxy for inappropriate behavior, rather than real inappropriate accesses because it is difficult to get labeled training data.)

To answer these questions, this chapter presents the design and development of the explanation-based auditing system prototype, which will be evaluated by the University of Michigan Health System's Compliance Office. The prototype is a single-node application that will be installed in the hospital's data center. Compliance officers are able to extract (de-identified) audit log data and patient information from the medical record system (e.g., Epic [26]), upload the data into the prototype, and then use the prototype to search for inappropriate behavior. The prototype allows for the analysis of the auditing system's effectiveness and accuracy using real hospital data and known instances of inappropriate use.

8.1.1 Contributions

The explanation-based auditing system prototype makes several contributions.

- **Integration With EMR Systems:** One of the major initial concerns about building the prototype was getting EMR data into the prototype. After working with the hospital's IT department, systems have been configured to extract *encounters* (e.g., appointments, medication orders, x-rays, etc.) and accesses for a given patient's medical record, dump the data to a text file, and then load the text file into the prototype.

- **Defining and Mining Explanations:** Once the data are loaded, explanations for accesses must be defined by compliance officers or mined directly from data. The prototype provides interpretable form-based web interfaces to create and mine clinical or operational reasons for access.

- **Exploring Accesses:** After explanations are defined, they can be used to filter appropriate accesses, leaving only a smaller number of suspicious accesses for review. A simple web interface allows compliance officers to search for accesses based on the patient, employee, or if there is a known reason for access. In addition, for each access that is explained by a clinical or operational reason, a text description can be attached to the access, which allows for easily interpretable access reports for both compliance officers and patients.

8.1.2 Demonstration

To demonstrate the effectiveness of the explanation-based auditing prototype, case studies are presented describing how compliance officers can use the auditing system. Specifically, this work discusses how the prototype system can be used to detect snooping employees, provide for proactive random audits of employee populations and quickly produce access reports for a patient that files a complaint.

8.2 System Design

The explanation-based auditing system prototype allows compliance officers to complete an end-to-end audit. That is, when a complaint is filed, compliance officers can quickly analyze the accesses in question to determine if an inappropriate access has occurred. There are numerous challenges in designing the prototype to allow for a complete auditing workflow such as organizing the underlying database to cleanly presenting information to compliance officers. This section outlines the

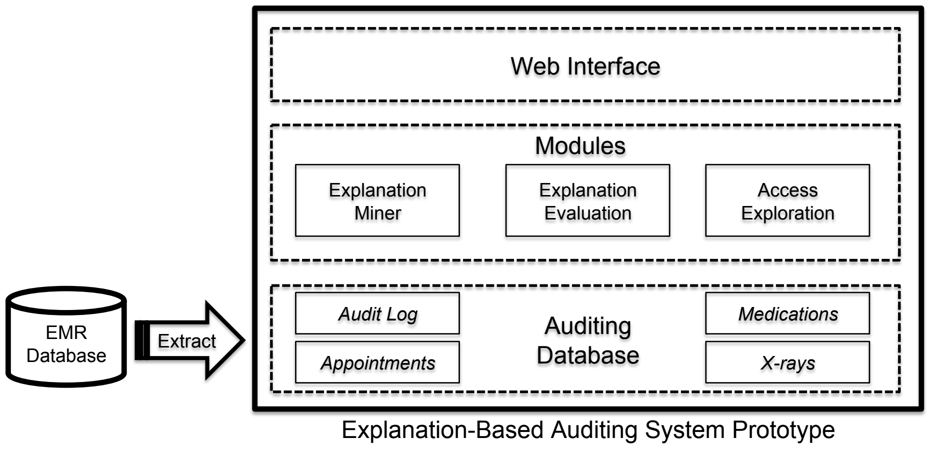


Figure 8.1: Overview of the prototype auditing system, which is composed of (1) a system to extract data from the EMR database, (2) a web interface to access the auditing tool, (3) modules to mine explanations, evaluate explanations and explore the results, and (4) a database to store the extracted EMR data and auditing results.

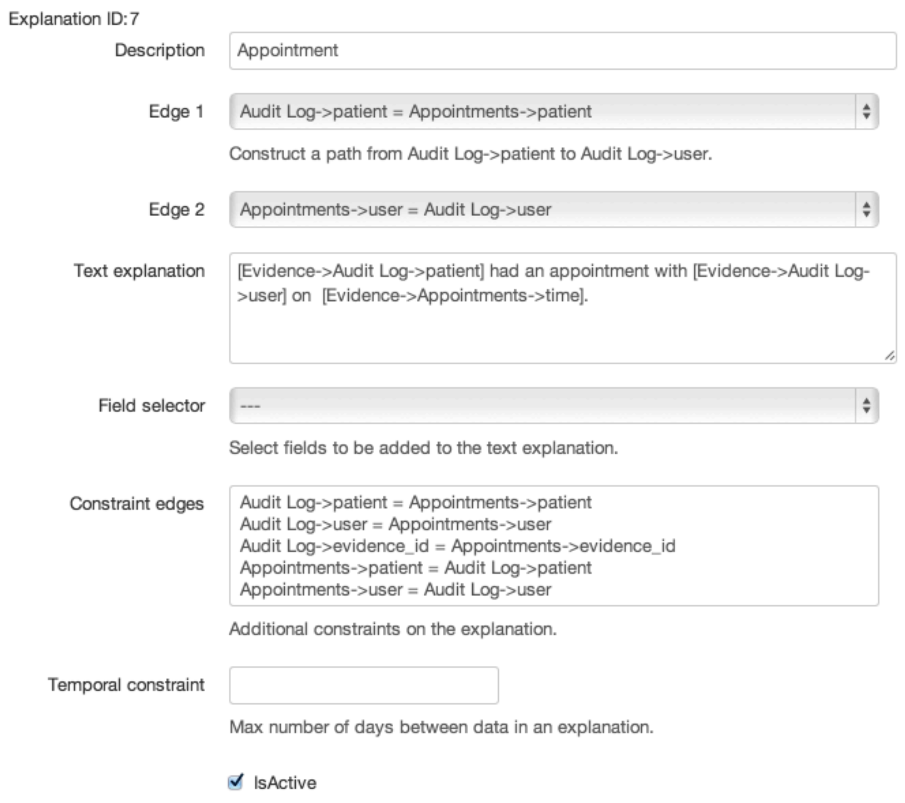
various components used to build the prototype (as shown in Figure 8.1).

Integration With Epic: To perform an audit, compliance officers must learn about a patient's treatment (e.g., who was involved in a patient's care). This manual exploration often involves analyzing clinical notes and contacting department leaders to discover which employees were assigned to treat a patient. As discussed previously, the explanation-based auditing system automates much of this manual procedure. However, before this automation can proceed, patient data must be made available to the auditing system. Ideally, the EMR system would allow the auditing system to be tightly integrated with the EMR database. Unfortunately, third-party integrations are not easily configured. Instead, for this work, mechanisms were put place (with the help of the hospital's IT staff) to dump data to a text file, which could then be uploaded into the auditing system. Now, to perform an audit, compliance officers simply enter a list of patient IDs and a time range, and then the associated patient data are extracted and uploaded. The current procedure works with the Epic EMR back-end system, but this type of integration is possible with other popular EMR systems (e.g., Cerner [14] and PracticeFusion [66]).

After developing the extraction tools, the next question to answer is: *What specific data items should be extracted?* Ideally, every piece of information describing patient treatment would be extracted. Interestingly, the Epic database stores much of this patient treatment information in a single *encounters* table. The encounters table lists patient appointments, visits, medications and over 100 other encounter types. Other data that are extracted include patient ICD-9 diagnosis information, the audit log, and employee department information. Additional types of information will be explored in the future.

Data Organization: The explanation-based auditing system's algorithms leverage relational operators in their computation. Therefore, it is only natural to store the extracted data in a relational database. In this case, a PostgreSQL database [65] stores the extracted data and the data are presented using a Python and Django web interface [25]. The database is organized in a few conceptual units: (i) patient encounters and accesses to medical records (i.e., hospital events), (ii) patient characteristics (e.g., ICD-9 data), (iii) employee characteristics (e.g., department information), and (iv) auditing meta-data such as which accesses are explained and why.

Explanations: The basic unit in the auditing system is an explanation. An explanation is represented by a path connecting the data that are accessed (i.e., the patient's medical record) to the employee accessing the data, which captures the reason why an access occurred. The prototype system allows compliance officers to manually construct explanations using a simple web form (Figure 8.2). Paths are formed by connecting edges, which are relationships between database attributes such as foreign-key relationships.



Explanation ID:7

Description	Appointment
Edge 1	Audit Log->patient = Appointments->patient Construct a path from Audit Log->patient to Audit Log->user.
Edge 2	Appointments->user = Audit Log->user
Text explanation	[Evidence->Audit Log->patient] had an appointment with [Evidence->Audit Log->user] on [Evidence->Appointments->time].
Field selector	--- Select fields to be added to the text explanation.
Constraint edges	Audit Log->patient = Appointments->patient Audit Log->user = Appointments->user Audit Log->evidence_id = Appointments->evidence_id Appointments->patient = Audit Log->patient Appointments->user = Audit Log->user Additional constraints on the explanation.
Temporal constraint	Max number of days between data in an explanation.

✓ IsActive

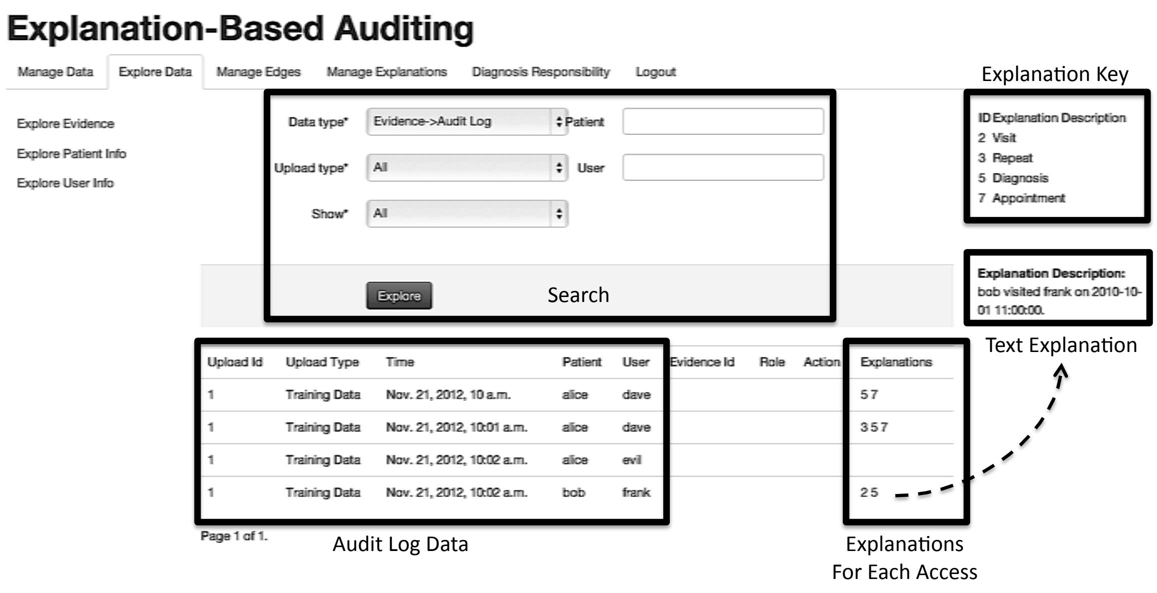
Figure 8.2: An example explanation as seen from the prototype's web interface. Notice that the edges create a path from the patient in the audit log to the employee (or user).

Various other properties can be specified after the explanation's path has been constructed. The first property is a templated text explanation (e.g., "[patient] had an appointment with [user] on [appointment.time]"). These text explanations are extremely useful for compliance officers to understand why accesses occur, and can be used to quickly create access reports. Other properties include additional (edge) constraints or the maximum amount of time between data used in an explanation (this temporal constraint allows compliance officers to express that an appointment can only explain accesses up to a month from the date of the appointment). Lastly, the explanation can be either in an active or inactive state (only active explanations are used to audit).

Explanation Miner: In the naive case, compliance officers can specify all explanations and their associated paths. However, there are hundreds of encounter types, each of which may require its own explanation. Therefore, the prototype provides the necessary algorithms to mine explanations from the data. Specifically, the algorithm's objective is to find all explanations that *explain* why at least $s\%$ of the accesses in the audit log occur (here the number of accesses explained is the number of accesses returned by a query evaluated on the database where the query's selection condition is logically equivalent to the explanation's path). The algorithm works by iteratively constructing paths of longer lengths. At the end of each iteration, the system prunes those paths with insufficient support (i.e., do not explain a sufficient number of accesses). After the frequent explanations are mined, they are presented to the compliance officer for approval (i.e., to set the explanation as active).

In addition to mining explanations, the auditing system also is able to fill-in missing types of data. One type of missing information is diagnosis responsibility information. Diagnosis responsibility information refers to facts describing which departments are responsible for treating a given diagnosis (e.g., oncologists treat cancer). The prototype analyzes access patterns and patient diagnosis information to learn responsibility information. The explanation mining algorithms can use this information directly to produce new explanations.

Access Explorer: After explanation are mined, they can be tested (or evaluated) on the data, after which compliance officers can explore the results (Figure 8.3). The prototype provides mechanisms to search for accesses by patient, employee or if an access occurred for a reason. For example, the compliance officer can choose to only display those suspicious accesses that cannot be explained by an active explanation, which can drastically reduce the number of accesses for review. In addition to simply viewing accesses, the prototype allows compliance officers to click on an access and



Explanation-Based Auditing

Manage Data | Explore Data | Manage Edges | Manage Explanations | Diagnosis Responsibility | Logout

Explore Evidence
Explore Patient Info
Explore User Info

Data type*: Evidence->Audit Log
Patient:
Upload type*: All
User:
Show*: All

ID	Explanation Description
2	Visit
3	Repeat
5	Diagnosis
7	Appointment

Explanation Description: bob visited frank on 2010-10-01 11:00:00.

Upload Id	Upload Type	Time	Patient	User	Evidence Id	Role	Action	Explanations
1	Training Data	Nov. 21, 2012, 10 a.m.	alice	dave				5 7
1	Training Data	Nov. 21, 2012, 10:01 a.m.	alice	dave				3 5 7
1	Training Data	Nov. 21, 2012, 10:02 a.m.	alice	evil				
1	Training Data	Nov. 21, 2012, 10:02 a.m.	bob	frank				2 5

Page 1 of 1.

Figure 8.3: Web interface to explore accesses. The compliance officer can search for accesses by patient, employee, or if there is a known reason for access. At the right of each row lists the set of explanations (specifically, their IDs) that explain why the access occurred (e.g., a patient's visit), if one exists. If an explanation is clicked, a text explanation is displayed.

display its associated templated text explanation.

8.3 Demonstration

There are many real-world scenarios where the explanation-based auditing system prototype can improve compliance officer efficiency. First, consider the case where a patient Alice files a complaint about a snooping employee. Without the prototype, compliance officers would need to analyze each access in the audit log for the patient, call department leaders and analyze clinical notes to find the snooper. However, with the prototype, a large number of the accesses can be filtered so that compliance officers have a much smaller subset to review. For example, Figure 8.3 shows four accesses to Alice's record, three of which can be explained and filtered by clinical or operational reasons (e.g., appointments or visits), leaving only Dr. Evil's access for review. Additionally, the auditing system can produce an access report describing the reason for each appropriate access (e.g., “Dr. Dave had an appointment with Alice”).

In addition to retrospective auditing, the prototype makes it possible to perform proactive random audits of employees. Because the prototype drastically reduces the auditing burden, compliance officers can audit accesses to a randomly selected patient each day. Ideally, these proactive audits, and employees’ awareness that they are occurring, will deter future inappropriate behavior.

8.4 Conclusion

The explanation-based auditing prototype allows compliance officers to quickly detect inappropriate accesses to electronic medical records. It has functionality to extract data from an EMR database, load them into the prototype, mine explanations and then filter appropriate behavior. The prototype will be evaluated in hospitals to determine its effectiveness and accuracy. In the future, various enhancements will be added to the prototype such as improved auditing analytics and exploration features.

CHAPTER IX

Auditing SQL Logs

9.1 Introduction

During normal use, most employees access EMR databases through restrictive user interfaces that allow the employee to access a single patient's record at a time. In these cases, the EMR typically stores an application-level audit log containing (employee, patient) pairs. However, in certain cases, both when dealing with electronic medical records and in other domains, employees are not restricted to such an interface, and are instead permitted to issue ad-hoc SQL queries over a backend relational DBMS.

For this reason, there has also been significant interest among the database research community and industry in building support for auditing at the level of a relational DBMS [6, 30, 31, 45, 59]. In this chapter, we move down a level in the software stack and consider a problem arising from DBMS-level auditing. Specifically, we consider the context where the data in the DBMS are protected by an access control policy, which designates which users can access which data. However, access control policies are notoriously difficult to configure, and mistakes are common [67]. In this chapter, we consider a common problem: A misconfiguration is detected in an access control policy after that policy has already been deployed for a period of time. To comply with data breach reporting rules, it is necessary to go back in time and determine which queries may have revealed unauthorized information.

An obvious approach to this problem is to maintain a log of all SQL queries and to retrieve those queries that explicitly accessed unauthorized data. However, if done naively, this fails to account for the fact that unauthorized information may have propagated (explicitly or implicitly) via updates. For example, when an insert operation copies a row from one table to another, this operation creates an explicit flow of information; a user who reads the copied row learns the original value. Similarly, when a row is updated based on a condition (e.g., UPDATE Patients SET Age = 'XXX' WHERE Name = 'Bob'), the operation creates an implicit flow of information since the value XXX implicitly reveals that Name = Bob. Thus, not only may a misconfigured access control policy allow a user to access unauthorized parts of the database, but the unauthorized data can be propagated to other parts of the database that can be read by future queries. Past work on database auditing considered the task of retrieving logged SQL queries that were affected by user-specified "sensitive" data, but did not address the challenge of updates [6].

9.1.1 Challenges

Effectively responding to database access control misconfigurations presents several challenges:

- **Updates:** The solution should be able to find all past queries that revealed unauthorized information, either by directly accessing unauthorized data or as the result of information propagation caused by update operations.
- **Lightweight/Non-disruptive:** The solution should easily integrate with existing DBMS infrastructure and introduce minimal overhead during normal database operation.

- **Efficient Response:** When a policy misconfiguration is detected, the solution should efficiently identify past queries that may have revealed unauthorized information.

9.1.2 Our Contributions

In response to these challenges, we propose the PolicyReplay framework, which is highlighted by the following contributions:

- We introduce the novel idea of a declarative *misconfiguration response (MR) query*, which retrieves all past database queries that may have revealed unauthorized information. Our approach is based on the following insight, which cleanly addresses the problem of information propagation: Conceptually, the MR-query returns to the point of the misconfiguration, and completely replays the log of operations (updates and queries) using the new (correct) policy. If a query returns exactly the same result under the old (incorrect) and new (correct) policies, we know that the query has not revealed any unauthorized information. However, if the result has changed, the query is marked as "suspicious" since unauthorized information may have been revealed. The formal semantics are described in Section 9.3.
- Of course, the naive algorithm of returning to the point of the misconfiguration and completely replaying all past database operations is inefficient. To improve efficiency, in Section 9.4 we introduce a set of optimizations based on *static pruning*, *delta tables*, and *partial and simultaneous re-execution.* Our experiments (Section 9.10) indicate that by replaying operations in an efficient manner we are able to reduce the total runtime by an order of magnitude in several common cases.
- Interestingly, the approaches presented in this chapter (that can detect when a query has revealed unauthorized information due to a policy misconfiguration)

can also be used to transform a SQL audit log to an application-level audit log. That is, if the old policy allows access to all data in the database and the new policy allows access to all data but a single sensitive row, then the MR-query can determine if the query accessed the sensitive row (if the row is accessed, an audit log entry is created). Unfortunately, applying the MR-query to transform SQL logs is slow because the procedure must be repeated for each sensitive row. Recent work on SELECT triggers (completed while at Microsoft Research) addresses these performance issues and allows for efficient SQL audit log transformations [31].

9.2 Preliminaries

The PolicyReplay framework supports a modern row-level access control model. During normal database operation, we maintain an *operation log*, which records the text of all SQL operations, as well as a *transaction-time backlog database* representation of historical data [44]. These structures are easily incorporated into an existing DBMS, and past work has shown that they can be maintained with little impact on normal database operation [6].

9.2.1 Row-Level Access Control Policies

The goal of our work is to develop methods for responding to misconfigurations in database access control policies. While the access control model itself is not our main contribution, we will use a modern row-level access control model (e.g., as found in Oracle's fine-grained access controls [60]) throughout the chapter.

Formally, we will consider an old policy P_{old} and a new policy P_{new}. For each database user U, policy P_{old} (respectively, P_{new}) contains a selection of the form $\sigma_{S_{old}}(S)$ (respectively, $\sigma_{S_{new}}(S)$) for every table S in the database, where S_{old} (respectively, S_{new}) is a boolean condition involving only the attributes in table S.

A user U is given access to the subset of the database specified by the selection conditions in the policy. When the user executes a database command (i.e., a SELECT, INSERT, UPDATE, or DELETE statement), each table S referenced by the command is transparently replaced with the view defined by the additional selection.

Example IX.1. As a simple example, consider a hospital database that contains a single table *Patients*. Consider a user Dan, and suppose that under policy P_{old} Dan is allowed to see the rows of the Patients table with $Age < 30$ (i.e., $Patients_{old} = (Age < 30)$). If Dan issues the query `SELECT * FROM PATIENTS` while P_{old} is in effect, then the query is automatically rewritten as `SELECT * FROM PATIENTS WHERE Age < 30`.

Updates are handled similarly. Suppose that Dan issues the data modification command `UPDATE Patients SET Department = 'Pediatrics'`; this is rewritten as `UPDATE Patients SET Department = 'Pediatrics' WHERE Age < 30`.

Of course, row-level access control has its shortcomings, and alternatives have been proposed [50, 68]. Nonetheless, the row-level approach remains popular.

9.2.2 Operation Log

During normal database operation, a log is maintained, which records the text of all DML operations (SELECT, INSERT, UPDATE, and DELETE) that are performed. The operation log itself is an append-only table; each entry in the log contains the *timestamp* at which the operation was executed, as well as the associated *sql* (a string). It may also contain additional fields, such as usernames, etc. Figure 9.1 shows a (pared-down) operation log.

Many existing database systems already support this kind of logging [47, 60, 79], and past work has demonstrated that the performance impact during normal database operation is minimal [6]. For the purposes of this work, we will assume that log entries are written within the same transaction as the recorded operation; this guarantees that the timestamp ordering on the log records is equivalent to the serialization order of the operations.

9.2.3 Transaction-Time Backlog Database

In addition to the operation log, we will make use of a simplified transaction-time backlog database [44], which supports two basic operations: insert a row and delete a row. Each time a row is inserted or deleted, the system records the following information: the time, the type of operation (insert or delete), and the value of the row being inserted or deleted.[1] Under this representation, rows are never modified or deleted in place; instead, the new or updated row is appended to the end of the table.

Formally, we will use the notation DB^b to refer to a transaction-time backlog database, and S^b to refer to a single backlog table in the database. We will use DB^τ to refer to the static snapshot database that exists at time τ and S^τ to refer to the static snapshot of table S at time τ. Any DB^τ can be constructed from DB^b using methods described in [44].

Example IX.2. Consider the backlog table $Patients^b$ shown in Figure 9.2(a), and suppose that Id is the primary key for $Patients$. $Patients^3$, the static snapshot of the table at time 3, is as follows:

Id	**Name**	**Age**	**Disease**
1	Alice	10	Flu
2	XXX	20	Ulcer
3	Carlos	35	Broken Arm

[1]The effects of SQL UPDATE commands are captured as follows: If a tuple is modified, we model this as a delete (of the old tuple), followed by an insert (of the new tuple).

9.3 MR-Queries: Problem Definition

When a policy misconfiguration is detected, a security administrator or compliance officer needs to take steps to respond to the misconfiguration (e.g., report the incident to government regulators). In order to do this, she must determine which queries have revealed unauthorized information. We will refer to this task as a *misconfiguration-response (MR) query*. The MR-query plays a central role in breach reporting (see Section 9.9).

The goal of the MR-query is to retrieve every logged SQL query that disclosed unauthorized information. In the simplest case, a logged query is returned by the MR-query if it explicitly reads unauthorized data. However, this simple approach does not capture the propagation of information via updates. Instead, our problem formulation is based on the observation that a query is guaranteed *not* to reveal unauthorized information if its result is the same under the old (incorrect) and new (correct) policies. In contrast, when the query results are not the same, there is no such guarantee; we will refer to these SQL queries as *suspicious queries*.

Definition IX.3 (MR-Query)**.** The MR-query takes as input the old policy P_{old}, the new policy P_{new}, timestamps t_1 and t_2 ($t_1 \leq t_2$), the operation log, and the transaction-time backlog database DB^b. The MR-query returns the set of all entries e in the operation log that contain SELECT queries, have timestamps between t_1 and t_2, and such that the results of the queries would have been different if P_{old} had been replaced with P_{new}, effective at time t_1.

The semantics of an MR-query are easily understood in terms of a naive algorithm. It begins by constructing a second backlog database $\widehat{DB}^b$ by creating a new backlog table $\widehat{S}^b$ for each table S^b in DB^b, and copying into $\widehat{S}^b$ every row in S^b with

lid	timestamp	sql
L1	1	SELECT * FROM Patients
L2	2	SELECT * FROM Patients WHERE Age < 12
L3	3	UPDATE Patients SET Name = 'XXX' WHERE Name = 'Bob'
L4	4	SELECT * FROM Patients WHERE Name = 'XXX'
L5	5	INSERT INTO Temp SELECT Id, Name, Age, Disease FROM Patients
L6	6	SELECT * FROM Temp

Figure 9.1: Sample Operation Log

Time	Op	Id	Name	Age	Disease
0	Ins	1	Alice	10	Flu
0	Ins	2	Bob	20	Ulcer
0	Ins	3	Carlos	35	Broken arm
3	Del	2	Bob	20	Ulcer
3	Ins	2	XXX	20	Ulcer

(a) $Patients^b$

Time	Op	Id	A	B	C
5	Ins	1	Alice	10	Flu
5	Ins	2	XXX	20	Ulcer

(b) $Temp^b$

Figure 9.2: The backlog database DB^b is the result of the actual execution (when P_{old} was in place).

$timestamp \leq t_1$. Starting from t_1, the algorithm replays the operation log (updates and queries) using the new policy P_{new}, and applying all data modifications to $\widehat{DB}^b$. For every SELECT query in the log, it compares the result obtained using P_{new} and $\widehat{DB}^b$ to the result obtained using P_{old} and DB^b. Details are provided in Algorithm 2 in Appendix 9.5.

Example IX.4. Consider a database consisting of two tables: $Patients(Id, Name, Age, Disease)$ and $Temp(Id, A, B, C)$. In both tables, Id is the primary key.

Time	Op	Id	Name	Age	Disease
0	Ins	1	Alice	10	Flu
0	Ins	2	Bob	20	Ulcer
0	Ins	3	Carlos	35	Broken arm

(a) $\widehat{Patients}^b$

Time	Op	Id	A	B	C
5	Ins	1	Alice	10	Flu

(b) $\widehat{Temp}^b$

Figure 9.3: The backlog database $\widehat{DB}^b$ is what would have resulted if P_{old} were replaced with P_{new}, effective at time 1.

Suppose that at time 1 the administrator deploys a policy allowing user Dan to see only those rows of $Patients$ with $Age < 30$ (i.e., $Patients_{old} = (Age < 30)$), and all rows of $Temp$ (i.e., $Temp_{old} = true$). Later, at time 7, she discovers that the policy was misconfigured, and she corrects the policy so that Dan can see only those Patients with $Age < 15$ (i.e., $Patients_{new} = (Age < 15)$ and $Temp_{new} = true$). Unfortunately, the misconfigured policy was in effect for the period between $t_1 = 1$ and $t_2 = 7$. In order to respond to the misconfiguration, the administrator needs to figure out which queries, evaluated during this time, would have produced different results if the correct policy had been in place.

Figure 9.1 shows an example operation log, and Figure 9.2 shows the backlog database DB^b. Observe that if P_{old} had been replaced with P_{new} at time 1, some of the resulting data modifications would have been different. The backlog database resulting from this case (denoted $\widehat{DB}^b$) is shown in Figure 9.3. For example, notice that the update in L_3 will not affect any tuples when P_{new} is in effect because user Dan does not have access to Patients with $Age > 15$ (i.e., Bob) in this case.

Finally, Figure 9.4 compares the results of each SELECT query when evaluated using DB^b and P_{old} and using $\widehat{DB}^b$ and P_{new}. Notice that the queries in L_1, L_4, and L_6 return different results; thus these log entries are returned as the result of the MR-query.

Less strict approaches may be insufficient to detect the disclosure of unauthorized information. For example, an alternative method might attach annotations to each row (e.g., [11]) and propagate these annotations from row to row as the log of operations is executed to track dependencies. A query would be marked suspicious if a row in the result is dependent on a row that is only accessible due to the misconfiguration. Therefore, by only executing the log under the old policy and combining the

Id	Name	Age	Disease
1	Alice	10	Flu
2	Bob	20	Ulcer

(a) L_1 (Old)

Id	Name	Age	Disease
1	Alice	10	Flu

(b) L_1 (New)

Id	Name	Age	Disease
1	Alice	10	Flu

(c) L_2 (Old)

Id	Name	Age	Disease
1	Alice	10	Flu

(d) L_2 (New)

Id	Name	Age	Disease
2	XXX	20	Ulcer

(e) L_4 (Old)

Id	Name	Age	Disease

(f) L_4 (New)

Id	A	B	C
1	Alice	10	Flu
2	XXX	20	Ulcer

(g) L_6 (Old)

Id	A	B	C
1	Alice	10	Flu

(h) L_6 (New)

Figure 9.4: Comparing the results of logged queries to illustrate the semantics of MR-queries

query results with annotation information, this approach can detect some suspicious activity; however, it has significant weaknesses that are managed by the MR-query.

Example IX.5. Consider the policies $Patients_{old} = (Age < 30)$, $Patients_{new} = (Age < 18)$ and $Temp_{new} = Temp_{old} = true$ and the log of operations and data in Figure 9.5. Initially, the user learns from operation O1 that Bob has flu. Then, the delete operation deletes those rows in Temp that have the same disease as a row in the Patients table. In this example, Bob is deleted because Alice also has the flu; however, Alice is only accessible due to the misconfiguration. Therefore, the user inappropriately learns from the empty set result of operation O3 that someone in the Patients table has the flu. Unfortunately, the annotation method cannot detect this unauthorized access because the result of $O3$ on $Temp$ is the empty set and contains no annotations to analyze. In contrast, the MR-query will mark operation O3 as suspicious since the result is different between the policies.

As the example shows, the absence of a row in the result can result in the disclosure of unauthorized information when combined with the result of other queries.

Time	**Op**	**Id**	**Name**	**Age**	**Disease**	**Annotations**
0	Ins	1	Alice	25	Flu	P1

(a) $Patients^b$

Time	**Op**	**Id**	**Name**	**Age**	**Disease**	**Annotations**
0	Ins	1	Bob	10	Flu	T1
2	Del	1	Bob	10	Flu	P1, T1

(b) $Temp^b$

lid	**time**	**sql**
O1	1	SELECT * FROM Temp
O2	2	DELETE FROM Temp t USING Patients p WHERE t.disease = p.disease
O3	3	SELECT * FROM Temp

(c) Operation Log

Id	**Name**	**Age**	**Disease**

(d) Result for $O3$ on Temp

Id	**Name**	**Age**	**Disease**
1	Bob	10	Flu

(e) Result for $O3$ on $\widehat{Temp}$

Figure 9.5: A backlog database and log to demonstrate the weaknesses of annotation methods.

Unfortunately, annotation approaches are not able to detect these breaches. [2]

9.4 MR-Query Evaluation

The naive algorithm is useful for expressing the semantics of an MR-query, but it would be inefficient to actually evaluate an MR-query in this way. In this section, we describe a set of optimizations, which greatly improve the efficiency of MR-queries:

- **Static Pruning:** When the operation log contains only queries (SELECT statements), in Section 9.4.2 we provide a static pruning condition by which we can determine that certain queries are unsuspicious, without re-executing them.

- **Delta Tables:** In Section 9.4.3, we extend the static pruning condition to an operation log that also contains data modifications (INSERT, UPDATE, and DELETE statements). The idea is to store a concise description of how the database has changed as a result of the new policy (i.e., the difference between $\widehat{DB}^b$ and DB^b) using structures called *delta tables*. Then, we extend the pruning conditions from

[2]There are more subtle cases as well. For example, consider the case where the old policy is too strict (e.g., for some table S, $S_{old} \subseteq S_{new}$). Operations like MINUS or EXCEPT can produce unauthorized accesses.

the query-only case.

- **Partial and Simultaneous Re-Execution:** When the above pruning strategies fail, it is necessary to re-execute certain operations. To improve performance in this case, in Section 9.4.4, we introduce two further optimizations: The first is based on the observation that we may be able to determine mid-execution that an operation is unsuspicious, in which case we can stop executing the operation. While the naive algorithm requires that we entirely re-execute each query twice (once on $\widehat{DB}$ using P_{new} and once on DB using P_{old}), our second optimization is based on the observation that these two queries actually share much computation; thus we propose to execute the two queries simultaneously.

9.4.1 Class of Operations

For the remainder of this chapter, we will restrict our discussion to the following classes of logged SQL operations: (i) select-project-join (SPJ) queries, (ii) insert operations where the rows to be inserted are determined by an SPJ query, (iii) update operations where attributes of a row are set to constant values if the row satisfies a selection condition, and (iv) delete operations where a row is deleted if it satisfies a selection condition. For simplicity, we will not address the larger class of aggregate-select-project-join (ASPJ) queries, but many of our techniques can be applied to this case.

9.4.2 Static Pruning (Queries Only)

We begin with the simplest case, where the operation log contains *only* queries (i.e., SELECT statements). In this case, it is sometimes possible to determine statically that a query is not suspicious (i.e., without re-executing the query). As we will see, this case is not particularly practical on its own, but it provides a building block for the general case in Section 9.4.3. This static pruning condition is formalized through *delta expressions*.

Definition IX.6 (Delta Expressions)**.** Delta expressions logically describe the differences between the old and new policies. Specifically, for table S:

- $\delta_S^- = S_{old} \wedge \neg S_{new}$ is a *logical description* of the tuples from S that are visible to the user under the old policy, but not under the new policy.
- $\delta_S^+ = S_{new} \wedge \neg S_{old}$ is a *logical description* of the tuples from S that are visible to the user under the new policy, but were not visible under the old policy.

The intuition for the static pruning condition is straightforward. Each table S can be broken down, logically, into three components: (1) tuples that were visible under the old policy, but are no longer visible under the new policy (δ_S^-), (2) tuples that were not visible under the old policy, but are visible under the new policy (δ_S^+), and (3) tuples whose visibility is unchanged. If we can determine that a query's selection condition filters out all tuples in δ_S^+ and δ_S^- (i.e., any tuples whose visibility has changed), and no rows from S have been modified, then we know that the result of the query was not affected by the policy misconfiguration. This intuition is formalized by the following theorem. (The proof can be found in Section 9.6.)

Theorem IX.7. *Consider a database with relations $S_1, ..., S_n$, and suppose that the operation log contains only queries (no updates). Let Q be a query in the log with associated selection condition C.*[3] *If the expression $C \wedge (\delta_{S_1}^- \vee \delta_{S_1}^+ \vee ... \vee \delta_{S_n}^- \vee \delta_{S_n}^+)$ is not satisfiable, then Q must* not *be suspicious.*[4]

[3] C is a standard propositional formula consisting of atoms of the form $attr\Theta constant$ and $attr1\Theta attr2$ connected by logical operations ($\wedge$, $\vee$, $\neg$), where $\Theta \in \{=, >, <, \leq, \geq, \neq\}$.

[4] Of course, the satisfiability problem is NP-complete [35]. However, the size of the input to the satisfiability problem here grows with the complexity of the conditions (δ_S^-, δ_S^+), not the data. Thus, we expect that in practice, where selection conditions are usually simple, this will perform reasonably well.

Example IX.8. To illustrate the static pruning condition, consider the first two log records in the operation log shown in Figure 9.1, and suppose again that $Patients_{old} = (Age < 30)$ and $Patients_{new} = (Age < 15)$. Thus, $\delta^{-}_{Patients} = (15 \leq Age < 30)$ and $\delta^{+}_{Patients} = false$.

We are not able to prune $L1$ using Theorem IX.7. For the second query ($L2$), however, we have $C = (Age < 12)$. Notice that $C \wedge (\delta^{-}_{Patients} \vee \delta^{+}_{Patients}) = (Age < 12) \wedge ((15 \leq Age < 30) \vee false)$ is not satisfiable. This means that, regardless of the underlying database instance, the query result could not have been affected by the misconfiguration. Thus, we know that $L2$ is not suspicious, and we can prune it.

9.4.3 Handling Data Updates

Unfortunately, the static pruning described in the last section is only valid in the case where there are no data modifications or updates. When there are updates, we must also consider the possibility that the underlying database instances may have been modified as a result of the policy misconfiguration. This challenge is illustrated by the following example.

Example IX.9. Consider again the operation log in Figure 9.1, and suppose that $Patients_{old} = (Age < 30)$ and $Patients_{new} = (Age < 15)$. Suppose also that there is no access control condition on the table $Temp$ (i.e., $Temp_{old} = Temp_{new} = true$). Logically, the portion of $Temp$ that is visible under both policies is unchanged. Indeed, we have $\delta^{-}_{Temp} = \delta^{+}_{Temp} = false$, so the static pruning condition from the previous section is technically satisfied, for example, for $L6$.

Unfortunately, this does not take into account the occurrence of data modifications. In the example, notice that the rows inserted into $Temp$ are different under the old and new policies (i.e., $Patients_{old}$ and $Patients_{new}$). For this reason, the result of the query in $L6$ is actually different in the two cases, and $L6$ should be returned by the MR-query!

To address this problem, we propose to construct *delta tables*, which store the difference between backlog tables S^b and $\widehat{S^b}$. As we replay the operation log, rather than applying updates to a full database copy (like the naive algorithm), our optimized algorithm (Algorithm 3 in Section 9.5) will use the delta tables to capture the difference between the updates that occurred when operating under the old policy and the updates that would have occurred under the new policy.

Definition IX.10 (Delta Tables)**.** Delta tables store the difference between the backlog versions of each table when using P_{old} and when using P_{new}. Specifically, for table S at time t:

- $\Delta_S^- = \sigma_{time \leq t}(S^b - \widehat{S^b})$ is the set of rows that get added to the backlog table when operating under the old policy, but not under the new policy.
- $\Delta_S^+ = \sigma_{time \leq t}(\widehat{S^b} - S^b)$ is the set of rows that get added to the backlog table when operating under the new policy, but not under the old policy.

Thus, $\sigma_{time \leq t}(\widehat{S^b}) = \sigma_{time \leq t}(S^b) \cup \Delta_S^+ - \Delta_S^-$.

Delta expressions and delta tables can be used in combination to develop a pruning condition that is valid in the presence of updates. For simplicity, consider first a logged operation that mentions a single table S. Suppose that the operation has timestamp t, and let C be the selection condition associated with the operation.[5] In this case, the following conditions are sufficient to guarantee that (i) If the operation is a query, it is *not* suspicious, and (ii) If the operation is a data modification, its

[5]Both queries (SELECT statements) and data modification operations (INSERT, UPDATE, and DELETE) can include selection conditions. For example, in our sample operation log, log record $L3$ includes the selection condition WHERE Name = “Bob.” In the case of an SQL statement that contains no explicit selection condition, let $C = true$.

effects are exactly the same under the old and new policies. Thus, we can safely ignore the operation if:

1. The expressions $C \wedge \delta_S^+$ and $C \wedge \delta_S^-$ are not satisfiable.

2. $\sigma_C(\Delta_S^+) = \sigma_C(\Delta_S^-) = \emptyset$.

The intuition is mostly analogous to the query-only case. Condition (1) allows us to determine statically that the logged operation relies only upon the portion of the data that is logically visible under both the old and new policies. Condition (2) additionally guarantees that none of the data selected by the operation has been altered (i.e., updated in a different way as a result of the old policy vs. the new policy). Of course, unlike the condition described in the previous section, this pruning condition is not completely static, since condition (2) depends on the specific data in the delta tables. However, this only requires re-processing the logged selection condition on the data that has been changed (i.e., the delta tables), rather than re-processing the query on the full database. For small misconfigurations, we observe that the sizes of delta tables are often small when compared to the size of the full database.

This intuition is formalized, and generalized to operations involving multiple tables, via the following theorem. (The proof can be found in Section 9.6.)

Theorem IX.11. *Consider an operation in the log with selection condition C and that references relations $S_1, ..., S_n$. Without loss of generality, let C be expressed in conjunctive normal form (CNF); that is, C is a conjunction of clauses, each of which is a disjunction of literals. The operation can be pruned if both of the following conditions are satisfied:*

1. *The expression $C \wedge (\delta_{S_1}^- \vee \delta_{S_1}^+ \vee ... \vee \delta_{S_n}^- \vee \delta_{S_n}^+)$ is not satisfiable.*

Time	Op	Id	Name	Age	Disease
3	Del	2	Bob	20	Ulcer
3	Ins	2	XXX	20	Ulcer

(a) $\Delta^-_{Patients}$

Time	Op	Id	Name	Age	Disease

(b) $\Delta^+_{Patients}$

Time	Op	Id	A	B	C
5	Ins	2	XXX	20	Ulcer

(c) Δ^-_{Temp}

Time	Op	Id	A	B	C

(d) Δ^+_{Temp}

Figure 9.6: Delta-Tables for Running Example

2. *Let C_{S_i} be the conjunction of clauses in C that mention* only *attributes in S_i. (If no such clauses exist, let $C_{S_i} = true$.) For each relation S_i, $\sigma_{C_{S_i}}(\Delta^+_{S_i}) = \sigma_{C_{S_i}}(\Delta^-_{S_i}) = \emptyset$.*

Example IX.12. To illustrate, consider the operation log shown in Figure 9.1. Consider also the backlog table $Patients^b$ shown in Figure 9.3. Suppose again that $Patients_{old} = (Age < 30)$, $Patients_{new} = (Age < 15)$, and $Temp_{old} = Temp_{new} = true$. Thus, $\delta^-_{Patients} = (15 \leq Age < 30)$, $\delta^+_{Patients} = false$, $\delta^-_{Temp} = false$, and $\delta^+_{Temp} = false$.

The alternate backlog database $\widehat{DB}^b$, which would be constructed if the new policy was in place, is shown in Figure 9.3. The delta tables $\Delta^-_{Patients}$, $\Delta^+_{Patients}$, Δ^-_{Temp}, and Δ^+_{Temp} for this example are shown in Figure 9.6. Using the above conditions, $L2$ can be pruned. However, the remaining operations $(L1, L3, L4, L5, L6)$ have to be at least partially re-executed, as we will describe in the next section.

9.4.4 Simultaneous and Partial Re-Execution

If the pruning strategies described in the previous subsections fail, it is necessary to at least partially re-execute the remaining logged operations. In this section, we introduce two additional optimizations. The first is based on *partial re-execution*; we can sometimes determine mid-execution that an operation can be ignored. The second is based on *simultaneous re-execution*; rather than executing each logged query twice (once with the old data and policy, and once with the new data and policy), we can often save computation by combining the two into a single query plan.

Partial Re-Execution

Our first optimization is based on the observation that it may not be necessary to completely re-execute every un-pruned logged operation; in some cases, we can determine mid-execution that the operation can be ignored / pruned.

Our basic approach is illustrated with a simple example. Consider the following SQL query, which was logged at time t:

```
SELECT *
FROM R, S, T
WHERE (R.A > 10 OR S.B < 20)
AND R.ID = S.ID AND S.ID = T.ID
```

Two sample plans for this query (under the old and new policies) are shown in Figure 9.7. For any time t, we can compute static snapshot S^t from the backlog S^b and $\widehat{S}^t$ from $\widehat{S}^b = S^b \cup \Delta_S^+ - \Delta_S^-$.

Extending the intuition from the previous section, we can safely ignore this query if we can conclude that the two plans produce the same result. In the general case, we can establish this by identifying a *cut* in the query plan such that the intermediate results of both queries at every point in the cut are equivalent. Figure 9.8 shows three possible cuts for the example plan.

One way to check whether a cut exists is to evaluate both queries in a "side-by-side" manner using the backlog database and delta tables. This approach re-evaluates both queries (old and new) from the bottom up; after evaluating each operator, it checks whether the results are the same in both cases. If a cut is found, there is no

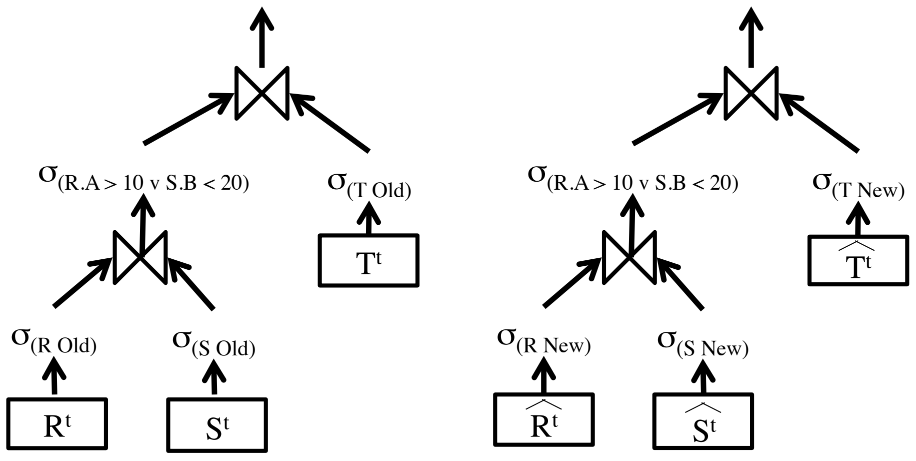


Figure 9.7: Query Plans For The Old and New Policies

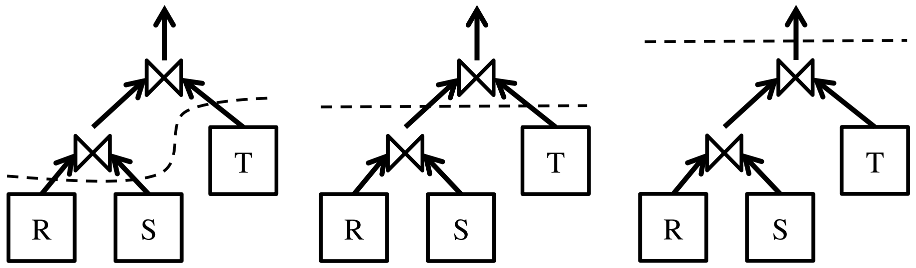


Figure 9.8: Possible Query Plan Cuts

need to continue executing the queries. If no cut is found, the process continues until both queries are completely re-executed. A discussion of implementation tradeoffs is in Section 9.8.

Of course, the weakness of this approach is that each operation must be run twice (once on the old data and policy, and once on the new data and policy). For this reason, we introduce one more optimization that allows us to evaluate both queries simultaneously.

Simultaneous Re-Execution

One way to test if a cut exists is to evaluate both queries (old and new) in a “side-by-side” manner, using the backlog database and delta tables. However, in the worst case, this process wastes a lot of work. Notice, for example, in Figure 9.7, if there is significant overlap between R^t and $\widehat{R}^t$, and between S^t and $\widehat{S}^t$, then the two queries are joining many of the same tuples.

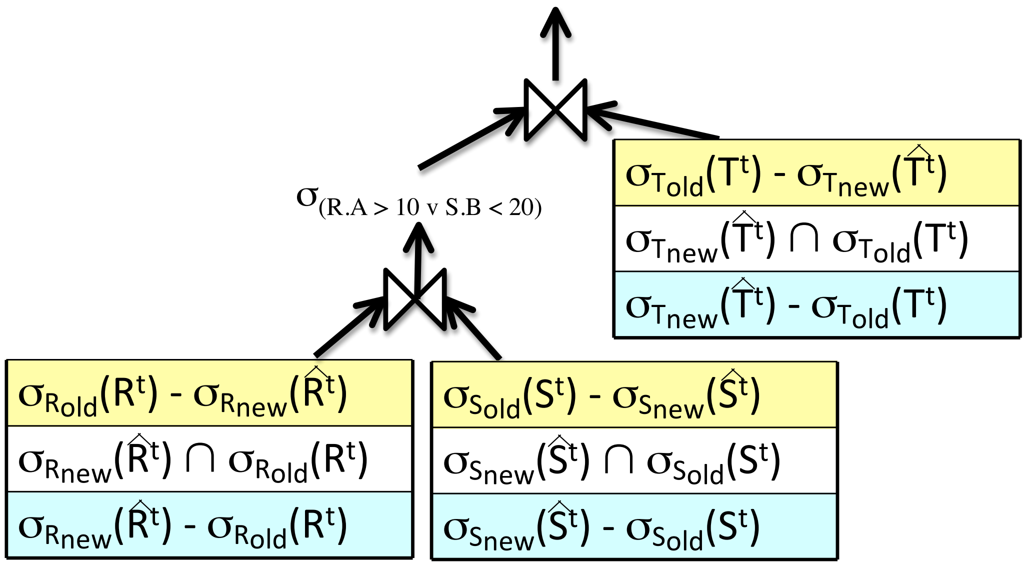
As a more efficient alternative, we instead propose merging the two (old and new) query plans to produce a single plan.[6] Figure 9.9 shows the merger of the plans in Figure 9.7. In this figure, the policy-based selection conditions (e.g., S_{old} and S_{new}) are pushed all the way down, and shown as part of the data. For each table S, we combine the rows from $\sigma_{S_{old}}(S^t)$ and $\sigma_{S_{new}}(\widehat{S}^t)$, and we add a flag to each row to indicate where it came from. "New" tuples are those that are emitted only under the new plan. "Old" tuples are emitted only under the old plan. "Unchanged" tuples are emitted under both plans. During query re-execution, these flags are created dynamically and propagated through the plan to ensure result correctness. More information on creating and propagating flags through the query plan can be found in Section 9.7.

When evaluating the combined query plan, we can conclude that the query was unaffected by the misconfiguration if there exists a cut in the plan such that the intermediate results along all points in the cut contain no tuples flagged as "Old" or "New."

9.4.5 Putting it All Together

Using the optimizations that we have described so far, we are now ready to describe our general algorithm for processing MR-queries, which addresses many of the inefficiencies of the naive algorithm. The algorithm begins by creating, for each table S^b in DB^b, tables Δ_S^- and Δ_S^+, which are initially empty. Then, starting from t_1, it replays the operation log forward. For each logged operation, it first checks

[6]In some ways, this is related to the idea of multi-query optimization [70, 84], the goal of which is to simultaneously evaluate multiple queries, with shared sub-expressions, on a single database. In contrast, in our setting, we need to evaluate the same query on two slightly different databases.



$\sigma_{(R.A > 10 \vee S.B < 20)}$
$\sigma_{Told}(T^t)$ - $\sigma_{Tnew}(\hat{T}^t)$
$\sigma_{Tnew}(\hat{T}^t) \cap \sigma_{Told}(T^t)$
$\sigma_{Tnew}(\hat{T}^t)$ - $\sigma_{Told}(T^t)$
$\sigma_{Rold}(R^t)$ - $\sigma_{Rnew}(\hat{R}^t)$
$\sigma_{Rnew}(\hat{R}^t) \cap \sigma_{Rold}(R^t)$
$\sigma_{Rnew}(\hat{R}^t)$ - $\sigma_{Rold}(R^t)$
$\sigma_{Sold}(S^t)$ - $\sigma_{Snew}(\hat{S}^t)$
$\sigma_{Snew}(\hat{S}^t) \cap \sigma_{Sold}(S^t)$
$\sigma_{Snew}(\hat{S}^t)$ - $\sigma_{Sold}(S^t)$

Figure 9.9: Combined Query Plan. Tuples flagged "New" are shown in blue (bottom third of each table), "Old" in yellow (top third), and "Unchanged" in white (middle third).

whether it is possible to prune the operation using the criterion in Theorem IX.11. If the pruning condition fails, then the algorithm must (at least partially) re-execute the logged operation. More specifically, we use the simultaneous re-execution plan described in Section 9.4.4. During re-execution, if a cut is found (as described in Section 9.4.4), then re-execution is aborted. Otherwise, the re-execution is carried to completion, at which point the results are compared (in the case of logged queries), or the delta-tables are updated (in the case of logged data modification operations). Pseudo-code is provided in Algorithm 3 in Section 9.5.

9.5 Algorithm Details

Pseudo-code for the naive algorithm, which is used to help illustrate the semantics of MR-queries, is provided in Algorithm 2.

Pseudo-code for the optimized algorithm, which includes static pruning, delta-tables, simultaneous and partial re-execution, is shown in Algorithm 3. In the pseudo-code, $Prunable(C, delta\ tables)$ returns true if the condition in Theorem IX.11 is

Algorithm 2 Naive Algorithm for Evaluating an MR-Query

```
Input: $P_{old}$, $P_{new}$, $t_1$, $t_2$, operation log, backlog database $DB^b$
Output: Set of suspicious queries
1: Construct a second backlog database $\widehat{DB}^b$ by creating a new backlog table $\widehat{S}^b$ for each table
   $S^b$ in $DB^b$. Copy into $\widehat{S}^b$ every row in $S^b$ with $timestamp \leq t_1$.
2: Let $e$ be the first entry in the operation log such that $e.timestamp \geq t_1$
3: while $e.timestamp \leq t_2$ do
4:   Let $t = e.timestamp$
5:   if $e.sql$ is a data modification operation then
6:     Evaluate $e.sql$ using policy $P_{new}$ and $\widehat{DB}^t$. Any data modifications are applied to $\widehat{DB}^b$.
7:   else if $e.sql$ is a SELECT statement then
8:     Let $Q = e.sql$
9:     Evaluate $Q$ on $\widehat{DB}^t$ (using $P_{new}$) and also on $DB^t$ (using $P_{old}$).
10:    if $Q(P_{new}(\widehat{DB}^t)) \neq Q(P_{old}(DB^t))$ then
11:      Add $e$ to the MR-query result set
12:  $e$ = next entry in the operation log
```

satisfied.

Algorithm 3 Optimized Algorithm for Evaluating an MR-Query

```
Input: $P_{old}$, $P_{new}$, $t_1$, $t_2$, operation log, backlog database $DB^b$
Output: Set of suspicious queries
1: For each table $S^b$ in $DB^b$, create tables $\Delta_S^-$ and $\Delta_S^+$, which are initially empty
2: Let $e$ be the first entry in the operation log such that $e.timestamp \geq t_1$
3: while $e.timestamp \leq t_2$ do
4:   Let $t = e.timestamp$
5:   Let $C$ be the selection condition associated with $e.sql$
6:   if $Prunable(C, delta\ tables)$ then
7:     skip the operation $e$
8:   else
9:     For every table $S^b$ in $DB^b$, let view $\widehat{S}^b = S^b \cup \Delta_S^+ - \Delta_S^-$.
10:    Simultaneously re-execute $e.sql$ on $\widehat{DB}^t$ with $P_{new}$ and on $DB^t$ with $P_{old}$
11:    if a cut is found then
12:      skip the operation $e$
13:    else if $e.sql$ is a SELECT statement then
14:      Add $e$ to the MR-query result set
15:    else if $e.sql$ is a data modification operation then
16:      Suppose that during the original execution, $e.sql$ added tuple set $T$ to $S^b$, but now
         $e.sql$ adds $T'$ to $S^b$. Update the delta-tables accordingly: $\Delta_S^+ = \Delta_S^+ \cup (T' - T)$; $\Delta_S^- =$
         $\Delta_S^- \cup (T - T')$
17:  $e$ = next entry in the operation log
```

9.6 Proofs

Proof of Theorem IX.7: The proof is straightforward. Suppose that Q was logged at t. If there are no updates, and $\sigma_C(P_{old}(DB^t)) = \sigma_C(P_{new}(DB^t))$ (i.e., the result

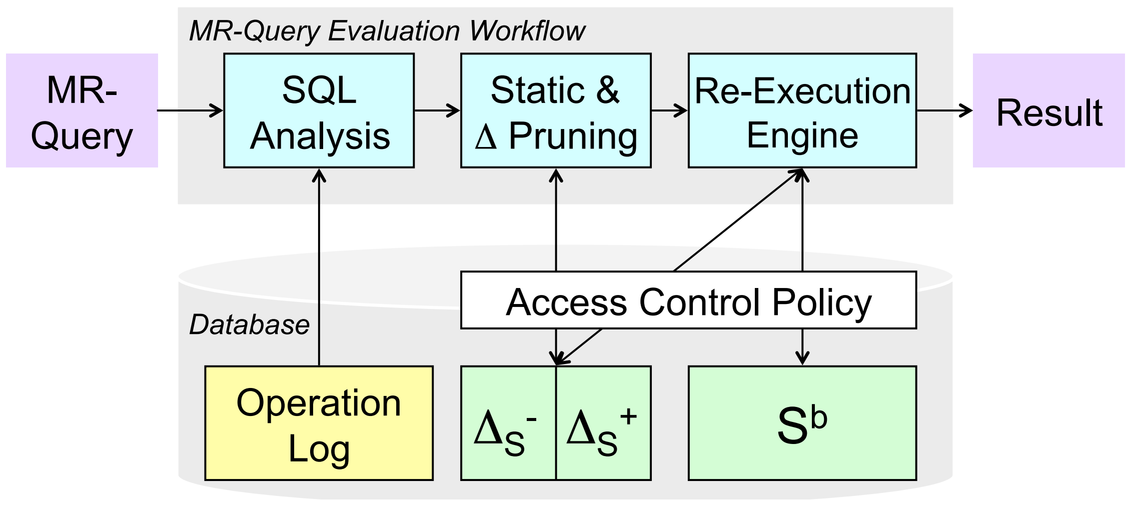


Figure 9.10: MR-Query Processor Components

of applying policy P_{old} and selection condition C is the same as applying P_{new} and C), then it is safe to prune Q.

Given the following definitions, we must show that $R_1 = R_2$.

$$\begin{aligned} R_1 &= \sigma_C(\sigma_{S1_{new}}(S_1^t) \times ... \times \sigma_{Sn_{new}}(S_n^t)) \\ R_2 &= \sigma_C(\sigma_{S1_{old}}(S_1^t) \times ... \times \sigma_{Sn_{old}}(S_n^t)) \end{aligned}$$

If $C \wedge (\delta_{S_1}^- \vee \delta_{S_1}^+ \vee ... \vee \delta_{S_n}^- \vee \delta_{S_n}^+)$ is not satisfiable, this means that none of the following expressions is satisfiable:

$$(C \wedge S1_{old} \wedge \neg S1_{new}), (C \wedge S1_{new} \wedge \neg S1_{old}), ...,$$
$$(C \wedge Sn_{old} \wedge \neg Sn_{new}), (C \wedge Sn_{new} \wedge \neg Sn_{old})$$

Using this, it is easy to show that $R_1 - R_2 = \emptyset$ and $R_2 - R_1 = \emptyset$.

Proof of Theorem IX.11: Suppose that the operation occurred at time t. If $\sigma_C(P_{old}(DB^t)) = \sigma_C(P_{new}(\widehat{DB}^t))$, then it is safe to prune the operation.

Given the following definitions, we must show that $R_1 = R_2$.

$$\begin{aligned} R_1 &= \sigma_C(\sigma_{S1_{new}}(\widehat{S_1}^t) \times ... \times \sigma_{Sn_{new}}(\widehat{S_n}^t)) \\ R_2 &= \sigma_C(\sigma_{S1_{old}}(S_1^t) \times ... \times \sigma_{Sn_{old}}(S_n^t)) \end{aligned}$$

These can be rewritten as follows (by pushing down selection conjuncts that refer only to a single table):

$$\begin{aligned} R_1 &= \sigma_C(\sigma_{S1_{new}}(\sigma_{C_{S_1}}(\widehat{S_1}^t)) \times ... \times \sigma_{Sn_{new}}(\sigma_{C_{S_n}}(\widehat{S_n}^t))) \\ R_2 &= \sigma_C(\sigma_{S1_{old}}(\sigma_{C_{S_1}}(S_1^t)) \times ... \times \sigma_{Sn_{old}}(\sigma_{C_{S_n}}(S_n^t))) \end{aligned}$$

Condition (2) is sufficient to guarantee that, for $i \in 1..n$,

$$\sigma_{C_{S_i}}(\widehat{S_i}^t) = \sigma_{C_{S_i}}(S_i^t).$$

The reason for this is that no tuples satisfying the condition C_{S_i} have been modified differently as the result of the different policies, since $\sigma_{C_{S_i}}(\Delta^+_{S_i}) = \sigma_{C_{S_i}}(\Delta^-_{S_i}) = \emptyset$.

Finally, per the same argument in the proof or Theorem IX.7, we can show that $R_1 - R_2 = \emptyset$ and $R_2 - R_1 = \emptyset$.

9.7 Simultaneous Re-Execution

9.7.1 Combined Tables and Flags

When executing a combined query on table S, we retrieve $\sigma_{S_{old}}(S^t) \cup \sigma_{S_{new}}(\widehat{S}^t)$, and we flag the input as follows:

- **"New" Flags:** $\sigma_{S_{new}}(\widehat{S}^t) - \sigma_{S_{old}}(S^t)$
- **"Old" Flags:** $\sigma_{S_{old}}(S^t) - \sigma_{S_{new}}(\widehat{S}^t)$
- **"Unchanged" Flags:** $\sigma_{S_{old}}(S^t) \cap \sigma_{S_{new}}(\widehat{S}^t)$

This can be explained as follows. When accessing a table S, under the old policy and data, we would have retrieved $Old(S) = \sigma_{S_{old}}(S^t)$. Under the new policy, we retrieve $New(S) = \sigma_{S_{new}}(\widehat{S}^t)$. This is shown, for example, in Figure 9.7.

When evaluating the combined policy, we will take all tuples from $New(S) \cup Old(S)$ as input. We want to flag all tuples in $New(S) - Old(S)$ as "New", tuples in $Old(S) - New(S)$ as "Old", and tuples in $New(S) \cap Old(S)$ as "Unchanged".

During query re-execution, these flags are created dynamically and propagated through the plan according to the following rules (This can be done in SQL, without modification to the DBMS engine):

- **Selection:** If a selection takes a tuple as input, and the tuple passes the selection filter, the output tuple keeps the same flag.
- **Projection:** Similarly, if a projection operator takes a tuple as input, the corresponding output tuple has the same flag.
- **Join:** The challenging operator is join. If a join takes as input two tuples with the same flag (i.e., both "New", both "Old", or both "Unchanged"), the emitted result tuple maintains that same flag. If the join takes as input two tuples such that one tuple is "Old" or "New", and the other "Unchanged", the resulting tuple inherits the "Old" or "New" flag. Finally, if a join takes as input two tuples such that one is "Old" and the other "New," even if the two tuples satisfy the join condition, no result tuple is produced. The reason for the final case is due to the fact that these input tuples are actually part of different conceptual query executions.

9.7.2 Implementation

The combined tables (and flags) described above can be computed using standard SQL. This implementation is for the class of queries described in Section 9.4.1.

The following outlines how to create a combined table, and assigns the appropriate flags to each row of the table ($New = 1$, $Old = -1$, $Unchanged = 0$). This is accomplished with SQL by taking the union of S^t with $\widehat{S^t}$; additionally, for each row from S^t, we add a flag of -1, while for each row from $\widehat{S^t}$, we add a flag of 1. We then group by all attributes in the table except the dynamically generated flag. If the sum of the flags in a group is -1, this means that the row only exists under the old policy. If the sum of the flags in a group is 0, this means that the row exists under the new and old policies (-1 + 1 = 0). If the sum of the flags in a group is 1, this means that the row only exists under the new policy.

When executing the combined query, we can propagate flags according to the rules outlined in Section 9.7.1 by adding the following constraint for all pairs of tables `R` and `S` referenced in the from clause of the query:

```
((S.flag != (-1) * R.flag) OR
  (S.flag = 0 AND R.flag = 0))
```

Finally, if a query result contains only tuples with flags = 0 (i.e., unchanged rows), then the query is not suspicious. More generally, if the result of the query under the new policy is the same as the result under the old policy (even if the result contains flagged rows), then the query is not suspicious. For SPJ queries, we can check if the results under the two policies are the same by grouping by all attributes and summing the value of the flag; thus, if the old policy and new policy produce the same result for a given row, the resulting sum for the row will be zero.

9.8 Partial Re-Execution

We can safely ignore a query if we can conclude that the query plans from the old and new policy produce the same result. In the general case, we can establish this by identifying a *cut* in the query plan such that the intermediate results of both queries at every point in the cut are equivalent. Figure 9.8 shows three possible cuts for the example plan. The second cut, for example, would require that both of the following conditions be satisfied:

1. $\sigma_{R.A>10 \vee S.B<20}(\sigma_{R_{New}}(\widehat{R}^t) \bowtie \sigma_{S_{New}}(\widehat{S}^t))$
   $= \sigma_{R.A>10 \vee S.B<20}(\sigma_{R_{Old}}(R^t) \bowtie \sigma_{S_{Old}}(S^t))$

2. $\sigma_{T_{New}}(\widehat{T}^t) = \sigma_{T_{Old}}(T^t)$

There are several possible ways that partial re-execution can be implemented. Our current prototype supports a rudimentary version of option (3). However, in this section, we describe alternative implementation strategies, as well as the important design considerations.

1. **Internal Database Filters**: The goal of partial re-execution is to execute the query plan until we can guarantee that the result is not affected by the misconfiguration. One possible implementation strategy is to add filters inside the query processor that examine the propagation of rows through a pipelined query plan. If information from the filters can be used to determine that old and new flagged rows do not pass the filters across a cut in the query plan, then the execution of the query can be stopped. This strategy has the least impact on the query processor, and does not alter query plans. On the other hand, this method may not be practical since the internals of the database engine must be modified.

2. **Materialize and Check Intermediate Results**: For many query plans, the query optimizer chooses to materialize, rather than pipeline, some portions of the query plan. We can leverage these materialized intermediate results to check

if there exits a cut in the plan that does not contain an old or new flagged row. If there exits a cut, then the execution of the query can be stopped. If old or new flagged rows do exist, the materialized results can be used as input to the next part of the query plan. The extra cost of this approach is storing the intermediate result, scanning the result for flags, and, in the cases when the result contains old or new flags, reading the intermediate result and sending it to the next part of the query plan. Overall, this approach appears practical only when the optimizer is already materializing certain intermediate query results.

3. **Left-Deep Query Plan Analyzer**: The third implementation strategy for partial re-execution relies on a left-deep query plan, which is commonly generated by modern query optimizers. In a left-deep plan, the input tables are joined one at a time, according to a total order. (For example, in Figure 9.7, R is joined with S, and then the result is joined with T. Thus, the join order is R, S, T.)

   Briefly, the partial re-execution algorithm for left-deep plans works as follows: We begin with the last table in the join order (In Figure 9.7, this is T.), and we check whether this table can be pruned according to Theorem IX.11. If not, we must re-execute the entire query (i.e., the only possible cut is the rightmost cut in Figure 9.8). If it can be pruned, then we consider the previous table in the join order (in this case, S), and check whether S can be pruned according to Theorem IX.11. This algorithm continues until it reaches a table that cannot be pruned, at which point we must execute the subquery rooted at that position of the plan. For example, if we find that we can prune T, but not S, then we consider the second cut in Figure 9.8 by re-executing $R \bowtie S$, and checking the flagged rows in the result. If the result contains no tuples flagged as "Old" or

"New," we can stop. Otherwise, we must execute the entire query.

9.9 Constructing Breach Reports

Misconfiguration-response queries are at the core of a broader framework for reporting data breaches. In this section, we provide a brief overview of how such reports can be constructed, and what information they can contain.

At a high level, there are two different kinds of reporting, with somewhat different goals and requirements:

- **Organizational Reporting:** In the basic setting, the organization that has experienced the breach must construct a single report summarizing the entire event. In the case of HITECH, for example, a covered entity (e.g., hospital) needs to compile a report to be sent to the regulatory government agency (in this case, the office of the Secretary of Health and Human Services). Similarly, contractors doing work on behalf of a covered entity must report breaches to the covered entity. For example, if a hospital has outsourced billing to a separate company, and that company experiences a breach, it must send a report to the hospital.
- **Individual Reporting:** In the second case, when an organization has experienced a breach, in addition to compiling a single report summarizing the event, it may also be necessary to notify individuals whose personal information was compromised as part of the breach.

We will focus primarily on organizational reporting, but we will also describe our initial ideas for extensions to individual reporting.

Throughout the main body of the chapter, for ease of exposition, we used very simplistic operation logs as examples. In practice, the operation log contains the `timestamp` and `sql` text, but it also contains additional attributes, including but not limited to: the database and/or application `username` of the individual who issued the SQL command, an application identifier `appID`, and sometimes the `purpose` for which the query was issued, or a description of an external data `recipient` [7].

For each suspicious query Q, we can also explain how the query was affected by the misconfiguration. In particular, we can summarize the difference between the result that occurred under the old policy P_{old} and the result that would have occurred if P_{old} had been replaced with P_{new}. Suppose Q was logged at time t. Let Q_1 be the result of evaluating Q on DB^t using P_{old}, and let Q_2 be the result of evaluating Q on $\widehat{DB}^t$ using P_{new}. The difference between the two result sets can be summarized by two tuple sets: $Q_{new} = Q_2 - Q_1$ and $Q_{old} = Q_1 - Q_2$.

Following a misconfiguration, the most detailed organizational breach report contains information about every suspicious query Q in the operation log (i.e., each record in the result of the MR-query). This information includes the attributes from the log (e.g., `timestamp`, `sql`, `username`, `appID`, `purpose`, `recipient`), Q_{old}, and Q_{new}. Of course, we can further restrict and summarize the information in this detailed report based on the requirements of the regulation. For example, HITECH requires reports to document to whom data was disclosed (which can be explained using `username` or `recipient`) and the type and amount of data disclosed (which can be explained using Q_{new} and Q_{old}).

Individual reports can be constructed in a similar way, but they present several additional challenges. Specifically, we only need to send a report to a user U if her information was involved in the breach, and the report should only explain how U's information was disclosed. In the future, we plan to extend the MR-query using an approach related to SQL GROUP BY. At a high level, the idea is to first divide the data into buckets, based on some attribute (e.g., a separate bucket for each patient's information). Then, the MR-query with GROUP BY (conceptually) processes the MR-query once per bucket, using only the data in that bucket.

9.10 Experimental Evaluation

To evaluate our ideas, we implemented the PolicyReplay system (Figure 9.10). Given information about a policy misconfiguration, a critical component of the system is the efficient evaluation of MR-queries. We compare the performance of our optimized MR-query processing algorithm (Algorithm 3) and the naive algorithm (Algorithm 2). We utilize numerous simulated data sets and workloads that are tuned by specific parameters (Section 9.10.2) to determine under what conditions our optimizations improve performance.

9.10.1 Implementation and Environment

Our prototype system is implemented as a thin Java layer on top of PostgreSQL[7]; it currently supports the static pruning, delta tables, and simultaneous re-execution optimizations, as well as a rudimentary implementation of partial re-execution. We use the Java Constraint Programming Solver (JaCoP)[8] to implement static pruning. SQL parsing is assisted by Zql, a Java SQL Parser[9]. The experiments were executed on a dual core CPU with 2 GB of RAM, running Red Hat Linux.

9.10.2 Data and Workload

MR-query performance is evaluated with multiple simulated data sets and workloads of logged SQL operations. The underlying database contains tables $T_1, ...T_n$ that are each composed of ten attributes $a_1, ..., a_{10}$. We added an indexed primary key, timestamp attribute and operation-type attribute to each table in order to con-

[7]http://www.postgresql.org
[8]http://jacop.osolpro.com/
[9]http://www.gibello.com/code/zql/

struct the backlog database. Table T_i consists of R_i rows. The values for attribute a_j are selected from a uniform distribution in the range $[1, \min(100 \times j^2, R_i)]$; the different ranges are used to vary the selectivity of selection conditions on each attribute. The simulated logged SQL workload contains INSERT, UPDATE, DELETE and SELECT operations as described in Section 9.4.1. The logged workload is generated by tuning the parameters in Figure 9.11.

Parameter	**Description**
Policy Misconfiguration (PM)	The selectivity of the disjunction of the delta expressions on the underlying tables
Operation Selectivity (Sel)	The selectivity of a logged operation on each table
Select to Update Ratio (Ratio, R)	The proportion of all logged operations that are SELECT statements (1.0 implies all SELECTs)
Predicate Attributes (P)	The number of attributes that may be used to create a literal in the selection condition, one of which is the attribute with the policy misconfiguration (P = 1 implies the attribute with the misconfiguration is always chosen for a literal, while P = 8 implies there is a $\frac{1}{8}$ probability that the attribute with the misconfiguration is chosen for a given literal.)
Database Size (Rows)	Number of rows initially in the database
Number of Logged Operations (Ops)	The total number of logged operations

Figure 9.11: Experimental SQL Workload Parameters

9.10.3 Results

Static Pruning (No Updates)

Our first set of experiments measures the effectiveness of static pruning in the simple case, where the operation log only contains queries. Figure 9.12 compares the runtime performance of evaluating an MR-query using the naive approach, which re-executes all logged queries, and the static pruning method, which only re-executes the queries that cannot be pruned. The figure shows performance across a range of policy misconfigurations for a workload with 1% selectivity on a single table. For small misconfigurations, static pruning is able to prune a large proportion of queries, resulting in improved performance. As the size of the misconfiguration grows, more operations must be re-executed. This trend is expected because, with a larger misconfiguration, it is more likely that a logged query's selection condition will intersect the delta expression. Additional pruning results can be found in Section 9.11.

Pruning (With Updates)

Next, when the logged workload also contains updates, we measure the benefits of pruning with delta tables (Theorem IX.11). If a logged operation cannot be pruned, it is re-executed. Consider a workload on a single table where the all the parameters are fixed except for the size of the policy misconfiguration. When the misconfiguration is small (1% and 10%) as shown in Figures 9.13 and 9.14, it is more efficient to evaluate the MR-query using pruning with delta tables (Pruning + Delta Tables) than the naive method; the MR-query is evaluated more quickly because fewer operations must be re-executed. Another benefit of the delta tables is that they remove the cost of copying the database prior to evaluating the MR-query; for large databases, this cost can be large.

While pruning reduces the number of operations that must be re-executed, re-executing an operation using delta tables is more costly than re-executing an operation without delta tables because there is an extra cost to construct the table $\widehat{S}$ from S, Δ_S^-, and Δ_S^+. Thus, as the size of the delta tables grows, the re-execution cost increases. We observe that there exists a tradeoff point when it is no longer advisable to use pruning with delta tables, but is more efficient to use the naive method. This tradeoff point is determined by the parameters in the workload such as the size of the misconfiguration, the ratio of selects to updates, and the selectivity of the operations. Additional experimental results are in Section 9.11.

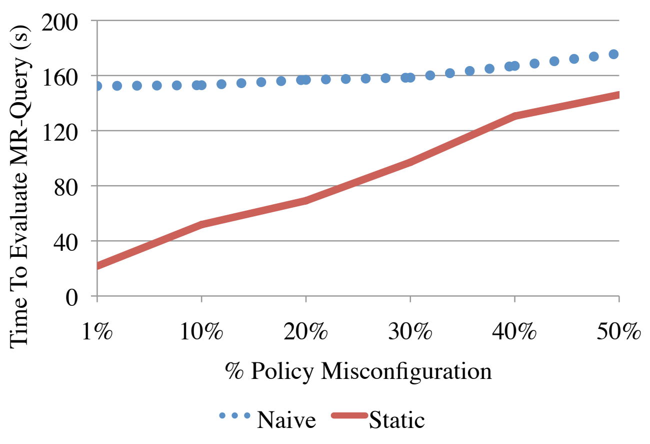


Figure 9.12: Static Pruning Performance(1% Sel., 250K Rows, P=1, R=1.0)

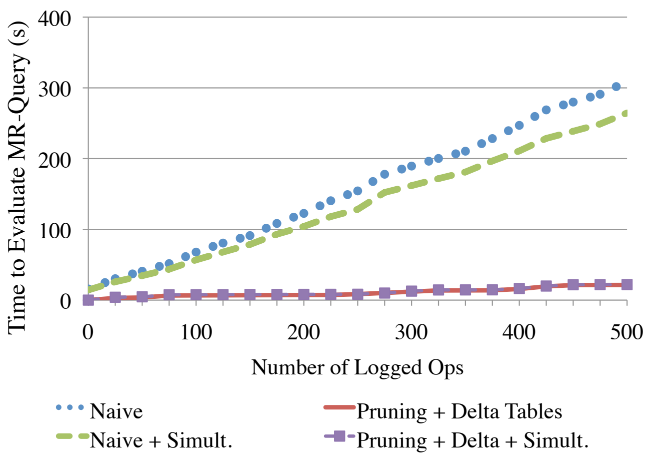


Figure 9.13: Performance - 1% PM(1% PM, 1% Sel., 250K Rows, P=1, R=0.9)

Simultaneous Query Evaluation

Finally, we evaluate the effectiveness of simultaneous re-execution. Figures 9.13 and 9.14 show the performance of the naive, naive plus simultaneous re-execution (Naive + Simult.), pruning with delta tables (Pruning + Delta Tables) and pruning with delta tables plus simultaneous re-execution (Pruning + Delta + Simult.) methods for different policy misconfigurations on a single table. We find that the performance of the naive method is improved when queries are simultaneously re-executed. The performance of the pruning with simultaneous approach improves

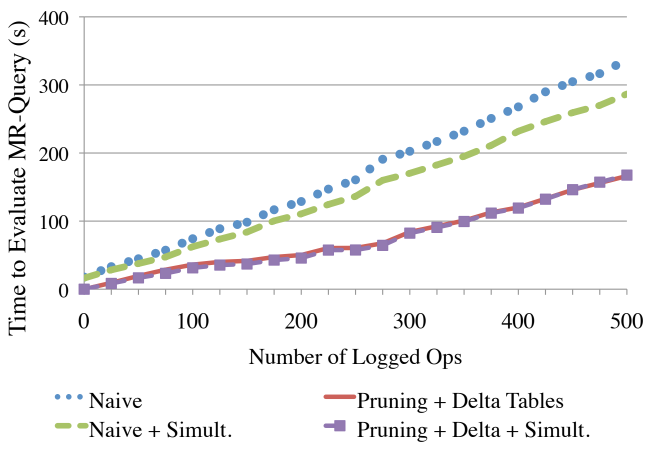


Figure 9.14: Performance - 10% PM(10% PM, 1% Sel., 250K Rows, P=1, R=0.9)

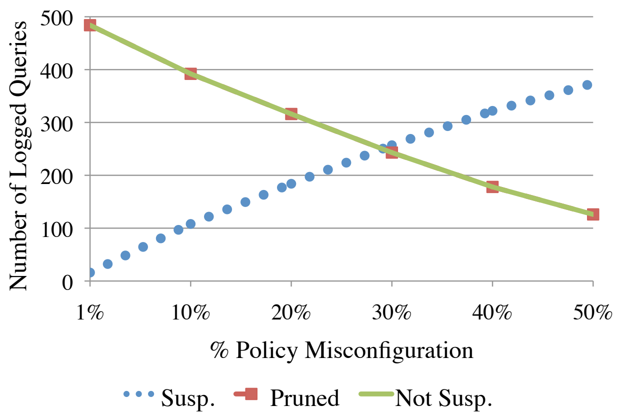


Figure 9.15: Static Pruning Statistics(1% Sel., 250K Rows, P=1)

slightly for small misconfigurations because only a few queries are re-executed; for larger misconfigurations, the benefits of simultaneous re-execution decrease.

9.11 Additional Experiments

9.11.1 Static Pruning (No Updates)

In addition to measuring the runtime performance of evaluating the MR-query, we also measured the effectiveness of static pruning by counting the number of logged queries that were suspicious, unsuspicious, and pruned. (Recall that the number of pruned queries must be $\leq$ the total number of unsuspicious queries.) Intuitively,

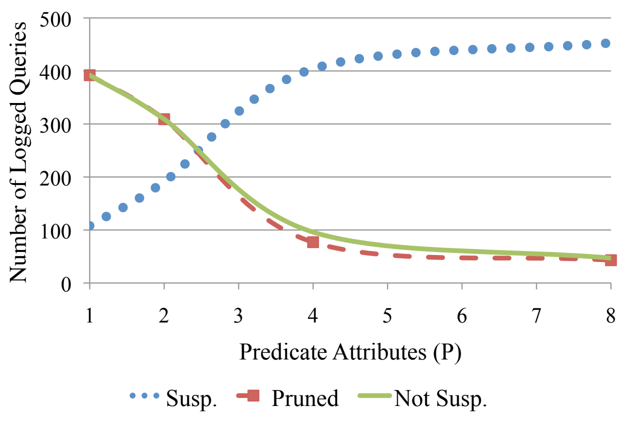


Figure 9.16: Static Pruning Statistics(1% Sel., 250K Rows, 10% PM)

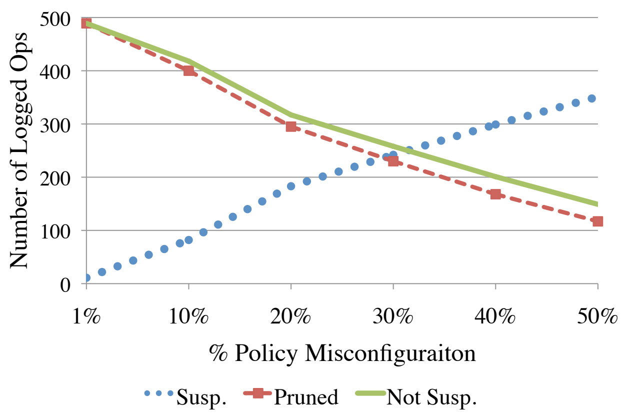


Figure 9.17: Pruning With Updates(1% Sel., 250K Rows, P=1, R=0.9)

as more queries are pruned, performance improves, since fewer queries must be re-executed.

Figure 9.15 shows the pruning information across a range of policy misconfigurations with a workload that has 1% selectivity on a single table, where the policy misconfiguration is on one attribute. For small misconfigurations (PM=1%), the static pruning is able to prune a large proportion of queries (484 pruned out of 500, in this case). As the misconfiguration gets larger (PM=50%), fewer queries can be pruned; however, we are able to prune all operations that are unsuspicious. This trend is expected because, with a lager misconfiguration, it is more likely that a

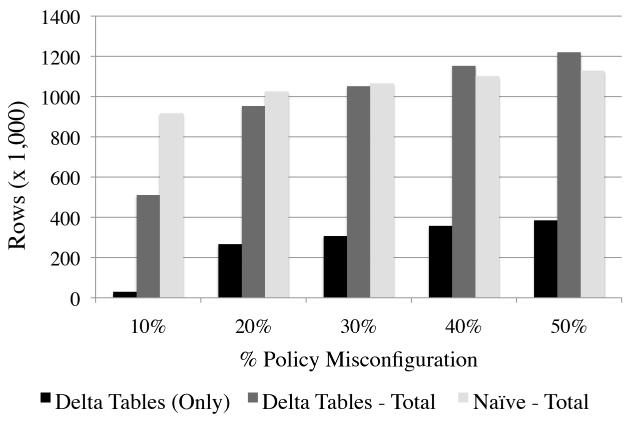


Figure 9.18: Database Size(1% Sel., 250K Rows, P=1, R=0.9)

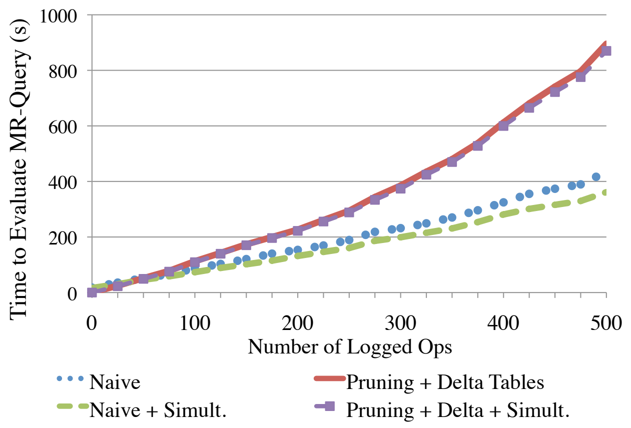


Figure 9.19: Performance - 50% PM(50%PM, 1%Sel., 250K Rows,P=1,0.9 Ratio)

logged query's selection condition will intersect with the delta expressions.

In order for an operation to be statically pruned, there must exist a literal in the selection condition that contradicts the delta expressions. Thus, when the attribute with the policy misconfiguration is not in the selection condition, it is less likely that a contradiction exists and the operation can be pruned. Figure 9.16 shows the pruning effectiveness as the number of predicate attributes (P) increases. As the number of predicate attributes increases, fewer operations are pruned. It is important to note that in some cases we are not able to prune all unsuspicious operations using the static pruning condition. In these cases, even though the delta expressions intersect

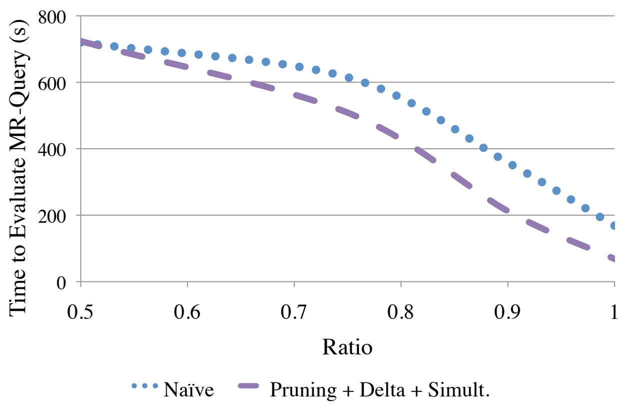


Figure 9.20: Ratio Impacts Performance(10% PM, 0.1% Sel., 250K Rows, P=1)

the selection condition (statically), there does not exist a row in the specific database instance that is affected by the misconfiguration.

9.11.2 Pruning (With Updates)

When updates are considered, in addition to measuring the runtime performance of evaluating the MR-query, we also counted the number of logged operations that were suspicious, unsuspicious, and pruned. Figure 9.17 shows these statistics when the log contains updates. As expected, for small misconfigurations, we are able to prune a large portion of the operations. As the misconfiguration gets larger, fewer operations are pruned.

While pruning reduces the number of operations that must be re-executed, as mentioned in Section 9.10.3, the cost of re-executing an operation using delta tables is larger than re-executing an operation without delta tables because there is an extra cost to construct the table $\widehat{S}$, which is formed by removing the rows in Δ_S^- from S and appending the rows from Δ_S^+. Thus, as the size of the delta tables grows, the cost of re-executing an operation increases. Figure 9.18 shows the number of rows in the database after using the naive method, which includes the rows in S and $\widehat{S}$, the number of rows contained in the delta tables, and the total number of rows when the delta tables are used, which includes S, Δ_S^-, Δ_S^+ and rows used to store the results of update operations that cannot be pruned. As expected, for small misconfigurations, the delta tables are small and the extra cost of constructing $\widehat{S}$ is small (see Figures 9.13 and 9.14). In contrast, for larger misconfigurations (50%), the delta tables are large, which slows down the performance of the pruning method (Figure 9.19); in this case, it is more efficient to evaluate the MR-query using the naive method.

The select to update ratio can impact the performance of evaluating an MR-query. Figure 9.20 shows the time to evaluate an MR-query for the naive method, and the pruning with delta tables plus simultaneous re-execution method for a range of ratios. When the log contains many updates (i.e., has a low ratio), the performance of both methods worsens due to the higher cost of writing to the database than reading. Additionally, a larger number of updates typically will increase the size of the delta tables, resulting in fewer operations being pruned and increasing the time to re-execute a query. For a ratio of 0.5, the naive method and pruning method have similar performance for the specified set of parameters. As the ratio gets larger (fewer updates), the pruning method outperforms the naive approach.

The challenge is to determine, before the MR-query is evaluated, which method (naive plus simultaneous re-execution, or pruning with delta tables plus simultaneous re-execution) is the most efficient given the workload parameters. We believe this decision can be made by developing a cost-based optimizer. One promising approach would use the database's query optimizer to estimate the cost of re-executing each logged operation using each of the two methods; given the total estimated cost, the appropriate method can be chosen.

9.11.3 Partial Re-Execution

We observed that pruning with delta tables as described by Theorem IX.11 captures many of the cases where partial re-execution is applicable. Intuitively, pruning with delta tables is a form of partial re-execution where the query plan is cut across the leaf selection conditions. However, we did find a few cases where partial re-execution did provide additional performance benefits. Particularly, partial re-execution is beneficial when the selection condition contains a clause that is composed of a disjunction of literals that references multiple tables. In some cases, new or old rows may not be filtered by the conjunction of clauses that only refer to a single table, but are then filtered out by the clause that references multiple tables. In such a case, partial re-execution can improve re-execution performance by only re-executing a subset of the query plan.

We used the rudimentary Left-Deep Query Plan Analyzer as described in Section 9.8 to determine what cuts in the query plan should be tested. We crafted a workload of an update operation and a query that would benefit from partial re-execution. The query had three tables R, S and T, and a selectivity of 1% for each table. The query plan analyzer determined that the table T could be removed so that the sub-query of R and S only needed to be evaluated. For this example, we found that partial re-execution reduced the run time of the query by 28%.

9.12 Related Work

While previous auditing work is related to this thesis, it does not consider the flow of information via updates. For example, if a record r is copied from one location to another location, say r', the auditor does not understand this. Thus, an audit on r would fail to retrieve SQL queries to which r' was indispensable. This thesis addresses this problem cleanly through the definition of MR-queries, which conceptually replay the entire database history, including updates, and retrieve all queries whose results have changed.

There are other variations of the data auditing problem that have been studied in the past. Lu and Miklau consider auditing a database under data retention restrictions [55]; this work is concerned only with database updates, not auditing queries (i.e., SELECT statements). Hasan et al. analyze efficient approaches to support term-immutable databases for compliance purposes [40]. Agrawal et al. consider auditing disclosure by relevance ranking [3]: Given a sensitive table that has been "leaked" from an organization, and a log of past SQL queries, determine which queries were most likely to have been the source of the leak.

Recent work has also focused on the problem of recovering from malicious database transactions [9, 20, 53, 54]. At a high level, if a committed transaction is discovered to have been malicious, its effects, and the effects of those transactions that depended on it, must be undone. One important difference between this and our approach is the level at which we reason about data dependencies. In transaction theory, a transaction T_2 is usually said to depend on another transaction T_1 if it reads a data object (e.g., a tuple) written by T_1. When defining the semantics of MR-queries, we are operating at a higher level of abstraction; notably, a query Q may read a tuple that was updated by some other command, but unless that tuple changes the result of Q, it is not considered to have influenced Q. Thus, while one might suggest taking a "transactional" approach to our problem (i.e., by tracking reads and writes), this approach would likely lead to larger result sets for MR-queries. Further, we consider it desirable to define the semantics on MR-queries based only on the *syntax* of the logged operations, rather than the specific plans used to execute them. Notice, for example, that the tuples read by a query vary based on the plan (e.g., scan vs. index lookup), which would affect the MR-query result if we were to take a transactional approach.

9.13 Summary

In this chapter, we introduced the PolicyReplay framework for responding to database access control misconfigurations. One of the critical components of this framework is the misconfiguration-response (MR) query, which retrieves those queries that may have revealed unauthorized information. The naive algorithm for evaluating MR-queries can be expensive. Thus, we have developed and evaluated a suite of techniques (including pruning, delta tables, partial re-execution, and simultaneous re-execution) for improving the performance of this operation.

CHAPTER X

Conclusion and Future Work

The accountability and security of electronic medical records is increasingly important. Recent U.S. legislation such as the Affordable Care Act, HIPAA and HITECH outline guidelines for the appropriate use of personal health information. Unfortunately, current technologies do not meet the requirements of these regulations. In particular, because of the dynamics of patient care, employees are able to access any patient's medical record, even though they may not be involved in the patient's treatment. While audit logs record all accesses to personal health information and compliance officers monitor the logs, there are too many accesses for compliance officers to monitor given their limited auditing budget.

This thesis presented the explanation-based auditing system to improve the accountability and security of personal health information and improve compliance officer efficiency. The main insight was that most appropriate accesses to medical records occur for valid clinical or operational reasons in the process of treating a patient, while inappropriate accesses do not. Therefore, if the auditing system can determine the reason why an access occurred, the access is likely appropriate and does not need to be reviewed further by the compliance officer. Intuitively, the auditing system serves as a filter on the audit log so the compliance officers have a much smaller subset of suspicious accesses to review.

The challenge is then how to capture these clinical or operational reasons for access. Chapter V described how explanations for access are represented as paths connecting the patient whose record was accessed to the employee accessing the patient's record. Interestingly, this model is successful because medical databases already store a large amount of patient data such as appointments, medication orders and X-rays that describe how patients are treated, and can be used to construct these paths. Moreover, Chapter V presented algorithms to automatically mine these explanations directly from the data, rather than requiring the compliance officer to manually specify them.

Unfortunately, many appropriate accesses cannot be explained using data explicitly stored in the database. In particular, databases are often missing data that, if filled-in, could be used to explain additional accesses. For example, patients make appointments with doctors, not nurses, which makes it difficult to explain why nurses access medical records. Chapter VI presented two approaches to fill-in missing types of data. Specifically, algorithms described how to discover collaborative working relationships between employees, and extract which diagnoses each department is responsible for treating. While adding these missing types of data improved the results of the auditing system, there are still opportunities to enhance the database's completeness.

An experimental evaluation using data from the University of Michigan Health System demonstrated that the explanation-based auditing system is capable of filtering over 94% of the accesses in the audit log. Unfortunately, a large number of suspicious accesses remain, which is still too large for compliance officers to review. In particular, broadly functional departments such as the Central Staffing, Pharmacy and Radiology Departments account for a large proportion of the unexplained accesses. Therefore, to best utilize compliance officers' time, Chapter VII presented techniques to order accessed by their suspiciousness so that the most suspicious accesses are reviewed first. The main observation here is that these broadly functional departments' accesses are temporally dependent on other events occurring in the hospital. This dependency can then be used to construct a suspiciousness metric to order accesses for review.

To close-the-loop and verify that the explanation-based auditing system accurately classifies accesses and improves compliance officers' efficiency, Chapter VIII described the implementation of a prototype auditing system. The prototype allows compliance officers to extract treatment information from a commercial EMR system, load the data into the prototype, create and mine explanations and then search for inappropriate behavior. The University of Michigan Health System's Compliance Office will test the prototype with previously known instances of inappropriate use to evaluate the practical benefits of the auditing system.

Lastly, this thesis explored issues related to DBMS-level auditing. Specifically, while employees often access data through restrictive user interfaces (that only allow access to a single medical record at a time), they can also execute ad-hoc SQL queries to analyze patient populations. In this case, the audit log records the SQL query's text rather than (employee, patient) pairs, which introduces ambiguity with respect to which patients' records were accessed. Chapter IX described the problem of tracking accesses at the DBMS level, particularly in the case of access control policy misconfigurations.

Moving forward, there are many opportunities to improve the explanation-based auditing system. First, the auditing system has thus far relied on de-identified patient data for its analysis. However, if patient privacy can be ensured, there are opportunities to incorporate clinical notes into the auditing system. Specifically, clinical notes can provide more details about a patient's treatment than lists of hospital events. In particular, clinical notes describe the process of a patient's treatment, reference other caregivers involved in the patient's care (e.g., consulting physicians) and note the patient's progress in raw text. Natural language processing techniques can be applied to clinical notes to extract additional facts about a patient's treatment (e.g., Alice was referred to Dr. Bob), which can be incorporated naturally into the auditing system.

Second, electronic medical records usage is likely to increase as the transition from paper to digital medical records completes. As a result, audit logs will record more accesses and the auditing system will need to scale accordingly. The explanation-based auditing system relies on a relational database for much of its computation, which has the potential to scale. However, there are opportunities to explore if distributed, parallel processing architectures such as Hadoop MapReduce [38] can more efficiently process the audit log.

BIBLIOGRAPHY

BIBLIOGRAPHY

[1] Personal communication with the University of Michigan Health System Compliance Office.

[2] Public Company Accounting Reform and Investor Protection Act of 2002 (Sarbanes-Oxley), 2002. Pub.L. 107-204, 11 Stat. 745.

[3] R. Agrawal, A. Evfimievski, J. Kiernan, and R. Velu. Auditing disclosure by relevance ranking. In *SIGMOD*, 2007.

[4] R. Agrawal and R. Srikant. Fast algorithms for mining association rules. In *VLDB*, pages 487–499, 1994.

[5] R. Agrawal and R. Srikant. Mining sequential patterns. In *ICDE*, pages 3–14, 1995.

[6] Rakesh Agrawal, Roberto J. Bayardo, Christos Faloutsos, Jerry Kiernan, Ralf Rantzau, and Ramakrishnan Srikant. Auditing compliance with a Hippocratic database. In *VLDB*, pages 516–527, 2004.

[7] Rakesh Agrawal, Jerry Kiernan, Ramarkishnan Srikant, and Yirong Xu. Hippocratic databases. In *VLDB*, 2004.

[8] P. V. Asaro, R. L. Herting, A. C. Roth, and M. R. Barnes. Effective audit trails–a taxonomy for determination of information requirements. *AMIA*, pages 663–665, 1999.

[9] K. Bai and P. Liu. Data damage tracking quarantine and recovery (DTQR) scheme for mission-critical database systems. In *EDBT*, 2009.

[10] John Barkley, Konstantin Beznosov, and Jinny Uppal. Supporting relationships in access control using role based access control. In *In Proceedings of the Fourth ACM Workshop on Role-Based Access Control*, pages 55–65, 1999.

[11] Deepavali Bhagwat, Laura Chiticariu, Wang Chiew Tan, and Gaurav Vijayvargiya. An annotation management system for relational databases. In *VLDB*, 2004.

[12] Aziz A. Boxwala, Jihoon Kim, Janice M. Grillo, and Lucila Ohno-Machado. Using statistical and machine learning to help institutions detect suspicious access to electronic health records. *JAMIA*, 18:498–505, 2011.

[13] Peter Buneman, Sanjeev Khanna, and Wang chiew Tan. Why and where: A characterization of data provenance. In *ICDT*, pages 316–330, 2001.

[14] Cerner. http://www.cerner.com/, 2013.

[15] Varun Chandola, Arindam Banerjee, and Vipin Kumar. Anomaly detection: A survey. *ACM Computing Surveys*, 41, 2009.

[16] Adriane Chapman and H. V. Jagadish. Why not? In *SIGMOD*, pages 523–534, 2009.

[17] You Chen and Bradley Malin. Detection of anomalous insiders in collaborative environments via relational analysis of access logs. CODASPY, pages 63–74, 2011.

[18] You Chen, Steve Nyemba, Wen Zhang, and Bradley Malin. Specializing network analysis to detect anomalous insider actions. *Security Informatics*, 1:1–24, 2012.

[19] James Cheney, Laura Chiticariu, and Wang Chiew Tan. Provenance in databases: Why, how, and where. *Foundations and Trends in Databases*, 2009.

[20] T. Chiueh and D. Pilania. Design, implementation, and evaluation of a repairable database management system. In *ICDE*, 2005.

[21] Barbara Cotter. Peeking at medical records an issue for health centers. *Colorado Springs Gazette*, 2011.

[22] Nilesh Dalvi, Pedro Domingos, Mausam, Sumit Sanghai, and Deepak Verma. Adversarial classification. In *KDD*, 2004.

[23] Department of Health and Human Services. Modifications to the HIPAA Privacy, Security, Enforcement, and Breach Notification Rules Under the Health Information Technology for Economic and Clinical Health Act and the Genetic Information Nondiscrimination Act. http://www.gpo.gov/fdsys/pkg/FR-2013-01-25/pdf/2013-01073.pdf, 2013.

[24] Department of Homeland Security. Handbook for Safeguarding Sensitive Personally Identifiable Information. http://www.dhs.gov/xlibrary/assets/privacy/privacy_guide_spii_handbook.pdf, 2011.

[25] Django: The Web framework for perfectionists with deadlines. www.djangoproject.com, 2013.

[26] Epic. http://www.epic.com/, 2013.

[27] Daniel Fabbri and Kristen LeFevre. Explanation-based auditing. In *PVLDB*, volume 5, pages 1–12, 2011.

[28] Daniel Fabbri and Kristen LeFevre. Explaining accesses to electronic medical records using diagnosis information. *Journal of the American Medical Informatics Association*, 2012.

[29] Daniel Fabbri, Kristen LeFevre, and David A. Hanauer. Explaining accesses to electronic health records. In *Proceedings of the 2011 Workshop on data mining for medicine and healthcare*, pages 10–17, 2011.

[30] Daniel Fabbri, Kristen LeFevre, and Qiang Zhu. PolicyReplay: Misconfiguration-Response Queries for Data Breach Reporting. In *VLDB*, pages 36–47, 2010.

[31] Daniel Fabbri, Ravi Ramamurthy, and Raghav Kaushik. SELECT triggers for data auditing. In *ICDE*, 2013.

[32] Ana Ferreira, Ricardo Cruz-Correia, Luis Antunes, and David W Chadwick. Access control: How can it improve patients healthcare? *Studies In Health Technology And Informatics*, 127:65–76, 2007.

[33] Stephanie Forrest, Steven A. Hofmeyr, Anil Somayaji, and Thomas A. Longstaff. A sense of self for unix processes. In *Proceedings of the IEEE Symposium on Security and Privacy*, 1996.

[34] Santo Fortunato. Community detection in graphs. *Physics Reports*, 486(3-5):75 – 174, 2010.

[35] M. R. Garey and David S. Johnson. *Computer and Intractability: A Guide to the Theory of NP-Completeness.* W. H. Freeman, 1979.

[36] Christos K. Georgiadis, Ioannis Mavridis, George Pangalos, and Roshan K. Thomas. Flexible team-based access control using contexts. In *Proceedings of the sixth ACM symposium on Access control models and technologies*, pages 21–27, 2001.

[37] Carl A. Gunter, David M. Liebovitz, and Bradley Malin. Experience-based access management: A life-cycle framework for identity and access management systems. *IEEE Security and Privacy*, 9:48–55, 2011.

[38] Hadoop. http://hadoop.apache.org, 2013.

[39] Jiawei Han, Jian Pei, and Yiwen Yin. Mining frequent patterns without candidate generation. In *SIGMOD*, pages 1–12, 2000.

[40] Ragib Hasan and Marianne Winslett. Efficient audit-based compliance for relational data retention. In *Proceedings of the 6th ACM Symposium on Information, Computer and Communications Security*, ASIACCS '11, pages 238–248, 2011.

[41] Scott Hensley. Snooping Tucson hospital workers fired in records breach. *NPR.org*, 2011.

[42] R.L. Herting Jr, P.V. Asaro, A.C. Roth, and M.R. Barnes. Using external data sources to improve audit trail analysis. *AMIA*, pages 795–799, 1999.

[43] Healthcare Information and Management Systems Society. Break the glass. www.himss.org/asp/ContentRedirector.asp?ContentID=75628, 2009.

[44] C. S. Jensen, L. Mark, and N. Roussopoulos. Incremental implementation model for relational databases with transaction time. *IEEE Transactions on Knowledge and Data Engineering*.

[45] R. Kaushik and R. Ramamurthy. Efficient auditing for complex SQL queries. In *SIGMOD*, pages 697–708, 2011.

[46] Glenn Kessler. Two fired for viewing Obama passport file. *Washington Post*, March 21 2008.

[47] D. Kiely. SQL Server 2008 Security Overview for Database Administrators. SQL Server Technical Article, October 2007.

[48] Christopher Kruegel and Giovanni Vigna. Anomaly detection of web-based attacks. In *Proceedings of the 10th ACM conference on Computer and communications security*, CCS '03, pages 251–261, 2003.

[49] Wenke Lee and Salvatore J. Stolfo. A framework for constructing features and models for intrusion detection systems. *ACM Trans. Inf. Syst. Secur.*, 3(4):227–261, 2000.

[50] K. LeFevre, R. Agrawal, V. Ercegovac, R. Ramakrishnan, Y. Xu, and D. DeWitt. Limiting disclosure in hippocratic databases. In *VLDB*, 2004.

[51] Xiaowei Li, Yuan Xue, and B. Malin. Towards understanding the usage pattern of web-based electronic medical record systems. In *IEEE Symposium on a World of Wireless, Mobile and Multimedia Networks*, pages 1–7, 2011.

[52] Xiaowei Li, Yuan Xue, and Bradley Malin. Detecting anomalous user behaviors in workflow-driven web applications. In *SRDS*, pages 1–10, 2012.

[53] P. Liu, P. Ammann, and S. Jajodia. Rewriting histories: Recovering from malicious transactions. *Distributed and Parallel Databases*, 8:7–40, 2000.

[54] D. Lomet, Z. Vagena, and R. Barga. Recovery from bad user transactions. In *SIGMOD*, 2006.

[55] W. Lu and G. Miklau. Auditing a database under retention restrictions. In *ICDE*, 2009.

[56] Bradley Malin, Steve Nyemba, and John Paulett. Learning relational policies from electronic health record access logs. *Journal of Biomedical Informatics*, 44(2):333–342, 2011.

[57] Marianne Kolbasuk McGee. Case study: Catching health data snoops. *Data Breach Today*, 2012.

[58] Prateek Mittal, Vern Paxson, Robin Sommer, and Mark Winterrowd. Securing mediated trace access using black-box permutation analysis. *HotNets*, 2009.

[59] R. Motwani, S. Nabar, and D. Thomas. Auditing SQL queries. In *ICDE*, pages 287–296, 2008.

[60] A. Nanda. Fine-grained auditing for real-world problems. *Oracle Magazine*, 2004.

[61] M. Newman. Analysis of weighted networks. *Phys. Rev. E*, 70, 2004.

[62] Charles Ornstein. Hospital to punish snooping on Spears. *Los Angeles Times*, 2008.

[63] Debprakash Patnaik, Patrick Butler, Naren Ramakrishnan, Laxmi Parida, Benjamin J. Keller, and David A. Hanauer. Experiences with mining temporal event sequences from electronic medical records: initial successes and some challenges. In *KDD*, pages 360–368, 2011.

[64] Raluca Ada Popa, Catherine M. S. Redfield, Nickolai Zeldovich, and Hari Balakrishnan. Cryptdb: protecting confidentiality with encrypted query processing. 2011.

[65] PostgreSQL. http://www.postgresql.org/, 2013.

[66] PracticeFusion. www.practicefusion.com/, 2013.

[67] R. Reeder, L. Bauer, L. Cranor, M. Reiter, K. Bacon, K. How, and H. Strong. Expandable grids for visualizing and authoring computer security policies. In *CHI*, 2008.

[68] S. Rizvi, A. Mendelzon, S. Sudershan, and P. Roy. Extending query rewriting techniques for fine-grained access control. In *SIGMOD*, pages 551–562, 2004.

[69] L. Rostad and O. Edsberg. A study of access control requirements for healthcare systems based on audit trails from access logs. In *Computer Security Applications Conference*, pages 175 –186, 2006.

[70] P. Roy, S. Seshadri, S. Sudarshan, and S. Bhobe. Efficient and extensible algorithms for multi-query optimization. In *SIGMOD*, 2000.

[71] G. Russello, Changyu Dong, and N. Dulay. A workflow-based access control framework for e-health applications. In *22nd International Conference on Advanced Information Networking and Applications - Workshops*, pages 111–120, 2008.

[72] J Salazar-Kish, D Tate, P D Hall, and K Homa. Development of CPR security using impact analysis. *AMIA*, pages 749–53, 2000.

[73] R. S. Sandhu, E. J. Coyne, H. L. Feinstein, and C. E. Youman. Role based access control models. *IEEE Computer*, 29:38–47, 1996.

[74] Sunita Sarawagi, Shiby Thomas, and Rakesh Agrawal. Integrating association rule mining with relational database systems: alternatives and implications. In *SIGMOD*, pages 343–354, 1998.

[75] Charlie Savage. Soldier Admits Providing Files to WikiLeaks. *New York Times*, 2013.

[76] J. F. Traub, Y. Yemini, and H. Wozniakowski. The statistical security of a statistical database. *ACM Transactions on Database Systems (TODS)*, 9(4):672–679, 1984.

[77] University of Michigan Health System, 2012.

[78] W. van der Aalst, B. Vandongen, J. Herbst, L. Maruster, G. Schimm, and A. Weijters. Workflow mining: A survey of issues and approaches. *Data and Knowledge Engineering*, pages 237–267, 2003.

[79] T. J. Wasserman. DB2 UDB Security Part 5: Understanding the DB2 Audit Facility, March 2006.

[80] Rui Xu and D. Wunsch II. Survey of clustering algorithms. *IEEE Transactions on Neural Networks*, 16:645–678, 2005.

[81] Yan Yan, Glenn Fung, Jennifer G. Dy, and Romer Rosales. Medical coding classification by leveraging inter-code relationships. In *International conference on knowledge discovery and data mining*, pages 193–202, 2010.

[82] Guangsen Zhang and Manish Parashar. Context-aware dynamic access control for pervasive applications, 2004.

[83] Wen Zhang, Carl A. Gunter, David Liebovitz, Jian Tian, and Bradley Malin. Role prediction using electronic medical record system audits. In *Proceedings of the 2nd USENIX conference on Health security and privacy*, 2011.

[84] Y. Zhong, P. Deshpande, J. Naughton, and A. Shukla. Simultaneous optimization and evaluation of multiple dimensional queries. In *SIGMOD*, 1998.