Eindhoven University of Technology

MASTER

Database data leakage detection systems
a framework for comparison and analysis

van Enckevort, G.

Award date:
2014

Link to publication

Disclaimer
This document contains a student thesis (bachelor's or master's), as authored by a student at Eindhoven University of Technology. Student theses are made available in the TU/e repository upon obtaining the required degree. The grade received is not published on the document as presented in the repository. The required complexity or quality of research of student theses may vary by program, and the required minimum study period may vary in duration.

General rights
Copyright and moral rights for the publications made accessible in the public portal are retained by the authors and/or other copyright owners and it is a condition of accessing publications that users recognise and abide by the legal requirements associated with these rights.

- Users may download and print one copy of any publication from the public portal for the purpose of private study or research.
- You may not further distribute the material or use it for any profit-making activity or commercial gain

Technische Universiteit **Eindhoven** University of Technology

Master's thesis

Database Data Leakage Detection Systems: a Framework for Comparison and Analysis

Author:
Gijs van Enckevort BSc

Graduation Supervisor:
Prof. Dr. Milan Petković
Graduation Tutor:
Elisa Costante MSc

Eindhoven, 24 February 2014

Abstract

Data leakage and data misuse cause significant losses for companies each year, in terms of monetary cost as well as in reputation damages. One way to prevent or discover data leakage and data misuse is to use an intrusion detection system. Extensive academic research has been done in this field. In this thesis we focus on database intrusion detection systems. In particular, anomaly based approaches to data leakage, which have the great theoretical advantage of detecting known and unknown attacks, are analysed in this thesis.

The objectives of this thesis are threefold: 1) creating a framework to benchmark existing solutions; 2) formulating both normal and misuse scenarios verified by experts for proper testing; and 3) exploring the effects of richer feature spaces on the detection rates and false positive rates.

To create a benchmark framework we used GNU health, an open source medical information system. The underlying architecture was modified to add a logging class which can intercept and log queries to and responses from the internal database. Normal behaviour and attack scenarios were then formulated with the help of domain experts. These scenarios were then simulated in GNU health, intercepted by the logger and parsed into realistic data sets. Finally these data sets were used to test, analyse and compare interesting existing solutions in a fair way by using Rapidminer, a visual tool for predictive analysis. This environment ensures that all simulations are processed and analysed in a consistent manner when testing different approaches and while simulating different scenarios. The formulated scenarios can be simulated in any hospital administration system. Further more, such a collection of attack scenarios, when complete and accurate, could form a basis for unified testing of database data leakage detection approaches in the future.

Anomaly based database intrusion detection solutions usually work by building profiles that define normal behaviour per user or role and subsequently flagging deviations from those profiles as attacks. Normal behaviour profiles are built over a set of features which represent important characteristics or attributes of database transactions. The features used for detection by different approaches from literature can be divided into three types: features about the query text (syntax based features), features about the result set (result set based features) and features about the context of the interaction (context based features). We compare three approaches from literature: one consisting only of syntax based features, one consisting of a combination of syntax based and context based features and one consisting of a combination of all three types of features.

This work also tries to determine the effect of a richer feature space on the detection rates and false positive rates. This was done by using different feature spaces for the same attacks and observing differences in performance. Our results show better false positive rates for the approach which uses all types of features. This approach is also able to detection a wider set of attacks.

Keywords: anomaly detection, data leakage, databases, intrusion detection

Contents

Chapter 1

Introduction

Every company uses data in some shape or form, for example when managing stocks, designing new products or scheduling work. Some companies even have a core business in the analysis of data. In addition, most companies also process personal or otherwise sensitive data, for example contact details about their customers for shipping or personal information about their employees for salary administration.
Companies have good reasons to keep their data safe: competitors might be interested in a company's new products; suppliers might want to know who their competitors are; and advertising companies might want to get their hands on email addresses to send their adverts to.
The (un)intentional disclosure or revelation of a company's data to unwanted (and probably malicious) recipients is commonly referred to as *data leakage* [1]. Protecting companies from facing a data leakage is essential to reduce the enormous costs data leakages can cause in terms of money, legal issues and reputation [2]. In a time when digital espionage and cyber crime are happening on a daily basis, the task of securing data is becoming increasingly harder.
Besides the threat of a data leakage caused by a cyber attack coming from an external party, organisations also have to deal with insider threats to data leakage. An insider threat can be represented by individuals within the organisation (e.g. employees) who cause harm due to their knowledge of internal procedures and resources. An actual insider attack is usually performed by employees, former employees, contractors or business associates who abuses their position to either steal data from the organisation or sabotage computer systems. When an employee abuses their access rights, the term *data misuse* is often used. When this data misuse is used in order to steal data, we speak of *data leakage*.

An example of data leakage is the theft by a senior IT technician of the NDB, the national intelligence service of Switzerland[1]. The employee became disgruntled as he felt that his advice on operating the data systems was not being taken seriously. In retaliation, the technician downloaded hundreds of thousands of printed pages containing information on counter-terrorism, totalling several terabytes of data, and carried it out of the office on portable hard drives. The technician had, from his function, administrative rights in the system, giving him unrestricted access to most of the service's data.
A second example is the theft of the laptop of a Home Depot employee in 2007[2]. The employee

[1]Reuters, "Swiss spy agency warns U.S., Britain about huge data leak", `http://www.reuters.com/article/2012/12/04/us-usa-switzerland-datatheft-idUSBRE8B30ID20121204`, last accessed 16 December 2013

[2]CNET, "Stolen Home Depot laptop exposes employee data", `http://news.cnet.com/8301-10784_3-9799644-7.html`, last accessed 16 December 2013

took the laptop, containing unencrypted personal information of roughly 5,600 coworkers, to do some additional work at home when it was stolen from his vehicle[3]. The target of the theft was most likely the laptop itself but this case shows how easy it can be to have large amounts of (personal) data leaked in a seemingly unrelated incident.
A third example is known as the largest data security breaches in history to date[4]: the attack on the Sony Playstation Network in April 2011. A database containing personal data of approximately 77 million users was accessed by an external party over the course of three days. The attack, disguised as a purchase, was exploiting a known vulnerability which was not dealt with yet[5].
The aforementioned examples are only three of thousands of incidents that happen every year around the globe. The scope of the problem becomes even clearer from surveys and research in this subject. A study by the Ponemon Institute [2] approximated that in 2012, data breaches cost companies an average of 136 USD per (leaked) record. This figure, however, varies greatly between countries, with an average of 199 USD in Germany being the most costly and India with 42 USD being the least costly. The total cost of data breaches was the highest in the US, whereas Brazil had the lowest total cost of the included countries, with 5.4 million and 1.1 million USD respectively. The same study reports that the average number of breached records per attack was 23,647. Note that no attacks in which more then 100,000 records were compromised were included in this number, as those incidents are not representative of general data breaches and could skew results.
In 2008, InsightExpress conducted a study [3] that exemplifies the omnipresence of the issue. Only 16 percent of employees in France claim to adhere to security policies all the time, employees in India admitted to changing IT security settings on business computers so they can view unauthorised websites and some Chinese employers even decided to audit all company computers for unauthorised content because of the high IT abuse levels at the time. This shows how all over the world, companies are susceptible to data leakage.

This work deals with the problem of data leakage, with a specific focus on leakages which happen at a database level. Since storing and retrieving data from the structured framework that is a database is very convenient, stealing that data can also be a very quick and easy process. A simple query can retrieve a large amount of information from the database at once. The breach on the Sony Playstation Network shows how big these data leaks can become.

[3]`http://doj.nh.gov/consumer/security-breaches/documents/home-depot-20070502.pdf`, last accessed 16 December 2013

[4]Reuters, "Sony Playstation suffers massive data breach", `http://www.reuters.com/article/2011/04/26/us-sony-stoldendata-idUSTRE73P6WB20110426`, last accessed 16 December 2013

[5]CNET, "The Playstation Network Breach (FAQ)", `http://news.cnet.com/8301-31021_3-20058950-260.html`, last accessed 16 December 2013

1.1 General Approaches

One way of detecting data leakage, or data misuse in general, is by using an Intrusion Detection System (IDS). Intrusion detection systems can be employed detect several kinds of computer security violations, including:

- attacks from the outside, such as exploits;
- attackers who assume the identity of a legitimate user and try to access the underlying system;
- and insiders attempting to abuse their privileges

In the context of data leakage detection in databases, such a system tries to detect attacks on or misuse inside the database system. It does so by looking at incoming database requests and determining whether the user's actions are considered an attack or not based on some criteria.
In general, two major kinds of IDSs can be distinguished [4]: *signature based* and *anomaly based* detection systems. Database intrusion detection systems, which are the focus of this work, are a specific kind of IDS for which the distinction between signature and anomaly based systems also applies.

1.1.1 Signature Based Detection Systems

In signature based detection systems, all undesirable behaviour is defined in rules or signatures. The system is then able to detect activities compliant with these signatures as intrusions.
As long as the signatures that specify the attacks are accurate and complete, this type of system can be very effective and efficient in detecting attacks. The amount of false positives, (i.e. the amount of normal actions falsely declared an attack) in such a system is usually quite low because it was specifically trained to detect attacks and the signature of normal behaviour is usually significantly different from that of an attack. Another advantage of signature based detection systems is that they tell the security officer , who has to resolve the issue, what kind of attack was detected. This gives the security officer insight into what happened and also makes it possible for him or her to determine whether the attack was real or actually a mistake by the system. The speed and effectivity of signature based intrusion detection systems makes them good candidates for many real-time systems such as virus scanners.

The main issues with signature based intrusion detection systems are threefold: detecting unknown attacks, coping with setting specific parameters, and detecting masked attacks. Each of these problems will be explained below.
The first and probably the most obvious issue with signature based systems is their inability to detect unknown attacks. If the system has learned no signature for an attack, then that attack simply cannot be detected. A result of this limitation is that, even assuming perfect knowledge of all attacks at the moment of deployment, the system would start running behind in knowledge of attacks from the moment it was introduced. A solution to this problem would be to update the signature database frequently in order to keep it up to date. A delayed update could have significant consequences, similar to the main issue with the Sony database breach. Any update to such a database containing normal activity marked as an attack could have large consequences as well, as they have had before with virus scanners[6].

[6]Mary Landesman, "Six Most Memorable Antivirus Mistakes", `http://www.antivirus.about.com/od/whatisavirus/tp/antivirusmistakes.htm`, accessed last on 16 December 2013

A second issue is setting specific parameters. Many different database systems exist, and they all have subtle (and not so subtle) differences. In addition, a lot of different applications for accessing databases exist who all might formulate their queries in a different way. Both these facts contribute to the fact that deploying a signature based detection system requires fine tuning to those specifics in order to function properly.
Finally, we have the issue of masked attacks. SQL and its dialects have many ways of expressing the same query, even in one system there are multiple ways of formulating the same query. Across different dialects the problem grows even bigger. For example, the word *SELECT* can be constructed in many ways, depending on the dialect.

- Oracle: *'SEL'* || *'ECT'*
- MS-SQL: *'SEL'+'ECT'*
- MySQL: *'SEL''ECT'*
- SQL Server: *CHAR(83) + CHAR(69) + CHAR(76) + CHAR(69) + CHAR(67) + CHAR(84).*

This means that, to deal with the great flexibility of the SQL language, many signatures have to be created. Otherwise even small variations of a known attack may let malicious activities to pass unnoticed.

1.1.2 Anomaly Based Detection Systems

Anomaly based detection systems take a rather different approach compared to signature based detection systems. In the training/set up phase, the system's normal behaviour is defined by learning it either from a set of rules or from previous system activity which has been verified by a security expert in the field where the activity originated from. Normal behaviour is usually defined over a set of features, representing characteristics of the interaction of the user with the database. These features might include the query text, the result set, the time of issuing or the location of the issuer. These features together form a feature space which can be very important for detecting certain types of attacks. During the detection phase (i.e. after the system has been deployed), every new incoming request is broken down in to these features which are then compared to the known normal behaviour. If the request does not fit the known pattern for normal behaviour, it is considered an anomaly and an alarm will be raised.
Often normal behaviour is not classified in general, but per user or per (logical) role in the system. The request is then compared to the normal behaviour of that user or role, and subsequently classified as normal or anomalous for that user or role. A second approach is to hide the actual user or role and let the intrusion detection system predict what user or role most likely sent the request. If the prediction is correct, the activity is deemed normal, otherwise an alarm is raised.
An advantage of anomaly based approach is that unknown or new attacks can just as easily be detected as known or old attacks because the information in the system only knows what normal behaviour looks like. This makes the approach very promising as it has the potential to detect every type of attack.

Unfortunately, a number of disadvantages exist for currently existing database anomaly detection systems as well. These include a relatively high false positive rates due to the difficulty in capturing all normal behaviour adequately during the training phase, overlap in activity between roles or users which increases false positive rates, the lack of capabilities in coping with quickly changing environments and the low amount of feedback that is given about why an alarm was

raised.
One of the more difficult issues with the anomaly based detection systems is that all normal behaviour must be trained before the system can be deployed. This is because all behaviour that has not been learned by the system will cause an alarm to be raised, and acceptable behaviour that has not been learned will therefore cause false alarms. This might seem similar to how all attacks must be learned beforehand for the signature based approach, but learning all normal behaviour might be more difficult achieve. Consider the example of an emergency situation in a hospital, now the actions of some users will possibly change compared to the average day. However, this should not cause alarms in a detection system since it is still considered acceptable behaviour. Training for all such possible cases might be almost impossible. In addition, people do not always behave exactly the same every day. Small deviations might have not been accounted for during the training phase which can again lead to more false positives. A high false positive rate is undesirable because every alarm (including false positives) has to be addressed by a security officer.
If we would capture all normal behaviour in a single profile, the system would be unable to recognise which users do what in the system. This means that all employees who behave like other employees (including accessing sensitive data they should not) will go undetected. The only suspicious activity would be those actions that no employee ever does. This approach is clearly insufficient for most organisations. Note that this approach also makes it easier for outsiders to masquerade themselves as employees in order to access (valuable) data of the organisation undetected.
By profiling normal behaviour per role or per user, we eliminate the issue we have with general classification. In this case, the role of an employee can be their function inside the organisation or a role based on access rights, such as in a Role Based Access Control administered system. The most common method of using these role or user profiles is to predict the role or user of an incoming request based on the profiles. If the user or role matches the actual issuer, we consider the request authentic and allow it. If the prediction classifies the action as more likely to be issued by some other user or role, an alert is raised. A problem that arises is when everyday activity of roles or users overlap. For example, a team manager of a research team might perform the same actions as the researchers of the team, but with some added scheduling or other administrative tasks. Now the normal research activity is seen (and has been learned before) for both roles. When the system sees a research activity request it will classify the originating role as either researcher or research team leader, marking the research activity of the other role as an attack. This means that a researcher performing his daily actions might cause a false alert in the system. If we give the same role to both users, we will fail to detect management-specific actions performed by researchers (the problem with a general approach). The same problem exists in overlap of activity between users. In general, the best case scenario for these approaches is to have roles or users with mutually exclusive activity. An alternative solution is, if the underlying system allows for it, to use a hierarchy of roles such as proposed by Wu et al. [5].
The third issue with anomaly based database detection systems is that any changes in role or user activity (reorganisations, promotions, etc) necessitates complete relearning of normal activity for that user or role. This is a laborious and difficult task, and especially inconvenient if such an event occurs frequently.
The final issue we will discuss is the amount of feedback given to the security officer who has to resolve the alarms. Most current approaches perform as a black box, a request is entered and the result is either “attack” or “not an attack” possibly in conjunction with the expected role or user. This does not aid the person tasked with resolving these issues very much. For signature based systems the feedback might contain what kind of attack was detected but since an anomaly based system only knows normal behaviour, it can be difficult to recognise false positives from

carefully crafted attacks.

Due to the theoretical vast detection potential, most literature adopted an anomaly based approach. The possibility of detecting unforseen attacks leads to the belief that it could have stopped some of the biggest digital attacks so far. However, as discussed before, there are still a number of issues that require attention in anomaly based detection systems. In addition to these issues, there are some limitations in the studies themselves as well. The most common limitations of existing solutions are the following:

- The solutions were tested on synthetic databases rather than databases originating from a real system. Data from the real world is usually hard to come by, but it would increase the reliability of the results.
- The feature space is usually very limited. The main reason for this is that a lot of work uses Naive Bayes as their classifier which assumes independence of features. To avoid overlap and dependence between features, information usually is abstracted of or features are merged with other features in to one. Information is lost in such an abstraction or merging of features which could otherwise have aided in detecting attacks.
- Most solutions only make use of features related to the syntax of the request (the query itself) or of features related to the data retrieved, not a combination of those.
- The false positive rate is too high for the solution to be practical in a big system or the solution puts impractical demands on the system in any other way.
- It is difficult to compare different works because there is no common data set or benchmark to do so.

1.2 Goals and Research Questions

This project focuses on anomaly based database intrusion detection systems and as such, overcoming some of the most common problems with those systems is the goal of this project. To achieve these goals, a number of research questions have been formulated.

Goal 1: Comparing existing solutions in the field of database anomaly detection systems on the same data set in order to provide a performance benchmark in terms of detection rate and false positive rate.

Subgoal 1.1: Specifying realistic scenarios related to a database in the health care domain to aid in testing the performance of various database anomaly detection systems. The scenarios should represent both normal behaviour and misuse situations.
Subgoal 1.2: Creating a environment for simulating different scenarios and testing the results against different database anomaly detection solutions.

- What are realistic misuse scenarios in the health care domain?
- Which of the existing solutions perform the best in the given misuse scenarios and settings?
- How do different solutions behave under the different types of scenarios in terms of detection rate and false positive rate?

Goal 2: Varying the feature space of data sets used in database anomaly detection systems to determine the impact on system performance in terms of detection rate and false positive rate.

- Can a richer feature space increase a database anomaly detection system's detection rate while decreasing the false positive rate?

1.3 Outline

The remainder of chapters describe the steps taken to answer the research questions. Chapter 2 contains a summary of the related work in the field of database anomaly detection systems and their advantages and disadvantages.
Our simulation and testing environment consists of three parts, a query and result set logger, a parser for the log files and a testing framework for comparison and analysis. These three are explained in detail in chapter 3.
Chapter 4 contains detailed information about the experiments conducted for the project. This chapter has been split into four sections: 4.1 about the experimental setup that was used, 4.2 about the feature selection that was used for our experiments, 4.3 on the normal behaviour and misuse scenarios simulated to retrieve our data sets and finally section 4.4 containing the results of our experiments and the discussion of those results.
The final conclusions, the answers to our research questions and the future work can be found in chapter 5.

Chapter 2

Related Work

The goal for a detection system is to distinguish normal behaviour from abnormal and mark the abnormal behaviour as an intrusion. In order to do this, normal behaviour needs to be modelled for each user of logical role in the system. When trying to model normal behaviour, different sources of information can be used.
Existing research in anomaly based database intrusion detection generally takes either one of the following two approaches: looking at the query involved with the request (*query centric approach*), or looking at the data retrieved by the request (*data centric approach*). In addition, some approaches use information related to the context, such as the time of day the query was executed or the user that issued the request. In this chapter, we will first look at the different feature types more closely and then discuss the state of the art with respect to different types of approaches. Finally, we will look at different attack types and compare a few of the related works in terms of feature selection and expected ability to detect different types of attacks.

2.1 Feature types

Features can be divided into three categories: *syntax based* features, *context based* features and *result set based* features. Syntax based features are features that tell us something about the query text such as the length of the query or the tables from which information was retrieved. Context based features contain meta data about the query, for example the ip of the sender of the query or the time the query was issued. Result set based features contain information concerning the result set retrieved by the query, examples include the amount of rows retrieved and what address details are retrieved by the query (if any). This last example is a very specific one, but it might be very useful in detecting an attack trying to gather all contact information of a company's clients.
The availability of context based features depends on the specific setup. For this reason, the amount and type of context based features varies greatly between different approaches. In some cases the result set of a query is not available to be used for the extraction of features due to the high cost of monitoring the result set of every query in the log files, for example in very large database systems that need to process a lot of transactions per second. Additionally, many types of queries do not have a result set. Examples include queries executing the *delete*, *insert* and *update* command.

2.2 Query centric approaches

An example of a query centric approach is the one proposed by Bertino et al. [6] which deals with database anomaly detection in RBAC-administered database systems, where each user is associated with a role. To represent data, the usage of one of three types of so-called *quiplets* is proposed, which differ in the detail of the information used. The five fields used in the quiplets are summarized as "*SQL Command, Projection Relation Information, Projection Attribute Information, Selection Relation Information and Selection Attribute Information*". Unlike the other fields, the command field is the same for all quiplets. For the coarse quiplet (or *c-quiplet*), the other fields are numeric and represent the count of the relations and attributes in the projection and selection clause of the query. For the medium-grain quiplet (or *m-quiplet*) the two relation fields contain a binary vector which denotes which tables are included in the query and which are not. The attribute fields are a vector which denotes the number of attributes that are used per relation. The four fields in the fine quiplet (or *f-quiplet*) contain the same binary vectors used in m-quiplets for the relations while they contain a vector of binary vectors for the attribute fields to denote exactly which attribute is being queried from each relation for the attribute information. The different quiplets represent a trade-off between speed and detection capabilities.
Normally the profiles are learned over roles. In the case that no role information is available, a clustering approach is proposed to create artificial roles which will be used instead. The profiles used for detection are built using the Naive Bayes Classifier [7]. In the detection phase, the model built in the training phase is used to predict a role given an incoming request. If the prediction does not match the real role, an alert is raised.
The described approach was tested using three data sets: two synthetic data set containing 100 tables; one with four roles and one with nine, and one real data set consisting of 8,368 SQL traces from eight roles on a database consisting of 130 tables. Randomly generated anomalous queries were added to all data sets to test the detection rate of the approach. One of the most interesting observations from the results in this study was that the m-quiplets outperformed the f-quiplets in some occasions. The biggest limitation is the relative high false positive rate observed for the tests with the real database. The authors state that the reason could lie in the nature of the specific data set. Other limitations include the fact that independence amongst features is assumed by the Naive Bayes classifier while this is not necessarily true. In addition, the frequency with which a role and query pair has been seen in the training phase influences the model which can causes a bias towards predicting certain roles.

A way to reduce the false positive rate was proposed by Wu et al. [5]. The main idea is the usage of a role-hierarchy in anomaly detection. Whenever an anomaly is detected where a senior function is profiled as a junior function (whose capabilities are a strict subset to those of the senior) it is not considered an anomaly. The feature space used in this work contains a combination of syntax based and context based features. This feature space can also be used when such a role hierarchy is not available by simply performing detection based on the role profiles or by performing detection based on profiles build on the basis of user-ids. Similar to [6], this work also uses Naive Bayes as their classification algorithm. The feature space proposed in this work can also be used without a role hierarchy and even with profiles based on users rather than roles. The approach was tested on AdventureWorks, a sample database of a fictitious company[1]. For the experiments, the work used the marketing and sales departments of the company. This led to a system with 12 roles of which two were abstract and had no user assigned to them. The final database was reduced to only include the 17 tables referenced by the users from the selected

[1]Microsoft MSDN. AdventureWorks sample OLTP database, `http://msdn.microsoft.com/en-us/library/ms124659.aspx` (Last accessed 13 January 2014)

departments. Legitimate queries were created by taking random combinations of users and applications those users are permitted to invoke. These applications then generated database activity on set tables on behalf of the user that invoked it. Anomalous queries were generated by taking legitimate transactions and changing the id and role of the sender to a role and id of some user in the other department.
Their tests show that the tolerance introduced by the role hierarchy does not lead to a significant decrease in detection rate compared to tests where this hierarchy was not used. The false positive rate for the tests where the role hierarchy was used was significantly lower.

Fonseca et al. [8] introduced an approach with two levels of abstraction. A command level abstraction which looks at single queries and a transaction level abstraction which considers execution paths and in general collections of queries. A query has to pass checks in both levels of abstraction in order to not be marked as an intrusion. The queries are reduced to their invariable part and translated into signatures for groups of queries to allow for a compact representation. The command level abstraction prevents attackers from changing the query in a significant way while the transaction level abstraction checks if the order of execution for the queries fits a normal execution path.
The performance of the approach was tested on a benchmark database. Besides random attacks, the authors also attempted to fool their own system by crafting attacks that are hard to detect and could be crafted by a well informed attacker. The results show an outstanding performance of 100% detection rate. The only false positive rates were found on attacks that specifically evade the command level abstraction by changing the text inside the strings and values in the where clause. The two real databases were used to determine the training time needed in a real system. Their results show that new profiles are learned even after hundreds of thousands commands over a nine week period. The main limitation of this approach is highlighted by the authors as well. Their system has a few specific weaknesses that especially insiders can exploit due to their knowledge of the system. One of the attacks that falls in to the category that is not detected by the system is the collection of large amounts of data, also called *data harvesting.*

Valeur et al. [9] propose an approach in which a number of statistical methods are used to calculate an anomaly score of a query compared to a learned profile. The features used in this calculation are the query skeleton (the query with all constants removed) in conjunction to the constants in the query in a separate vector. The profiles are built in the first part of the training period. Then more normal activity is used to establish anomaly thresholds of normal activity for those profiles. The anomaly score of queries in the detection phase is then compared to the maximum threshold found in the training phase in order to determine whether a query is considered anomalous or not. The approach was tested by performing known vulnerabilities on a web-portal and logging the queries performed. The false positive observed in the tests are very low (especially after some fine-tuning) but the detection rates are very low as well (up to 4%). In addition, a limited variety of attacks were tested.

Chung et al. [10] propose usage of a distance measure between database schema's and several thresholds to determine the so-called working scope of a user. If the distance from the working scope to the pending request is too large it is considered an anomaly. The effectiveness of the approach is shown via the use of two examples and has not been tested on any large data set. This lack of testing also represents the biggest limitation of the approach.

Low et al. [11] propose a database containing fingerprints of queries per user to check for anomalies. The fingerprints can be seen as an abstract version of actual (sets of) queries and are compared to the fingerprints of incoming queries. This work differs from others in that it takes the manner in which the query is formulated into account rather than what tables or attributes are accessed. Lee et al. [12] then further explored solutions for coping with issues that occur with the management of potential enormous set of fingerprints and other issues with the approach. Tests on both these systems are limited to the impact on the system performance rather than detection capabilities.

2.3 Data centric approaches

To build normal behaviour profiles, instead of looking at *how* a query is formulated, it is also possible to look at *what* data it retrieves. The profiles used for detection in data centric approaches are built on the information found in the result sets of queries rather than the queries themselves. For example, we can learn what diseases a doctor usually treats patients for instead of the fact that he or she looks at the disease table.
In [13], Mathew et al. applied this approach; the result set of the query is processed and converted into an S-vector. This is a vector consisting of a number of statistical measurements of the entries of the result set. The format of the S-Vector differs according to the database schema, while the values differ according to each different query. By keeping track of statistical summaries instead of individual records and by using a record with a fixed dimension, this S-vector prevents the normal behaviour model to become prohibitively large. User profiles are made by making clusters of S-vectors. During the detection phase, if a user sends a query of which the S-vector does not belong to his or her profile cluster, it is considered an attack. A limitation of the S-vectors is that they contain statistical summaries of the values, this means S-vectors cannot help in detecting some attacks based on very specific entries such as a specific disease. One other limitation is that a varying ordering of attributes with the same data can change the contents of a S-vector which may lead to false positives in the detection phase.

In the approach proposed by Gafny et al. [14], context based features play a relatively big role. A decision tree is generated from the context of queries in the normal behaviour in conjunction with the result sets for those queries. The decisions in the tree are made based on the context information while the leaves contains sets of rules which are true for the result set of normal behaviour. By following the decision tree using the context of the incoming request in the detection phase, a set of rules is retrieved. The extent to which the result set of the request follows the rules is converted to a numerical value and compared to a set threshold. When the score is higher than the threshold, the request is considered malicious. This threshold score is used to make a tradeoff between the false positive rate and the detection rate. The approach can in theory be used with various methods for calculating the extent to which a result set follows the rules and for inducing detection models.
The method used for building the decision tree in the tests was based off the Jaccard similary coefficient [15]. The approach was tested on a synthetic database albeit with realistic attack scenarios. The test results show a very high detection rate (93%) with a low false positive rate (9%) and a high detection rate (88%) and a very low false positive rate (1%), with two different settings for the threshold. A disadvantage of the approach is that it assumes legitimate user behaviour is completely captured in a set of rules and not being learned by observing issued queries. This requires an extra conversion step compared to most other approaches.

Another data centric approach was proposed by Harel et al. in[16]. The study contains a proposal for a so-called M-score, a score that represents the misuseability weight of tabular data. Using this score, anomaly detection can be performed by comparing the M-score of usual data requests with the M-score of new requests. In addition, the severity of a data leakage can be estimated by using this M-score as it represents the sensitivity of the data as well. A disadvantage of this approach lies in the fact that the score does not tell us anything about what type of data is being accessed, only the overall misuseability of the data. For example a very sensitive result set with only a few records might score just as high as a large number of records with low sensitivity. In addition, a patient attacker might be able to circumvent detection by performing multiple malicious queries with a low score, obtaining a lot of sensitive information overall.

2.4 Hybrid approaches

A hybrid approach is one that uses a combination of syntax based and result set based features, possibly with some context based features as well. This makes it a combination between the query centric approach and the data centric approach described in the previous sections.
The approach proposed by Costante et al. [17] uses a histogram based detection method. For each feature, every possible data value is mapped to a bin. For integers or dates, such a bin can be an interval; for nominal values, every value can have its own bin; for more complex types such as sets, multiple bins are used, mapping each item in the set to a bin. For each entry in the training set, the amount of items mapped to each bin is tallied in order to make a profile for a user or a role. These values are then normalised to create a distribution over bins for all features and profiles. Next, there is a tuning phase where the security officer can set a threshold which will be used in the detection phase, as well as set some exceptions to the learned profiles: some features might occur rarely but they should not be considered anomalous and to allow this, the security officer can mark such situations to prevent false positives in the detection phase. The detection is fairly straightforward: if any of the features in the incoming request occurs less frequently for the corresponding profile than the threshold (and is not one of the exceptions), the request is considered an anomaly. An anomaly score is also calculated in an attempt to quantify the severity of the anomaly. Finally, the work describes a broader framework including a feedback system in which a security officer can mark certain anomaly alerts as false positives and prevent them from occurring again in the future.
The histogram based detection method was tested on two data sets and with two feature spaces, one data set was obtained by monitoring the activities of a operational Oracle database of an IT company and the other data set was a synthetical data set obtained by simulating realistic misuse scenarios. One of the feature spaces used only syntax and context based features while the other one included a small number of result set based features as well, creating a hybrid approach. The results show that the approach has a low false positive rate of less than 1.65% across both data sets with relative high detection rates (over 70%) for the hybrid approach. The addition of the result set based features resulted in a higher detection rate but also a higher false positive rate. Three main limitations are noted by the authors. First, the features are all considered independently from each other in the detection phase while combinations of features might contain valuable information as well. Second, the queries are analysed individually, leaving the system vulnerable to carefully crafted combinations of queries by an insider who is aware of the system in place. Finally the feedback mechanism gives the security officer a lot of power, if he or she makes a mistake or has a malicious intent the system might not be as effective as hoped.

The solution of Santos et al. [18] looks at numerical meta-data regarding the SQL commands that are being sent to the database in a user session as well as numerical information about each of those commands in order to detect intrusions. Intrusions are detected by performing

statistical tests on the data, if all features match the pre-built profile for all tests, the request is considered normal. In addition, a risk exposure matrix to increase the alert management efficiency is proposed.
The approach was tested using a wide variety of attacks which were randomly generated and merged with an existing synthetic database which contains queries representing typical reporting behaviour. A drawback of this approach is that detailed information about queries is lost in the abstraction into numerical features.

2.5 Attack types

Similar to the distinction that can be made between the anomaly detection approaches, we can make a distinction between different kinds of attacks. One attack might involve a person accessing types of data he normally never accesses (e.g. financial information instead of medical information) while another attack might inspect similar types of data, but with different values (e.g. medical information on children instead of elderly). The first type of attack can be detected by looking at syntax based features while to detect the second type of attack, we need to look at the values returned, i.e. at result set based features.
This distinction was also made by Mathew et al. [13], they defined three types of attack in relation to a normal query: (1) different result schema/different result tuples, (2) similar result schema/different result tuples and (3) similar result schema/similar result tuples. Attacks of type (1) should normally be detected by both query centric and data centric solutions. Attacks of type (2) should generally be detected by data centric approaches, while the performance of the query centric approaches will depend on the way the query was formulated as well as the way queries are interpreted. Type (3) attacks are typically hard to detect for approaches which consider the statistical distribution such as the one described in [13], while an approach which looks at actual data values such as [17] should be able to detect it. Similarly to attacks of type (2), attacks of type (3) may be detectable by query centric approach depending on the specific query and the adopted approach.

2.6 Feature spaces in the state of art

The related work presented so far shows us that there is no golden standard for which features to use when detecting anomalies in database systems. There are even significant differences amongst solutions who fall in the same category of approaches. In addition, the way in which information is represented differs as well. We summarise the reasons different feature spaces are used in different approaches with the following three observations:

- If the goal is to make a system which detects anomalies in requests before those requests are executed, we are faced with some strict timing demands: the detection system should not slow down the application by a noticeable amount. This means we have very little time to perform our detection. Therefore it has to be as fast as possible. Adding a large amount of features could slow down the system. In a response to this limitation, including some sort of threshold or introducing different grains of detection in order to allow a tradeoff between performance and speed [14][17][6] is a common choice.

- Since time is an important asset when detecting anomalies, a majority of the solutions favor the usage of the Naive Bayes classifier for its low computational cost [6]. The downside of using Naive Bayes, however, is that it assumes strong independence between the features. Since this assumption is rather strict and the algorithm seems to perform reasonably well in practice, most solutions use it regardless. Adding features would possibly increase the dependence between the attributes and as such might not contribute as much to the detection rate as they would with other algorithms (e.g. decision trees).

- Many different types of queries exist. Some of them modify the contents of a table while others simply retrieve data. When we select features that might aid in the detection of anomalies, we have to take into account that the information might simply not be present in all types of queries. Usually this is circumvented by not using features that occur infrequently.

In this thesis we compare several variations of three approaches available in the state of the art presented so far: the quiplets from the work of Bertino et al. [6], the approaches described in Wu et al. [5]'s work and the histogram based method with the added result set based features described in Costante et al.'s [17] work. These three works were selected because these three works represent different approaches in terms of the types of features used. The quiplets consists of only syntax based features, the feature space used for Wu et al.'s work contains a combination of context based features and syntax based features and Costante et al.'s work contain a hybrid solution that uses all three types of features. Table 2.1 contains an overview of which features are used in which work. The features used contain a tick (✓) if the approach includes the specific feature.

Table 2.1: Overview of features used by the three works compared in this thesis.

type	**feature**	**description**	**Bertino et al. c-quiplet**	**Bertino et al. m-quiplet**	**Bertino et al. f-quiplet**	**Wu et al.**	**Costante et al. hybrid**
syntax based	`query_attr_list`	a list containing the attributes selected in the *SELECT* part of the query.			✓		✓
	`query_attr_number`	the number of attributes selected in the *SELECT* part of the query (the amount of items in `query_attr_list`).	✓			✓	✓
	`query_attr_counter`	a list containing the amount of attributes used per table for the *SELECT* part of the query.		✓			
	`query_length`	the length of the query (amount of characters).					✓
	`query_table_list`	a list containing the tables in the *FROM* part of the query.		✓	✓	✓	✓
	`query_table_number`	the number of tables selected in the *FROM* part of the query (the amount of items in `query_table_list`).	✓				✓
	`query_where_attr_list`	a list containing the attributes in the *WHERE* part of the query.			✓		✓
	`query_where_attr_list_number`	the number of attributes in the *WHERE* part of the query (the amount of items in `query_where_attr_list`).	✓				✓
	`query_where_attr_counter`	a list containing the amount of attributes used per table for the *WHERE* part of the query.		✓			✓
	`query_where_length`	the length of the *WHERE* part of the query (amount of characters).					✓
	`query_where_table_list`	a list containing the tables in the *WHERE* part of the query.		✓	✓		✓
	`query_where_table_list_number`	the number of tables in the *WHERE* part of the query (the amount of items in `query_where_table_list`).	✓				
	`query_command`	the SQL command used.	✓	✓	✓	✓	✓
context based	`query_timestamp`	the time the query was logged.				✓	✓
	`web_app_role`	the role of the user sending the query.	✓	✓	✓	✓	✓
	`web_app_uid`	the user-id of the user sending the query.	✓	✓	✓	✓	✓
	`department_id`	an identifier for the department of the company.				✓	
	`access_type`	an identifier for how the database was accessed.				✓	
	`area_constraint`	the location where the query was sent from.				✓	
result set based	`rows_bytes`	the number of bytes of the result set.					✓
	`rows_number`	the number of rows of the result set.					✓
	`result_set`	aggregate information about the most interesting columns of the result set.					✓

Chapter 3

Methods

In order to fairly compare work from the literature we need a collection of data sets to use for testing the approaches we want to compare. Almost no overlap exists between the data sets used to benchmark the current solutions which makes it hard to compare the different available solutions. In order to create such data sets and perform the comparison, a testing framework was set up.
The recent steps towards digitalisation and the abundance of sensitive information in a health care system has drawn us towards using the health care domain as the application domain for this thesis. Health records of patients may contain very sensitive information and in any case contain personally identifiable information. This kind of information has to be handled with care and access to such information has to be assigned cautiously. Nevertheless, emergency situations sometimes call for usage of data without proper consent of the patient in question. In addition, the size of a hospital in terms of amount of employees is usually too high for manual supervision of all data accesses. Even if the amount of requests would be feasible for manual processing, reading queries and tables of data and comparing them to normal behaviour is not an easy task. It requires extensive knowledge of the database system and of all the normal activities of every employee. Evidently, a largely automated system is preferred. Our system tries to provide an environment for simulating scenarios in a health care information sytem and testing existing solutions in such scenarios. Figure 3.1 contains the overview of our testing framework, the logging class, log files, query and result set parser, feature information and form our contribution for the testing framework. Activity is simulated for all users via the Tryton[1] client (which serves as the interface to GNU health[2]). The database queries resulting from this activity are logged by the logger class which is located in the Tryton server. The logger class writes the activity to log files which are then parsed by a custom written parser. The parsing result are inserted in a local database and then used for experiments and analysis using Rapidminer[3].
All the steps mentioned above will be explained in detail in the remainder of this chapter.

[1]`http://www.tryton.org/`
[2]`http://health.gnu.org/`
[3]`http://rapidminer.com`

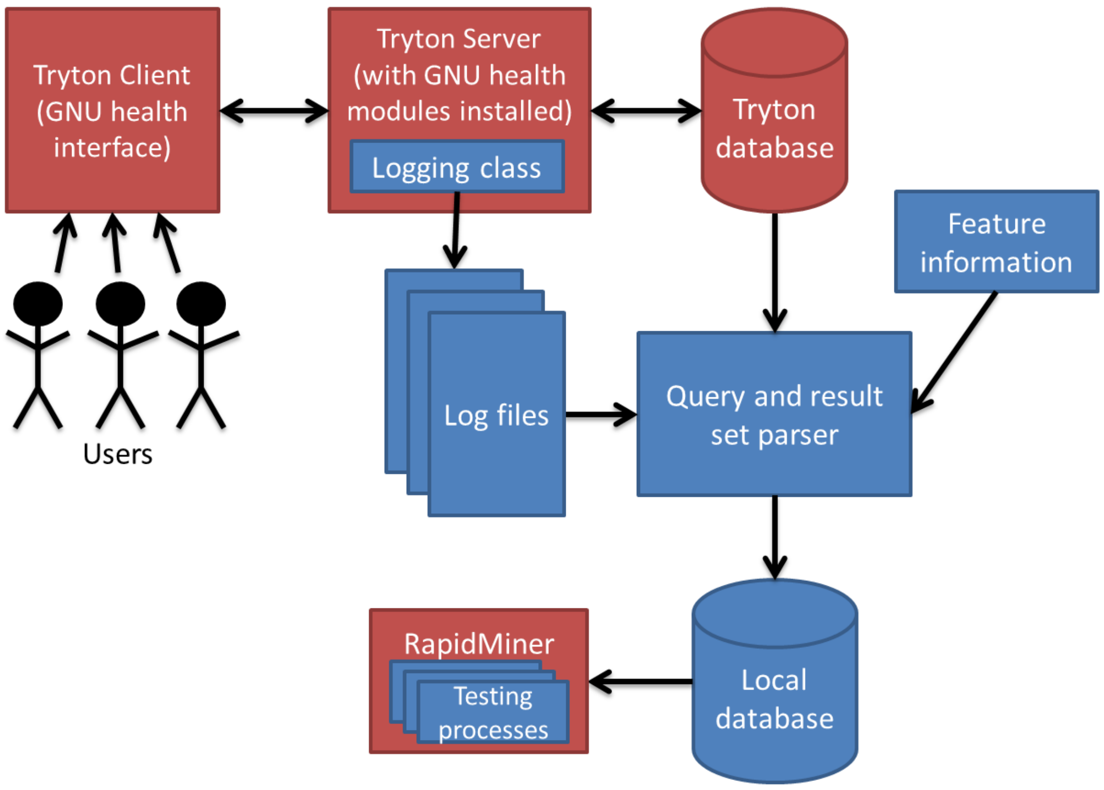

Tryton Client (GNU health interface)
Tryton Server (with GNU health modules installed)
Logging class
Tryton database
Feature information
Users
Log files
Query and result set parser
RapidMiner
Testing processes
Local database

Figure 3.1: The complete architecture of the testing framework

3.1 Logger

The free and open source health and hospital information system GNUhealth[4] running on the high-level general purpose application platform Tryton[5] was used as the information system for simulating activity. The modular and open source nature has lead for GNU health to be adopted by the United Nations University International Institute for Global Health[6], for training and implementation globally, particularly in developing countries. Additionally it is being used by, amongst others, the Jamaica ministry of health for their Public Healthcare Network, as well as hospitals in Argentina and Bangladesh, universities in Indonesia and the Commonwealth Caribbean.
Tryton is a application platform that can be used to many ends. In its core it provides functionalities for accounting, project management and more. Beyond that, it can easily be extended to make it suitable for other purposes with the use of modules. GNU health is basically a set of modules which allow Tryton to function as a health care information system.
As depicted in figure 3.1, Tryton uses a server-client architecture. This architecture allows multiple clients to connect while keeping updates and maintenance restricted to the server only. This also means that it is possible to use external modules in Tryton by installing them on the server. In our case, we use the health-modules provided by the GNU health project to turn Tryton in

[4]`http://health.gnu.org/`
[5]`http://www.tryton.org/`
[6]`http://iigh.unu.edu/`

to create a health information system. For our experiments we simulate normal behaviour and realistic misuse scenarios in the Tryton client.

As Tryton runs on Python and is open-source, we have the opportunity to write a class that extends the database request handling module. The logger will write all queries and the corresponding result sets to log files. The logger class makes no modifications and as such has no influence on the functionality of the remainder of the system.
As mentioned earlier, the task of the logger class is to extract the queries and result sets from Tryton and write them to a (log) file. By doing logging in this way, we can process the data request on a later moment. Some research has been done to designing a real-time data leakage detection solution, i.e. one that tries to detect an anomaly in a request before it is executed. In the case of a real-time data leakage system, the system must conclude whether a request is anomalous or not before sending the result set back to the issuer. Attempting to compare approaches from literature in a real-time setting was not done in this project for a number of reasons:

- Speed: preliminary tests showed that logging all database activity slows the system down noticeably. The addition of a complete parsing of both the query and the result set and applying the results to a detection model would worsen the impact.
- Training and Testing convenience: since anomaly detection models have to rely on a pre-learned model of normal behaviour, saving data requests to file at some point is preferred in any case. In addition, we want to test a data set multiple times with varying test setups or against different algorithms. Reenacting activity for every test would be very time consuming as well as difficult because we have to get details such as the system time exactly the same as before.
- Flexibility: by not implementing all the steps in the process in one location, we do not restrict ourselves to a specific context for implementing the parser and the detection model.

A benefit of the decision of implementing the logger class as a part of the database handling module of Tryton is that the logger can also be used for logging other applications running on Tryton and not only GNUhealth.

Since this project is limited to data leakage (i.e. data leaving the system) and does not account for other kind of data misuse (e.g. removing or modifying data), we only need to log queries that execute the `SELECT` command. Doing this will also guarantee that a result set can be retrieved, even though it may be empty. The remainder of the logger is straightforward: simply write all relevant information available to a comma separated values (csv) file. In our case this information is the database name, the full query, the result set after executing the query and finally the current time. The user and their role in the system are added to the logs as those are needed to create user and role profiles. We have to add this user and role information manually because it is not available in the query handling class of Tryton.

3.2 Parser

Before we can train the detection model using the logged information, we need to break the query and result set down into useful features. To be able to cover all types of anomaly detection approaches we have to be able to extract both syntax and result set based features. The syntax based features are usually completely determined by the approach that wants to use the features. For example, if we want to parse a log file for usage with *f-quiplets*, we need to process the query in such a way that gives us the features used in that quiplet. On the other hand, the result set based features used are sometimes a combination of what the approach dictates (e.g. amount of rows in the result set) and application specific features (e.g. disease names in medical applications, bank account number in financial applications). For our comparisons, we need to determine a number of features that can be useful for detecting attacks. Information required for the parser to create and populate these features is read from a file that was created in advance. In figure 3.1, this file is labeled "feature information". More details on the feature selection for our experiments can be found in section 4.2.

The creation of some syntax based features requires full knowledge of the layout of the database of the system where the activity was performed on. For this reason, the parser connects to the Tryton database to extract meta data about the layout from the `information_schema` table.

As said before, the task of extracting features from queries and result sets is done by a custom written parser in the Python language. The data is read one entry at a time from the CSV-file provided by the logger after which the parsing can begin. This parsing has two major parts: the parsing of the SQL query and the parsing of the result set.

The parsing of the SQL mainly consists of finding keywords (most importantly "`FROM`" and "`WHERE`") and using those to split up the query. Tricky parts include functions (such as `extract()` for timestamps), reverting aliases and correctly escaping and removing quotes and parentheses in to order to parse the entire query correctly. In the end this leaves us with all the features we need: lists of attributes and tables as well as some other statistics about the query such as the length of the query and the number of attributes. These features are then temporarily stored in a list in the format we need them to be for the detection. In our case this means storing them in several different variations as there is overlap in what information is used, but not in the format of storing that information.

Parsing the result set is also a difficult task as we have to know what type of information is in the tables. For example, we process a nominal value (such as a user-id) differently from a integer value (e.g. the number of days a person has been hospitalised) despite both being represented as integer in the result set. We use the information from the SQL parser (which column represents which attribute) in combination with details from the feature information file described previously. For every (result set based) feature we want to use in the detection phase, the file contains the type of data the attribute is in. Using the conversion table 3.1, we determine how to process each column. Since a result set from a query can retrieve multiple rows, some statistics or aggregates over all rows are used as features.

After the parsing is done, all features will be written to a local database. Since Tryton uses a PostgreSQL[7] database for its data management, the same database management system was chosen here as well to ensure minimal conversion issues between different types of databases in the entire data flow from logger to experiments. Since all features for all tested detection algorithms have to be converted by the parser, the resulting database schema has a relatively large number of attributes. The average processing time for a single entry (i.e. one query and result set) for our tests was approximately 35 ms, which is reasonable if we keep in mind that the process was not optimised for speed. The size of the query in some cases reaches over fifty thousand characters and the result sets contain up to a thousand rows of data for a single query. The speed of this step is not the most important part since we only need to convert the data once in order to perform many experiments.

[7] `http://www.postgresql.org/`

Table 3.1: Conversion table from database attribute to detection features

Attribute type	**Example**	**Feature**
Numerical	age of a patient	average value maximum value minimum value standard deviation
Nominal	patient ID	top most frequent value and frequency second most frequent value and frequency third most frequent value and frequency
Text	(additional) critical info	average length maximum length minimum length standard deviation of length most frequent value fraction of rows with special (non-alphanumerical) characters
Date	date of birth	maximum value minimum value top most frequent value

3.3 Rapidminer

RapidMiner, an open source predictive analysis and data mining tool, was used for testing various approaches. Rapidminer is a powerful data miner with a substantial number of built-in features, allowing for quick and reliable data mining. It is used by companies over the world, including Lufthansa, Pepsi, Siemens AG and Volkswagen.
Rapidminer was chosen because it has standard interfaces for various database systems and built-in processes for almost all analysis required for this project. Examples used for this project include k-fold cross validation, partitioning data using preset ratios and the generation of Naive Bayes classification models. The generation and modification of attributes and values for testing purposes have been used as well. The default *Performance* process was used when possible to retrieve detection results.

Before any testing, the data is loaded from the database into Rapidminer and preprocessed. All test data sets are filtered to exclude most system queries not directly related to the user activity. The resulting data table is stored locally for later use. An example of such a preprocessing process can be found in figure 3.2. The database containing normal behaviour is split into two parts, 70% training data and 30% testing data, this split has been done only once to not allow any randomness affect the results and comparisons.

For all approaches, a cross validation using the training data set was performed. Where possible, the built-in k-fold cross validation process was used for $k = 10$. The model built from the complete *training* set was stored for the detection tests. In the case that an approach is not fit for usage with the built-in cross validation process, we create a custom process to perform the 10-fold cross validation. The process for the cross-validation generally looks similar to the example given in figure 3.3.

Besides performing cross validation, the performance of the models was also tested using the

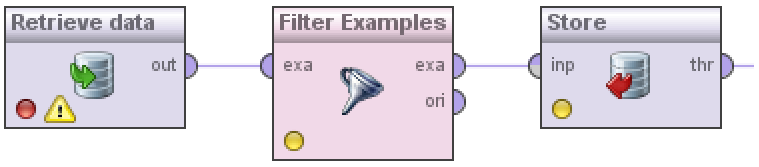


Figure 3.2: Example process used for data preprocessing.

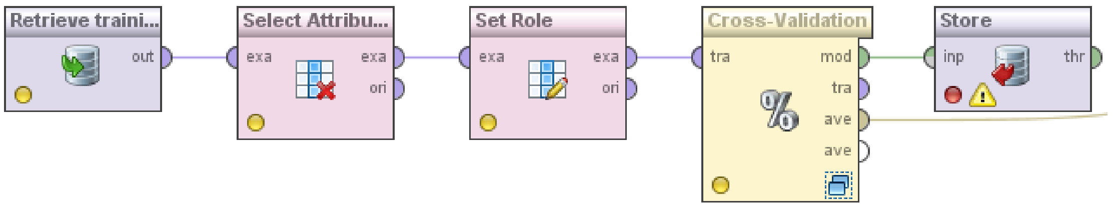


Figure 3.3: Example process used for cross-validation.

remaining 30% of the normal behaviour that was stored as the testing set. Since this data set contains only normal behaviour, every detected attack is a false positive and as such this will give us a false positive rating. An example for a process that performs such a test is given in figure 3.4.

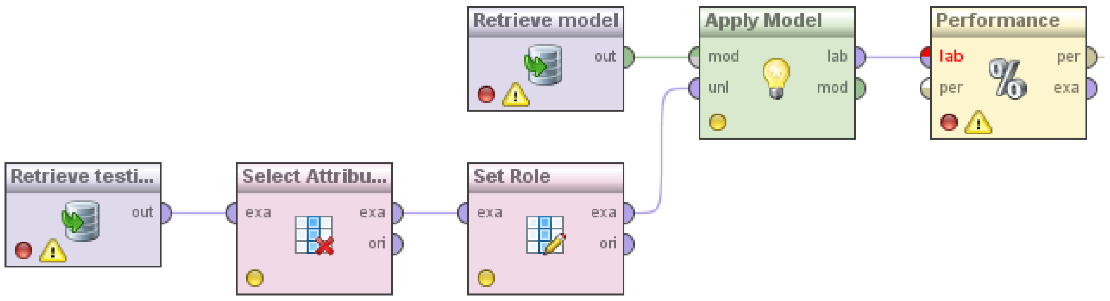


Figure 3.4: Example process used for testing the false positive rates.

The general process for the detection rate experiments are fairly straightforward for most experiments, simply apply the model built in the training phase to the testing set and input the results into a performance process. Obviously there are some small alterations to accommodate for the specifics of each approach, but the example process in figure 3.5 sums up the most important parts.

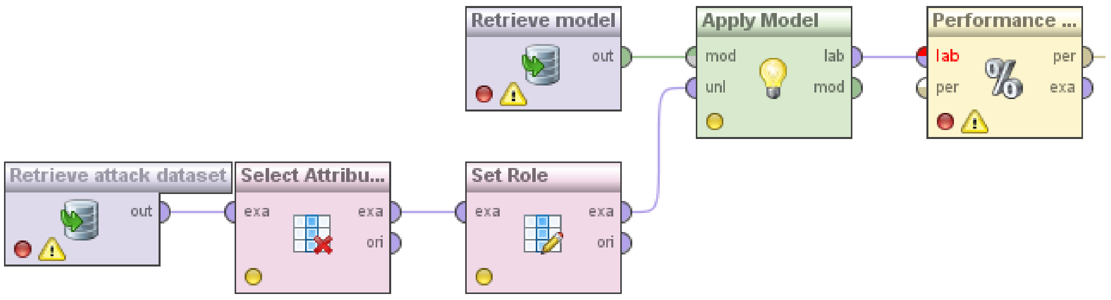


Figure 3.5: Example process used for testing the detection performance.

Chapter 4

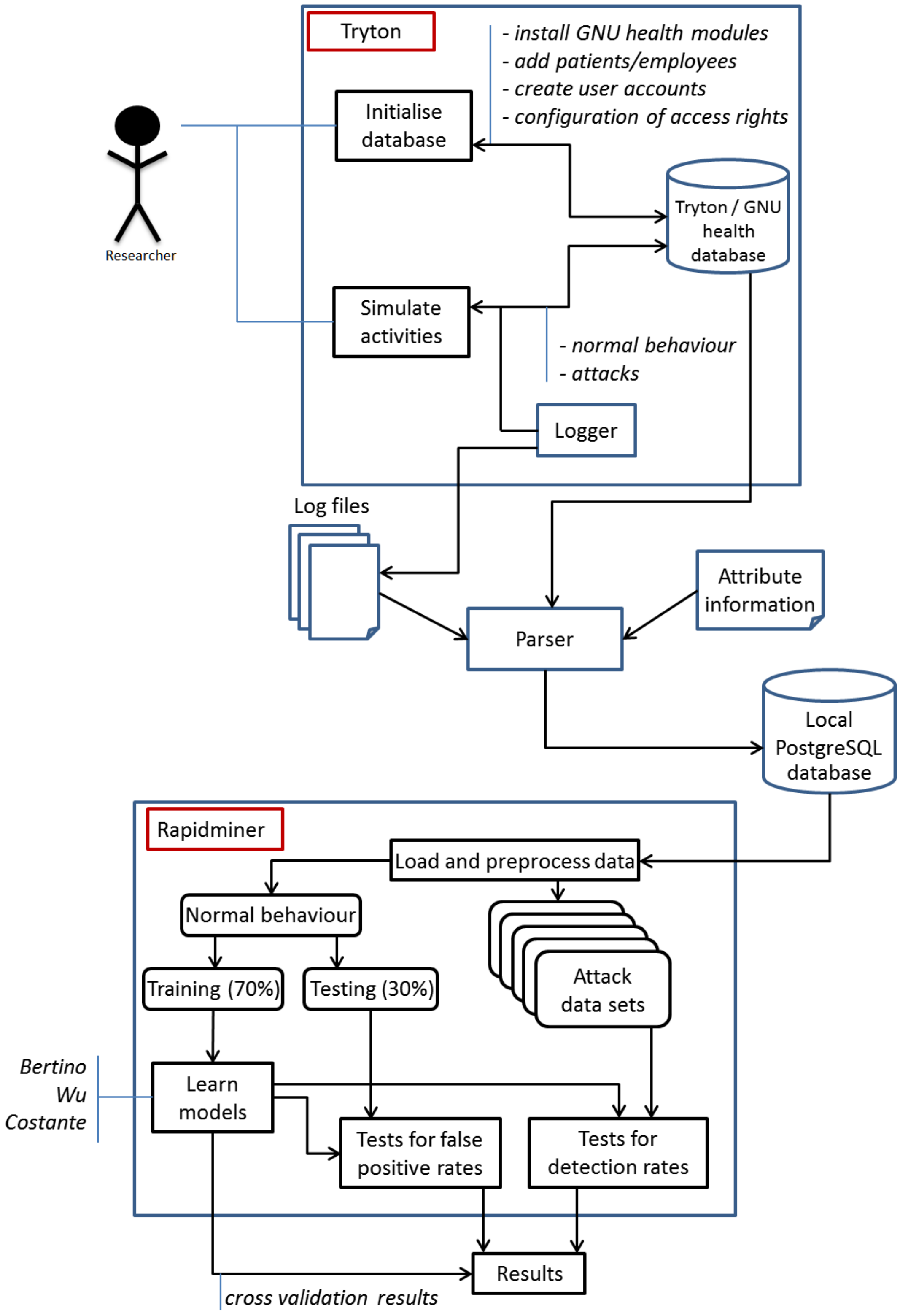
Experiments

To test and compare the performance of different existing database data leakage detection systems we carried out a series of experiments, as described in detail in the next sections. The approaches chosen for comparison are the *c-*, *m-* and *f-quiplets* described in [6], the three approaches by Wu et al. as described in [5] and finally the histogram based method described in [17]. The details on the feature selection used for these experiments can be found in section 4.2.
In order to compare the approaches in a meaningful manner, we want to have a common data set. To create such a data set, we formulated normal behaviour as well as five misuse scenarios which can be simulated using the testing environment described in the previous chapter. To ensure that the tests are realistic, medical experts verified the validity of the scenarios, behaviour and setup. Section 4.3 contains all information about the scenarios.
Finally, in section 4.4 we present and discuss the results of our experiments.

4.1 Experimental Setup

The environment used for the experiments was the testing environment described in the previous chapter. A complete overview of process used to perform the experiments can be found in figure 4.1.
A Tryton server was installed on a virtual machine running Ubuntu 12.04. The GNU health modules were installed in Tryton and the logger class was implemented next. The parser and feature information files were added to the machine to allow for parsing on the same machine. Finally a PostgreSQL database was created to allow for storage of the data sets.
Before the logger was deployed, the GNU health information system was filled with 500 randomly generated patients with a variety of diseases, appointments and other details. In addition, all employees were added to the system and subsequently assigned a user account. Next, we activated the logger class so that any activity performed from that point on would be logged. All activity was then simulated manually by logging in to the Tryton client on different user accounts and performing actions considered normal for them, generating log files. All the details on the user activity will be presented in section 4.3. The log files were completed by manually adding user id and role data afterwards. The parser was then used to read all the log files and create a set of tables (one for each attack and one for normal behaviour) in the local database containing the features used for testing.
The tables from the local database were then read by a process in Rapidminer which preprocessed and stored the data in Rapidminer. After the preprocessing, the models were learned for all approaches and subsequently stored for usage in the test processes. The test processes used

Figure 4.1: The process which was used for performing our experiments.



the preprocessed data sets as well as the learned models to generate performance results per data set for all approaches. As a final step, the accuracy measure retrieved in some processes were converted to a false positive rate or a detection rate, depending on the data set used. For the control set (normal behaviour), every incorrectly predicted role or id is considered a false alarm which adds to the false positive rate. For the attack data sets, every incorrectly predicted role or user id is considered a true alarm which adds to the detection rate.

In total there are five data sets containing simulations of the misuse scenarios and a control data set containing only normal behaviour, which will all be presented in section 4.3. All these data sets were tested using all variations of the methods described in three studies from literature: Bertino et al.'s quiplets [6], Wu et al.'s feature space using user profiling, simple role profiling and role hierarchy profiling [5] and Costante et al.'s histogram based method [17].
From the work of Bertino, all three types of quiplets were tested twice, once with role and once with user id as profiling feature.
Wu et al.'s feature space was used for tests with user id and role as profiled feature, as well as the role with role hierarchy introduced in their work. In our case the only hierarchy in the system is that the activities of the nurses are a subset of the activities of the doctors. This means that the doctor role is senior to the nurse role and that any doctor profiled as a nurse will not be considered an anomaly whereas they normally would. Since both Wu et al.'s and Bertino's approach use Naive Bayes as their detection model generation algorithm, these tests also show us how much the detection rate and false positive rate are affected by the choice of features and by the way gathered information is represented.
Costante et al.'s histogram method allows for a wide variety of features, the impact of which we test by using four set-ups: two for user id and two for role as profiled feature. For both profiled features we performed one test with only syntax and context based features and one test with a combination of syntax, context and result set based features. The selection of features is explained in the next section. The choices for bins were made such that any numerical values (include dates) were on an interval that guarantees at most 100 bins. In general, the decision depends on the precision we need. For example, the time of day might be needed at a precision that indicates whether an employee worked the morning, afternoon or night shift in some applications, which would be three bins. In other cases where working times vary a lot more a precision of one hour (meaning 24 bins) might not suffice.

4.2 Feature Selection

In an ideal situation we would use all the information contained in the query and result set in a way that can clearly help in detecting an attack. Doing so with the Naive Bayes classifier is probably not a good idea as the added features are likely to have some dependency with existing features. Thus the main assumption of Naive Bayes is not met anymore. For this reason and to accurately reproduce the results from the original work, we use the features that were used in the original works as long as they were available and meaningful in our setup. The SQL command field and IP address were always the same for all queries so they were omitted during the experiments. In addition, our system did not include department or regions like the database used by the original work of Wu et al. [5]. Since one of our goals is to include experiments with more features, we use the histogram based approach by Costante et al. [17] to perform these experiments. This approach does not try to estimate a user or role but compares features one by one, making it more suitable for a larger number of features. We used two different feature spaces for our experiments with the histogram based method. One consisting of purely syntax and context based features and one consisting of a combination of syntax, context and result set based features. The syntax and context based features used were in both cases the same set of features as used in the original work, the only difference is the selection of result set based features which we will discuss shortly. The complete list of features used for each approach has already been presented in table 2.1.

The result set based features used in the experiments include the number of rows and the number of bytes of data. These two features can help detecting data harvesting. The remaining result set based feature is named "result set" in table 2.1. This result set is in fact a list of features as every attribute that is profiled is converted to multiple features by the parser, depending on the type of the attribute. The conversion table used in this step was presented in table 3.1. For example:

Query:

```
SELECT sex, age, disease_name AS dis, primary_care_doctor AS doc
FROM patient
WHERE age = 37;
```

Result set: (partially shown)

sex	**age**	**dis**	**doc**
Male	37	Flu	4
Male	37	HIV	2
Female	37	Cholera	1
Female	37	Flu	4
...	...	...	...

Resulting feature table: (partially shown)

sex_top1	**sex_top2**	**sex_top3**	**age_avg**	**age_max**	**age_min**	**age_std**	**dis_top1**	...
Male;12	Female;11		37	37	37	0	Flu;5	...

Here we see that the `sex` attribute is of the nominal type and therefore will be converted to three features containing the top three most observed values with their frequencies (as `<value>;<frequency>`). The `age` attribute is a numerical value and as such will be summarized by four features representing the average, maximum, minimum and standard deviation of all the values seen in the result set. The full list of attributes and the reason they were selected is listed below:

- The *patient id* is included so we can know who's data somebody is accessing.
- The *patient's date of birth* is included to detect when the general age group of a request changes from normal behaviour.
- The *sex* of the patient is included. As it would be strange for a gynaecologist to inspect data of males while he or she normally only treats females.
- The patient's *critical info* field contains any especially important information that a doctor or nurse must know when treating the patient. Since this information can be very sensitive, it is included in our list.
- The *primary care doctor* of a patient is the doctor who normally treats a patient. If some other else is accessing the file, then this is probably an anomaly.

- The *disease* is a very obvious choice for a hospital, this value indicates what disease a patient is being treated for. This value can be considered sensitive and personal information which is why we included it.
- The *registration patient* field contains the id of patient that is currently admitted for treatment in the hospital. Since this probably changes the behaviour of the nurses and doctors towards the patient compared to a patient that is currently not being treated, it is included.
- The fields *diagnosis*, *patient* and *doctor* concern an evaluation of a patient. The doctor field is the doctor who diagnosed the patient in the evaluation, which should be the primary care doctor.
- A patients *family disease* history can give a doctor clues to what might cause some symptoms, but they can also be valuable information for possible future employees or health insurance companies. That is why it is included in our feature space. Both the patient and the name of the disease are included as separate features.
- We also include three features about lab tests, the *requestor* of the lab test, the *patient* who's blood is tested (or perhaps tissue or something else) and the *doctor* who performed the tests. Note that this last person is most likely an analyst and not the primary care doctor.
- The *patient who received a vaccination* field can again help us detect who accesses detailed information about a patient.
- Our final three features concern a surgery, the fields we use as features are the *patient's id*, the *patient's age* and the *operating physician.* Again these represent details that nobody should access without good reasons.

4.3 Scenarios

The limitation of using synthetic data sets (those made up for the sake of testing rather than those originating from a real system) can be overcome in two ways: by using a full sized real data set acquired from an organisation, or by simulating realistic misuse scenarios in an synthetic information system. Since the former option is difficult to organise for the scope of this project, the latter approach was chosen. Normal behaviour and five data misuse scenarios for four roles were formulated and then verified by medical experts from the Roessingh medical rehabilitation centre[1]. Besides usage for these experiments, these scenarios could form a basis for a general test set on which future solutions can be tested.

In our experiments we simulated a hospital with with six doctors, four nurses, one secretary and one administrator. The hospital has around 500 patients of which the majority are just in the system and not actively being treated. The employee and patient data were generated randomly by a small python script. The remainder of information such a diseases, appointments and employee login accounts were added manually before the logging of activities started.
Every patient in the system is assigned to a primary care doctor. For our misuse scenarios this has the following semantics: if doctor A is the primary care doctor of patient B, then doctor A is the doctor who treats patient B and no other doctor is treating patient B.
Nurses also treat patients, but no patients have a nurse directly assigned to them in the system,

[1]http://www.roessingh.nl/

i.e. patients can be treated by several nurses.
The secretary also deals with patients, but she does not have a set of patients assigned to her and cannot access medical information. In addition, she does some miscellaneous activity in the administrative part of the system.
The administrator governs the entire database, if anything is wrong: he should look in to it. Normally this involves adding new users to the database, changing settings in the system and updating stocks when new goods arrive.

In case of an emergency, we want doctors and nurses to be able to be as flexible as they have to be in that situation without being hindered by the system. For this reason, all doctors and nurses have access to the patient records of all patients, a common practice in real cases.
The secretary has access to all administrative parts of the database as well as access to the patient demographic data, appointments and patient evaluations.
The administrator has full access to the entire database. It is his responsibility to keep the system running so he must be able to access it.

The normal behaviour that was simulated for our experiments consists of all actions that are to be expected from each role. This means inspecting and sometimes modifying details of a patient for doctors and nurses. As mentioned, doctors only inspect and modify files of patients of which they are the primary care doctor. For the administrator and secretary this means performing various actions on the administrative part of the system. For the secretary this includes inspecting (not modifying) patient data and adding appointments.
During the simulations, the system time was changed frequently and the order of actions was randomised in order to avoid causing detection mistakes due to the biases in context due to the simulations.
Together with normal behaviour we also simulated misuse scenarios to test the ability of the approaches under analysis to detect them. When presenting a misuse scenario (or attack) we also discuss what features are necessary for an approach to be able to detect that attack. For example, if an attack is anomalous because a user accesses a system during night time, then the approach must include the time of day in some feature otherwise it will be unable to detect the attack.

Attack 1: administrator accesses medical data

In this scenario, the system administrator misuses his privileges to access patient data. The administrator usually checks stocks and does administrative work with respect to the user accounts of the system. He or she uses a lot of different tables of the database but none of those are directly related to patient data. In this attack the admin accesses patient data.

Since our system only has one administrator, we expect to see detection perform similarly for both *role* and *user-id* as profiled feature for the data set resulting from this scenario. As both the data values as the tables accessed by the administrator are new to the system, both query and data centric approaches are expected to have high detection rates for this scenario.

Attack 2: doctor specialised in treating children accesses data of elderly patient

In our second scenario we consider a doctor in the hospital. This doctor only treats patients aged eighteen or less because of his or her specialisation. Since all doctors can access all patient data in this setup, this doctor can as well. In this attack, the doctor looks at data of a patient who is much older than eighteen and who he or she does not treat.

Since all patients are treated by doctors, we do not expect high detection rates for tests with *role* as the profiled feature. Since the actual value of the age field is important, we expect only data centric approaches that include this field to detect the anomaly accurately.

Attack 3: doctor specialised in specific disease accesses data of patient with different disease

For our third scenario we consider a second doctor. This doctor is specialised in treating people with a certain disease and as such, only does this in the hospital. The attack consists of this doctor inspecting the file of someone who does not suffer from the disease that he or she is specialised in. Note that our setup does not account for the severity or sensitivity of disease in question during the detection phase. Specific diseases may affect the severity in a real case of data misuse greatly.

Inspecting the disease entry for a patient is considered normal activity for a doctor. The way to detect this kind of anomaly is by looking at data values to find what specific disease is being looked at. This means we expect only result set based approaches to detect this anomaly. Additionally, since other doctors in the hospital treat the disease in question, we need the *user-id* as the profiled feature in order to find this anomaly.

Attack 4: doctor accesses file of a patient she does not treat

Our third doctor can be described as a generalist, he or she treats all different kinds of patients for various complaints they may have. However, as any other doctor, he or she only treats the patients that are assigned to him or her. In this attack, the doctor accesses the file of a patient he or she does not treat.

Again we have a case where the values are more important than the tables used. Further more, we expect only the test for *user-id* to be raising an alarm since some other doctor would normally access this file (as every patient has a doctor assigned to them).

Attack 5: nurse logs in at an unusual time

In our fifth and final scenario we have a nurse. This nurse usually only works day shifts from 9am to 5pm. This is important since in this case, he or she logs in and inspects patient data files around 11pm.

This scenario is different from the others in the sense that the anomalous feature is context based rather than result set or syntax based. This means that only approaches that include the time of day as a feature should be finding anomalies in this scenario.

4.4 Results

The results for the control data set containing only normal behaviour can be found in table 4.1. The normal behaviour was first split into two portions, 70% went to the training set (21344 rows) and the remaining 30% were put in the testing set (9148 rows). For all approaches, ten-fold cross-validation was performed on the training set to estimate the performance of the models on an independent data set. Since we know that the training set consist entirely of acceptable behaviour, we can mark every detected anomaly as a false positive. The resulting false positive rates (FPR) can be found in the first row of the table. The second row of data contains the corresponding error margins (EM).

By performing tests using the testing set and the models built from the training set, we are able to test the performance of the model on the remaining normal behaviour. The results of these tests can show us if significant parts of the normal behaviour did not end up in the training set due to the random split made initially. In such a case these results would differ by a large margin from those in table 4.1. The results of these tests can be found in table 4.2.

The results of the data sets retrieved from simulating misuse scenarios can be found in table 4.3. The column labeled "data set" shows which misuse scenario was simulated for creating that data set. Since these attack data sets only contain attacks, every entry marked as an anomaly is considered a true positive. In the end, this gives us detection rates (or true positive rate) per attack and approach.

Since the false positive rates of the cross validation in table 4.1 are more reliable than those in table 4.2, they will be used for comparison and analysis. The most important to note about table 4.2 is that the differences with 4.1 are quite slim overall. This means that, overall, most activity was distributed fairly evenly between the two data sets.

The false positive rates found in our experiments are vastly superior for both variations of the histogram based method (0,64% - 3,05%, highlighted in bold) compared to the other solutions, particularly the variation without result set based features. The difference between the hybrid approach and the query centric approach is a factor 4 for the tests where the user's role was used as profiled feature, and a factor 2,7 for the tests where the user-id was used as the profiled feature. In both cases, the query centric approach outperforms the hybrid approach in terms of false positive rate.
The false positive rates for the approaches that use Naive Bayes as their classification algorithm are rather high (29,74% - 71,72%), with better results for Wu et al.'s approaches where the role profiles were used. A reason for the high false positive rates can be found in a bias developed in the training phase by using the Naive Bayes algorithm. This bias results from the uneven distribution of user ids and roles in the training set. This causes the model to predict the most occurring roles and ids much more often than others almost regardless of the features. Since certain roles have more activities in their normal behaviour then other roles, such a bias is hard to avoid. In general, Naive Bayes will always have this problem when there is an unbalanced distribution. A random generation of normal activity as performed in related work probably gives a more even distribution amongst roles which leads to a better performance overall.

The detection rates are also affected by the bias described in the previous paragraph. As long as the real sender of a query is not the role or id to which the bias is towards, chances are high it will be marked as an attack. For this reason, it is hard to appreciate the high detection rates seen for some of the attacks. Specifically attacks number 2 through 4, where the features to detect those attacks are missing are and the results found for these experiments are most likely not actual detections but rather a result of this bias.

Table 4.1: Average false positive rates (FPR) and corresponding error margins (EM) for ten-fold cross validation on the control set.

	Bertino						**Wu**			**Histogram Based**			
	c-quiplet		**m-quiplet**		**f-quiplet**				**role hierarchy**	**without result set**		**with result set**	
	id	**role**	**id**	**role**	**id**	**role**	**id**	**role**	**role**	**id**	**role**	**id**	**role**
FPR	71,72%	39,04%	69,32%	50,08%	69,28%	50,03%	56,04%	32,22%	29,74%	**1,12%**	**0,64%**	**3,05%**	**2,57%**
EM	0,67%	0,98%	0,89%	0,99%	0,91%	1,00%	0,65%	0,38%	0,46%	0,11%	0,11%	0,31%	0,38%

Table 4.2: False positive rates (FPR) for the tests using the testing set on the model build on the training set.

	Bertino						**Wu**			**Histogram Based**			
	c-quiplet		**m-quiplet**		**f-quiplet**				**role hierarchy**	**without result set**		**with result set**	
	id	**role**	**id**	**role**	**id**	**role**	**id**	**role**	**role**	**id**	**role**	**id**	**role**
FPR	72,18%	40,04%	69,61%	50,03%	69,58%	50,87%	56,13%	33,36%	30,50%	0,92%	0,37%	2,82%	2,31%

Table 4.3: Detection rates for the data sets resulting from simulating the misuse scenarios described in section 4.3.

	Bertino						**Wu**			**Histogram Based**			
	c-quiplet		**m-quiplet**		**f-quiplet**				**role hierarchy**	**without result set**		**with result set**	
data set	**id**	**role**	**id**	**role**	**id**	**role**	**id**	**role**	**role**	**id**	**role**	**id**	**role**
attack 1	**100,00%**	**100,00%**	**100,00%**	**100,00%**	**100,00%**	**100,00%**	**99,29%**	**99,65%**	**99,65%**	**97,88%**	**97,88%**	**97,88%**	**97,88%**
attack 2	97,94%	19,54%	98,34%	84,09%	98,34%	84,09%	100,00%	93,00%	93,00%	0,63%	0,00%	**100,00%**	0,99%
attack 3	100,00%	0,00%	100,00%	0,00%	100,00%	0,00%	0,00%	0,00%	0,00%	3,70%	3,70%	**85,19%**	85,19%
attack 4	72,79%	25,59%	91,15%	80,06%	91,15%	80,06%	100,00%	0,00%	0,00%	6,67%	0,00%	**73,33%**	46,67%
attack 5	87,30%	19,94%	92,04%	85,62%	91,99%	85,67%	**100,00%**	**100,00%**	**100,00%**	**100,00%**	**100,00%**	**100,00%**	**100,00%**

For the attacks where the approaches were capable of detecting it we see relative high detection rates across the board (73,33% - 100,00%), these are highlighted in bold in table 4.3. The detection power in the histogram based method with a hybrid feature space is clearly observed in our results.
In general we see that the detection rates for the tests using the user id as profiled feature have higher or similar detection rates compared to the tests where the role was used for creating profiles. On the other hand, the false positive rates for the roles tests are lower (and thus better) for the role-profiles. The reason behind this is that the id-profiles are more specific and can therefore be distinguished better. This means that it is easier to distinguish between small differences in behaviour, causing quicker detection of attacks but also more errors in the case of small harmless deviations from normal behaviour. Overall, we can say that there is a trade off to be made when using one of these two approaches.

Attack data set 1 shows the best detection rates overall (97,88% - 100,00%). We expected both query centric and data centric approaches to detect the attack and they did.
The second data set shows us the benefit of adding the result set based features in the histogram based approach. We did not expect to see high results for tests using roles to built profiles and this turned out to be the case. The only approach theoretically capable of detecting the attack delivered with a 100% detection rate.
The remarkable part about the results of the third data set is that the detection rate for the histogram based approach with result set based features where the role was used to create profiles (85,19%). The fact that this rate is the same as the tests for the id-profiles means that the disease did not occur for any of the doctors in the normal behaviour used for creating the model. Since the data set containing normal behaviour has been split for training in order to test the false positive rating, it is probable that all occurrences of this specific disease ended up in the testing set. Training the model on a data set with more redundancy in the normal behaviour will increase the odds of preventing mistakes like these.
Data set number 4 appears to be the hardest to detect (73,33%). This is probably because the difference between the attack and the normal behaviour is too small to reliably detect it. An approach that considers multiple features in conjunction might have a higher detection rate for this scenario, as long as it uses result set based features as well.
Our final data set is different from the other as it requires a specific context based feature for detection: the time of the day. From the results we clearly see excellent results (100,00%) from the approaches that included this feature. This example shows us that, even with a bias, either approach can be very effective in detecting attacks as long as the correct features have been selected.

Another statistic that can be considered is the speed of the algorithms tested. There may be a difference in the level of optimisation in the different processes in Rapidminer but these statistics will give a reasonable basis for comparison regardless. All tests were measured with a precision of seconds and performed on a data set containing 2228 entries. The processes using Naive Bayes take less than one second, with no noticeable differences between the different feature spaces. The histogram based approach with result set based features included takes three seconds while it takes two to three seconds for the version without any result set based features. These results show that the Naive Bayes algorithm is definitely quicker with the capability of processing over 2228 entries per second, but that the histogram based approach performs reasonably well as well with over 700 queries per second.

Chapter 5

Conclusions and Future Work

In this chapter we will first draw some conclusions with respect to our results from the previous chapters as well as formulate answers to the research questions posed in the introduction. Finally we will look at what future work is left and in what direction open issues lie.

5.1 Conclusions

In this thesis we described a benchmark framework used to analyse and compare different anomaly based database data leakage solutions. The framework has been developed and used to measure false positive rates and detection rates of selected existing data leakage solutions. The results of our experiments have been presented in chapter 4. These results show that Bertino et al.'s solution, with false positive rates between 39,04% and 71,72% seem inadequate at distinguishing normal behaviour from attacks. The detection tests for which the user-id was used as profiled feature perform equal or better than those for which the role was used. However, the detection rate results are likely to be insignificant due to a bias developed during the learning phase. As expected (and as observed in the original work), the coarse grained quiplets perform significantly worse than the other two types of quiplets. The small feature space size in combination with the choice of using Naive Bayes as their classification algorithm leads to a very quick solution, capable of handling thousands of queries per second.

The added benefit of the role hierarchy introduced by Wu et al. is clear from the false positive rate although it is still rather high (29,74% - 56,04%). In addition, the fact that it sometimes marks queries as normal while they otherwise would be considered anomalous did not cause any decrease in detection rate in our attack tests. Overall, Wu et al.'s choice of feature space performs well in the misuse scenarios where they were expected to.

The histogram based method performed the best in our experiments, with a relatively low false positive rates for both the exclusively query centric setup as the combined feature space (0,64% - 3,05%). The detection rates are fairly high, although there are significant gaps in the detection for two of our attack data sets. This shows that the choice of features is very important for achieving high detection rates. The detection rates found in our experiments reflect the expectations of theoretical detection capabilities per misuse scenario.

The experiments also show that the selection of features is very important in achieving high detection rates. In addition we can conclude that using more features may cause the false positive rate to increase, but it may also increases the detection possibilities.

Finally, we will answer our research questions posed in the introduction:

- **What are realistic misuse scenarios in the health care domain?**
  The misuse scenarios specified in section 4.3 are verified by experts at the Roessingh medical rehabilitation centre. This means these could serve as a basis for testing future work against similar data sets. However, this does not mean our misuse scenario set is complete. As discussed, the specification of normal behaviour is equally important to build highly reliable data sets.

- **Which of the existing solutions perform the best in the given misuse scenarios and settings?**
  In general, the histogram based method performs best in our tests. The false positive rates are very low in comparison to the other solutions tested and the detection rate is similar for most tests. The difference between using *id* or *role* as profiled feature are similar for all tests. The false positives are lower for *role*, but so are the detection rates. For the histogram based method, we see a clear improvement in detection rates (and the difference between being able to detect certain types of attacks or not) when including result set based features but this also comes at the cost of a higher false positive rate.

- **How do different solutions behave under the different types of misuse scenarios in terms of detection rate and false positive rate?**
  In general, solutions have proven to be very effective in detecting attacks for which they have specific features in their feature space (e.g. time of day). One thing we can say is that for the histogram based method, the detection rates for *role*- and *id*-tests and with or without using features from the result set match the expectations per scenario in terms of which setups would be capable in detecting which attack.

- **Can a richer feature space increase a database anomaly detection system's detection rate while decreasing the false positive rate?**
  Yes it can, if the right features are selected and sufficient training has been done. Some attacks cannot be detected reliably without the inclusion of certain features. However, many possible features are not present in all requests and this sparsity can cause false positives in normal behaviour. Careful selection of features and sufficient training data can possibly nullify the disadvantages while still improving the detection.

5.2 Future Work

The solutions presented in this thesis have some limitations and there are still some issues that can be addressed.
The misuse scenarios used for simulation described in this work may function as a basis for comparison in future work, but the set is not complete. A complete set could serve as a basis for future comparisons and benchmarks of database data leakage detection solutions or, when generalised, even data misuse detection solutions in general. Even completely different approaches of misuse detection could be tested and compared in a meaningful manner.

The hospital system used in this work is fairly small. Larger systems will lead to better comparisons as the systems are more close to real life system. In addition, it allows for testing how the different detection systems behave under larger and more complex role hierarchies.

Two major kinds of attack were reviewed in this work: query centric attacks and data centric attacks. For the tests using Costante et al.'s histogram based approach we see a clear distinction amongst setups between where it is and where it is not able to detect the attack. Research could go towards if more types of attacks exist or if a more fine grained distinction between types of attacks can be made. Such additions lead to better testing and enable better comparison of solutions.

Our tests show that a richer feature space can indeed increase the detection rate albeit at the cost of a higher false positive rate. More training or a different feature selection might be able to improve the results even further. It also may be possible to reduce the features that contribute the most down to characteristics which can be applied in general to distinguish potent features on other information systems.

Most approaches use the Naive Bayes classifier which might not be ideal for large feature spaces. Perhaps some other algorithm fares better with larger feature space. One option considered in literature to reasonable success is using decision trees (with several decision tree building algorithms). An advantage would be the capability of dealing with attributes who together have a strong correlation with attacks, a disadvantage is that the size of a tree grows rapidly as the feature space grows bigger. Testing and comparing the detection performance of decision trees and other machine-learning algorithms would be another interesting further direction.

Bibliography

[1] P. Gordon, "Data Leakage - Threats and Mitigation," tech. rep., SANS Institute, 2007.

[2] "2013 Cost of Data Breach Study: Global Analysis," tech. rep., May 2013.

[3] "Data Leakage Worldwide: Common Mistakes Employees Make," tech. rep., September 2008.

[4] S. Axelsson, "The base-rate fallacy and the difficulty of intrusion detection," *ACM Transactions on Information and System Security (TISSEC)*, vol. 3, no. 3, pp. 186–205, 2000.

[5] G. Z. Wu, S. L. Osborn, and X. Jin, "Database Intrusion Detection Using Role Profiling with Role Hierarchy," vol. 5576 of *Lecture Notes in Computer Science*, pp. 33–48, Springer-Verlag Berlin Heidelberg, 2009.

[6] E. Bertino, E. Terzi, A. Kamra, and A. Vakali, "Intrusion detection in RBAC-administered databases," in *Computer Security Applications Conference, 21st Annual*, pp. 10 pp. –182, December 2005.

[7] T. Mitchell, "Bayesian learning," *Machine learning. New York: McGraw-Hill Education*, 1997.

[8] J. Fonseca, M. Vieira, and H. Madeira, "Detecting Malicious SQL," in *Trust, Privacy and Security in Digital Business* (C. Lambrinoudakis, G. Pernul, and A. M. Tjoa, eds.), vol. 4657 of *Lecture Notes in Computer Science*, pp. 259–268, Springer Berlin Heidelberg, 2007.

[9] F. Valeur, D. Mutz, and G. Vigna, "A Learning-Based Approach to the Detection of SQL Attacks," in *Detection of Intrusions and Malware, and Vulnerability Assessment* (K. Julisch and C. Kruegel, eds.), vol. 3548 of *Lecture Notes in Computer Science*, pp. 123–140, Springer Berlin Heidelberg, 2005.

[10] C. Y. Chung, M. Gertz, and K. Levitt, "DEMIDS: A misuse detection system for database systems," in *Integrity and Internal Control in Information Systems*, pp. 159–178, Springer, 2000.

[11] W. L. Low, J. Lee, and P. Teoh, "DIDAFIT: Detecting Intrusions in Databases Through Fingerprinting Transactions.," in *ICEIS*, pp. 121–128, Citeseer, 2002.

[12] S. Y. Lee, W. L. Low, and P. Y. Wong, "Learning fingerprints for a database intrusion detection system," in *Computer Security - ESORICS 2002*, pp. 264–279, Springer, 2002.

[13] S. Mathew, M. Petropoulos, H. Ngo, and S. Upadhyaya, “A Data-Centric Approach to Insider Attack Detection in Database Systems,” in *Recent Advances in Intrusion Detection* (S. Jha, R. Sommer, and C. Kreibich, eds.), vol. 6307 of *Lecture Notes in Computer Science*, pp. 382–401, Springer Berlin Heidelberg, 2010.

[14] M. Gafny, A. Shabtai, L. Rokach, and Y. Elovici, “Poster: Applying Unsupervised Context-based Analysis for Detecting Unauthorized Data Disclosure,” in *Proceedings of the 18th ACM Conference on Computer and Communications Security*, CCS ’11, (New York, NY, USA), pp. 765–768, ACM, 2011.

[15] A. K. Jain and R. C. Dubes, *Algorithms for clustering data.* Prentice-Hall, Inc., 1988.

[16] A. Harel, A. Shabtai, L. Rokach, and Y. Elovici, “M-Score: A Misuseability Weight Measure,” *IEEE Transactions on Dependable and Secure Computing*, vol. 9, pp. 414 –428, May-June 2012.

[17] E. Costante, J. den Hartog, M. Petković, S. Etalle, and M. Pechenizkiy, “Hunting the Unknown. White-Box Database Leakage Detection,” *To appear.*

[18] R. Santos, J. Bernardino, M. Vieira, and D. Rasteiro, “Securing Data Warehouses from Web-Based Intrusions,” in *Web Information Systems Engineering - WISE 2012* (X. Wang, I. Cruz, A. Delis, and G. Huang, eds.), vol. 7651 of *Lecture Notes in Computer Science*, pp. 681–688, Springer Berlin Heidelberg, 2012.