NTNU

Norwegian University of
Science and Technology

Profile based Intrusion Detection for Internet Banking Systems

Kåre Nordvik Karlsen
Tarje Killingberg

Master of Science in Computer Science
Submission date: June 2008
Supervisor: Torbjørn Skramstad, IDI
Co-supervisor: Lillian Røstad, IDI

Norwegian University of Science and Technology
Department of Computer and Information Science

Problem Description

Machine learning is a central area of intrusion detection systems (IDS). Evaluations of machine learning methods used in different domains reveals that their performance varies substantially within each domain. The goal of this project is to analyze the performance of different machine learning methods within the domain of Internet banking systems. The goal of this project can be divided into two sub-goals.

The first goal deals with the evaluation of data sources and attributes selection, in order to find out which data material is best suited for detection of fraud and misuse. This is conducted by building profiles, which learns the normal behavior for usage of the system at different levels of data detail, ranging from single attribute values to general user behavior. Further, the task includes evaluating the different profiles, attributes, and data sources in order to find the subset that contributes the most.

The second goal is to find and evaluate different machine learning methods with emphasis on performance in this specific domain. The most important evaluation criterion is anomaly detection rate. It is assumed that Internet banking systems execute transactions in batch-mode, and hence, real time detection of abnormal behavior is a secondary requirement, compared to that of the quality of the detection.

Assignment given: 16. January 2008
Supervisor: Torbjørn Skramstad, IDI

Abstract

A review of publications treating security in Internet banking systems has uncovered a practice that finds security by obscurity just as important as actual security measures. The key reason for this is that security measures do not provide a sufficient return on investment by fraud and misuse detection. Hence, the banks have so far taken the risk of providing poor security in their systems, and instead compensated the compromised users. This introduces the need for a cost-efficient, non-intrusive and customizable novel fraud and misuse detection system. This report describes the work done in researching such a system, based on audit data from a highly customized system, and using machine learning methods to provide functionality.

By choosing to use audit data as the primary source of information, data can be gathered from the system in close to real-time, without interfering with the existing functionality. Audit mechanisms are commonly present in any system, thus they are the primary source from which a non-intrusive solution can be obtained.

This report proposes the use of profiles to learn a baseline of the normal interaction between a user and the system. Each profile looks at the available data at different levels of abstraction so that different properties in the behavior can be learned. By using these profiles, each profile can be refined to learn its level of abstraction, while still providing a complete picture of a user's behavior.

Machine learning methods can be used to automatically learn a baseline for normal behavior based on a set of historical data. The learned behavior can then be used to compare new instances against the baseline in order to classify them as normal or abnormal. Abnormal behavior would then be an indication that a user is conducting illegitimate activity.

The results of our proposed solution are satisfactory. We are able to detect anomalies by different profiles and data sources. However, there are issues when it comes to evaluating the solution. Since we are trying to detect novel fraud and misuse behavior, there is no apparent test set to compare against. Some options for evaluation of anomaly detection exist. However, we found none of these to be satisfactory. Further research needs to be conducted in this area before a functional solution can be created. This report uses results and experiences to create a foundation for such further research.

Keywords: Fraud detection, Internet banking systems, machine learning, anomaly detection, defined context

Acknowledgements

We would like to announce our gratitude for all guidance, feedback and support received during this project. In this matter, our adviser, and main supervisor, Espen Fossen at Kantega AS has been our main resource. Kantega AS has provided office space, computers, and free lunch. Also, supervisor at IDI, Lillian Røstad, has provided valuable feedback. Further, for this project, we have acquired access to a test system containing audit logs from an actual banking application. We would like to express our appreciation for this.

Our friend, Hans-Christian, has provided invaluable thoughts on the content and quality of our report. In addition, friends and families have supported us and supplied other points of view. Finally, we thank our beloveds, Carina and Karoline, for believing in us and providing support and understanding during late work hours in the final stages of this project.

Without your support this work would not have been possible.

Thank you.

"To predict the behavior of ordinary people in advance, you only have to assume that they will always try to escape a disagreeable situation with the smallest possible expenditure of intelligence".

Friedrich Nietzsche
German philosopher (1844 - 1900)

Preface

This report is the final result of the master's project in the 10th semester of a master of technology degree in computer science. It is elaborated by graduating students Tarje Killingberg and Kåre Karlsen. The assignment is formulated in collaboration with Kantega AS and the group for Design and Use of Information Systems at the Department of Computer and Information Science (IDI) at the Norwegian University of Science and Technology (NTNU). In this report we conduct research and provide results in the area of anomaly detection in users' interaction with a large Internet banking system, with the goal to detect fraud and malicious behavior.

Brief table of contents

Contents

List of Tables

List of Figures

List of Equations

Acronyms

ANN	Artificial Neural Network
BMU	Best Matching Unit
CLF	Common Log Format
CPU	Central Processing Unit
CSS	Cascading Style Sheets
DoS	Denial of Service
DDoS	Distributed Denial of Service
FSS	Feature Subset Selection
HIDS	Host based Intrusion Detection System
HTTP	HyperText Transfer Protocol
ICD	Idealized Character Distribution
IDE	Integrated Development Environment
IDI	Department of Computer and Information Science
IDS	Intrusion Detection System
IP	Internet Protocol
JDT	Java Development Tools
KID	Customer ID
LVQ	Learning Vector Quantizer
MADAM ID	Mining Audit Data for Automated Models for Intrusion Detection
MB	Megabyte
NA	Not Available
NIDS	Network based Intrusion Detection System
NTNU	Norwegian University of Science and Technology
PC	Personal Computer
RBF	Radial Basis Function
ROC	Receiver Operation Characteristic
SQL	Structured Query Language
SSH	Secure Shell
SSN	Social Security Number
SVG	Scalable Vector Graphics
SVM	Support Vector Machine
TCP	Transmission Control Protocol
Trojan	Trojan horse
UML	Unified Modeling Language
URL	Uniform Resource Locator

WASC-TC	Web Application Security Consortium: Threat Classification
WEKA	Waikato Environment for Knowledge Analysis
XML	Extensible Markup Language

Mathematical acronyms

γ	The bound of the radius to the hyperspace
ϵ	The tolerance of termination criterion
ν	Ratio of normal points
$\tilde{\mathbf{v}}$	Vector
$\mathbf{f}(\tilde{\mathbf{v}})$	Classification function f defined on input $\vec{v}$
$\mathbf{p}$	Configuration parameter
$\mathbf{f}(\tilde{\mathbf{v}}, \mathbf{p})$	Classification function f defined on input $\vec{v}$ and configuration p
$(\tilde{\mathbf{v}}_\mathbf{i}, \mathbf{y}_\mathbf{i})$	Vector number i, $\vec{v_i}$, and its associated truth value, y_i
$\mathbf{E}(\mathbf{y}, \tilde{\mathbf{v}})$	Error function resulting in 0 if y equals $f(\vec{v})$, otherwise 1.
$\mathbf{b}$	Bias term
$\mathbf{n}$	Number of instances *(SVM)*
$\tilde{\mathbf{w}}$	Normal vector
$\mathbf{\Phi}(\mathbf{x})$	Transformation function on x
$\mathbf{K}(\tilde{\mathbf{v}}_\mathbf{i}, \tilde{\mathbf{v}}_\mathbf{j})$	The kernel function ($K(\vec{v_i}, \vec{v_j}) = \Phi(\vec{v_i}) \cdot \Phi(\vec{v_j})$)
$\mathbf{R}(\mathbf{p})$	Risk estimate as a function of p
$\mathbf{P}$	Probability distribution
$\mathbf{sgn}(\mathbf{x})$	Function that returns an integer indicating the sign of x
$\mathbf{S}$	Simple subset
$\neg\mathbf{S}$	The negation of S
$\mathbf{exp}(\mathbf{x})$	The exponential function (e^x)
$\mathbb{R}$	Real number
$\mathbf{O_i}$	Output node number i
$\mathbf{I_i}$	Input node number i
$\tilde{\mathbf{V}}$	Input vector
$\tilde{\mathbf{W}}$	Weight vector
$\mathbf{v_i}$	Value numer i of $\vec{V}$
$\mathbf{w_i}$	Value numer i of $\vec{W}$
$\mathbf{d}(\tilde{\mathbf{V}}, \tilde{\mathbf{W}})$	Distance between vectors $\vec{V}$ and $\vec{W}$
$\mathbf{d_E}(\tilde{\mathbf{V}}, \tilde{\mathbf{W}})$	Euclidean distance between vectors $\vec{V}$ and $\vec{W}$
$\mathbf{t}$	A variable holding the current time step
$\sigma(\mathbf{t})$	Radius as a function of t
$\sigma_\mathbf{0}$	The radius for $t = 0$
λ	Exponential denominator for $\sigma(t)$
$\mathbf{N}$	Number training cycles
$\mathbf{Q}(\mathbf{t})$	The quantization error as a function of t
$\mathbf{X}(\mathbf{t})$	A stochastic process

$\mathbf{P}(\mathbf{A}\|\mathbf{B})$	The probability of A, given B
$\mathbf{m}$	The order of a Markov chain
$\mathbf{n}$	Number of output nodes *(ANN)*

Chapter 1

Introduction

Summary

This chapter contains an evaluation of the current security of Internet banking systems. These findings make up the motivation for this project. Further, we define a taxonomy of the threats to such a system, such that we may understand and define an attack, an attacker, and the relationship between them. This is used to define three threat scenarios, which are of special concern. Based on the preceding discussion, we formulate a problem definition, in conjunction with four main research questions that need to be answered in order to provide a solution for the defined problem. Finally, we describe the layout of the rest of this report.

1.1 Motivation

There appears to be an impression in the general public that Internet banking systems are secure [1], but when we scratch the surface we soon realize that this impression is based on false premises. Several references [2], [3] and [4] describe a series of security holes and vulnerabilities not covered by the banking system's own security measures. An increasing number of successful Internet frauds seem to support these findings [5]. Article [6] and [HMT06] discuss the possibility that banking authorities think that security is not worth the cost. This has been confirmed in [7] by a survey concluding in that most fraud detection software costs more than the prevented loss from detected frauds. Further, it appears that they instead consider security by obscurity as a good and cheap alternative. Surveys also show that users find the apparent trust and reputation more important than the actual security level of the system [NAH05] and [8]. Consequently, this result in low security costs, but also a higher possibility of successful fraud attempts. So far, the banks have compensated users having been victims of fraud. This has however been a voluntary practice conducted in order to ensure user satisfaction, not an imposed legislation [9]. The question still remains to see how this practice will be conducted when the cost of compensating the users will exceed the cost of reduced trust and image.

The Internet banking system's security should however not be considered lightly. There are several reasons for this, beyond the obvious false sense of security felt by the users. Firstly, the practice of security by obscurity is only valid as long as the security methods are obscured [HMT06]. Once the obscurity is threatened by a successful fraud attempt, the security measures of that system would fail. Obscurity is always a valid requirement for any security measure, but it should never be used as a security measure itself. Secondly, the rate of successful fraud attempts so far has not been significant, compared to other types of fraud. Other areas of money transactions, such as credit card transactions have had a much higher risk of fraud. The trends in fraud schemes seem to change as fraud detection is implemented in systems. This is true for credit cards, where fraud preventive systems have lead to a decrease in credit card fraud by 16% in the UK over the last two years [5]. The rate of successful non-credit card fraud attempts using the Internet experiences a different trend. The rate of successful fraud attempts against Internet banking systems increased by 44% from 2005 to 2006 to a total value of £33.5 million in the UK alone [5]. Other countries have experienced similar growth [10]. No global statistical data material was found pertaining to this subject, but the trend in the UK should be valid for most western countries.

Therefore, cost-efficient security measures, which can detect fraud and fraud attempts within Internet banking systems, seem increasingly relevant and valuable. The goal of this project is to investigate options in the detection of fraud. In this report we present a solution for detection of fraud and fraud attempts based on machine learning methodologies. It will be based on that most traffic in Internet banking systems is legal, and can be used to determine a pattern of normal behavior. New traffic can be compared to this pattern to find anomalies, which could indicate fraudulent activity.

1.2 Taxonomy of threats

A logical approach when wanting to detect all types of attacks would be to create a complete taxonomy of all possible threats. However, such an approach is outside the scope of this project and is still an area of much research activity. Instead, we will attempt to describe the subject in a general way using previous research and our own additions as we see fit [CM02]. The following is a generalization of the threat problem divided into two sections, attacker and attack, and a final section with a description of three threats of specific relevance to this domain [How98], [Lou01]. The first section, *attacker*, tries to understand the motivation and characteristics of an attacker. The second, *attack*, shows a general model of how attacks could be performed. The third, *scenarios*, describe three scenarios for the most relevant attackers and attack paths.

1.2.1 Attackers

In order to understand how a system is at risk, an understanding of the benefits of attacking the system is needed. Hence, knowing the motivation and mindset of an attacker is critical when trying to defend against and detect attacks. The first obvious distinction that can be made is between external and internal attackers. The internal attacker has trusted access to the system at risk, and misuses this access to perform malicious actions. The external attacker does not have such access, but has to gain it by attacking the system itself or by attacking a trusted user. The internal attacker has advantages since he does not need to bypass the first level of defense, which prevents access by unauthorized users to the system.

The second distinction of attackers is by technical skill. The level of skill varies from professionals, who can perform customized attacks on the system, to unskilled users, who use automated tools and scanners created by others.

The third and final distinction relates to the objective of the attacker

when attacking the system. Some attackers have a specific objective such as some information or code. Others may just want to cause the system general disruption, e.g. as a protest or to prove poor security. Yet others might just be exploring and testing the system for fun, stretching security measures, and testing accessibility. These attacks have a varying degree of probability and the damage varies widely. This is because the target varies for each type of objective, from confidential data and assets, to simple tests of security barriers.

1.2.2 Attacks

The core of an attack is to bypass the security policy of the system. In achieving this, the attacker gains the capability to perform operations which should not be allowed. In order to do this, a system process needs to be accessed in an unforeseen way such that it alters functionality. This could be done by starting, controlling, hijacking or aborting a system process such that the attacker gains access, performs commands or makes some alteration in system data and assets that goes beyond the security policy restrictions.

A key term when discussing attacks is vulnerabilities. Every attack needs to exploit a security vulnerability in order to bypass the security policy. Security vulnerabilities may exist in software implementation, such as input validation errors allowing Structured Query Language (SQL) injections or buffer overflows. Vulnerabilities may also originate from misconfiguration of systems and programs, for instance wrong management of permissions. Finally, system or protocol design can inhibit security vulnerabilities. Figure 1.1 shows a general model of how a *flaw* (vulnerability) allows *access* of *assets*.

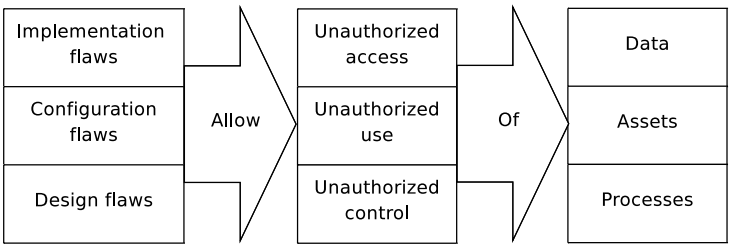

Figure 1.1: Model of attack methodologies

To make a classification of the different types of attacks is outside the scope of this project. However, some distinctions can be made. Firstly, attacks can be separated by which vulnerability they exploit. Secondly by

what privileges the attacker would gain, and, finally, by which resources he tries to access.

Another key distinction that can be made is between remote and local attacks. The terms remote and local do not reflect the physical location of an attacker, but rather the distinction of privileges. A model that represents the different paths and attackers was proposed in [Zan06]. However, due to the problem domain at hand, some modifications have been made. A local attacker has access privileges to the system, while a remote attacker has not. The types of attackers use different attack paths, each of which can develop into different types of attacks. The local attacker, or insider, would try to gain more privileges than he currently has. The remote attacker, or masquerader, has several attack paths he might consider. One path is a direct attack aiming for privileged local access using a remote exploit or a legal user's credentials. A second path is a break in through the non-privileged local access. A third path is attempting to attack through a local user using a Trojan. This final path was not represented in the original model, but is none the less an important and highly relevant one.

1.2.3 Scenarios

Due to the problem formulation and domain restriction of this project, the models proposed in the previous sections can be simplified to enclose three attack scenarios. In order to understand the type of attackers and attacks relevant for this project, we have described three scenarios which describe the most important attackers and attack paths we have to consider.

The insider

An insider is someone who has legitimate access to the system, but misuses this access to perform illegal operations. While doing so, he may mimic the actions of a masquerader, see section 1.2.3, in order to avoid detection and face the consequences of his actions. The main problem with an insider attack is that it has bypassed the first layer of defense, since the user has legitimate access to restricted parts of the system. Figure 1.2 shows an illustration of the insider scenario. It shows how a legitimate user can use his access to the system to attack it from the inside, gaining access to resources outside his legitimate areas.

When using a profile for the user's normal behavior, one could detect deviations from some of the profile's parameters, while others would remain the same. For instance, the insider could be carrying out his normal work during daytime, and exploit the system at night. He would then deviate from

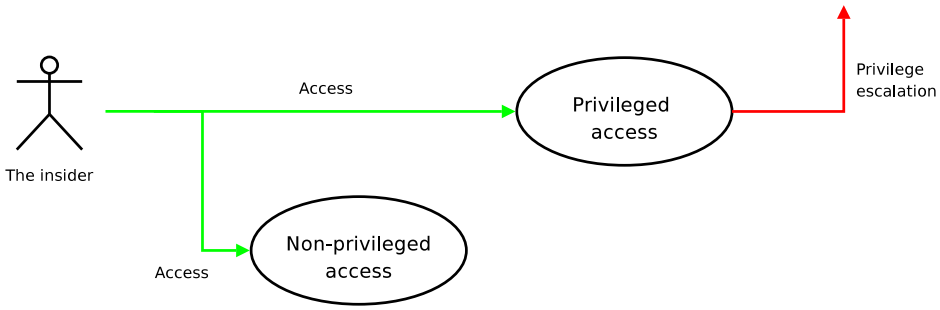

Figure 1.2: The insider scenario

normal login time, and perhaps also session time as well as time since last login. He could be using his normal computer, since it would look even more suspicious if he used someone else's, and the ratio between successful and failed logins would perhaps look normal. A problem is that if the insider is aware of what parameters the behavior profile consists of, he can behave in a way that makes it look like a masquerader has used his account.

A detection scheme learning normal system usage would probably not give a satisfying rate of accuracy on detection of insider attacks. Hence, it would be necessary to provide an additional scheme for detection of these kinds of attacks. A key issue with these kinds of attacks is that the user accesses areas and functions that he normally does not have access to. In order to gain this access, he has to probe, stress, and test the system for weakness points, security gaps, and logical breaches. This would imply that the user needs to perform actions and use functionality not normally used. Hence, by looking for probing actions and break-in attacks from the inside, one could gain additional detections on anomalous behavior.

The masquerader

A masquerader is someone who mimics a legitimate user in order to gain access to restricted information or functionality. There are two types of masquerading techniques used. The first is to get the login information needed to access the system, and then use this information from some remote location. The second technique is harder to detect if it is performed successfully. It is called session hijacking and occurs when the masquerader hijacks another user's connection after he has logged in. Hijacking a session from a remote location has proved to be hard, since sessions often are related to an Internet Protocol (IP) address by the host [GSC06]. Hence, the attacker needs to receive IP-packets not addressed to himself. Hijack attacks done by Trojans can overcome this problem by being executed on the legitimate user's

computer. This type of attack is defined in the next scenario.

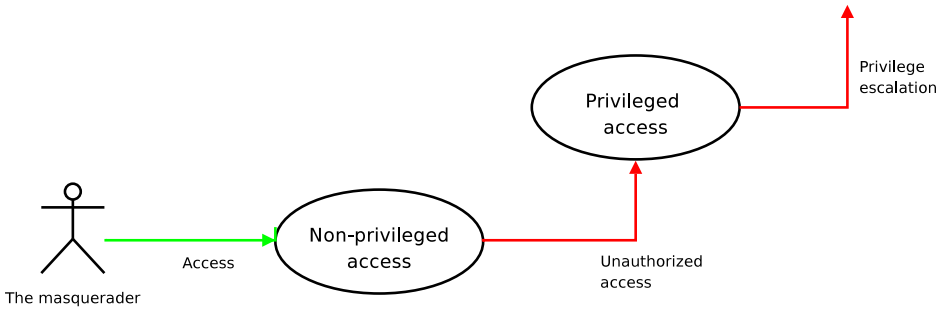

Figure 1.3: The masquerader scenario

A masquerader can be discovered by behavior that differs from that of the ordinary user, such as logging in at different hours and from different clients than the regular user. Furthermore, the session time would probably be different. Since the masquerader already has acquired the user's login information, he would probably not make any greater number of failed login attempts than the user whose identity he assumes. However, a strong indication of a masquerader being in the system is that some user shows abnormal behavior, and that this behavior has been preceded by a password-guessing attack against the user's account. The method by which an attacker has obtained the login credentials is not of interests to us, as long as it has not happened within the system.

Figure 1.3 illustrates an example of the masquerader threat. It shows how the masquerader uses a legitimate user's login credentials to access restricted areas of the system. This access could be used to perform malicious actions within the legitimate user's personal area, or elevated further to gain access to other parts of the system.

The Trojan

A Trojan horse, or just Trojan, is in this context a piece of software that may appear to perform some actions, while it in fact is performing other hidden and malicious actions. One of the most common uses of Trojans involves installing them to open backdoors on regular users' computers, such that the attacker can monitor and assume control of their functionality at any given moment. Here, the threat against Internet banking systems lies.

As shown in figure 1.4, the malicious user can use a Trojan to take over a legitimate user's session in the banking system, without any notice from the user. The figure illustrates how a malicious user can use a Trojan to

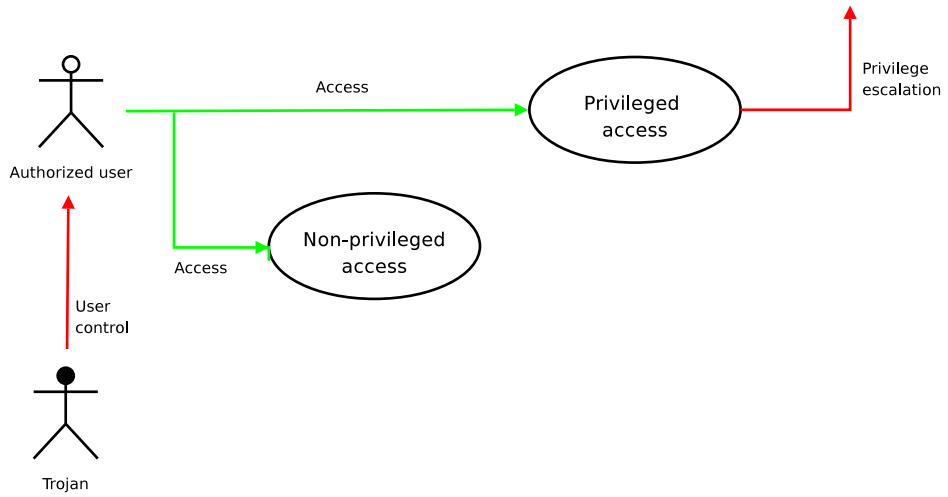

Figure 1.4: The Trojan scenario

access the banking system. Firstly, the attacker needs to get the legitimate user to download and run the Trojan software. Then, the Trojan software opens a backdoor on the legitimate user's computer that can be used by the malicious user to access the computer and its functionality. When this backdoor is installed, the malicious user can monitor the legitimate user's actions and wait for the right moment to hijack the control.

Obviously, this is a type of attack that is very hard to detect, due to the fact that it is a session of a legitimate user that is taken over through the user's computer. The obvious place to protect against this type of attack would be on the user's computer. Unfortunately, it seems to be little focus on the security of user's computers as a threat to Internet banking security. Many banks recommend security measures for user's computers, however these have so far been voluntary [11]. In either case, the existence of this threat must be acknowledged in order to find methods to detect it.

1.3 Problem definition

When the surrounding threats to the system have been identified, we can begin to elaborate a problem definition that will be answered by this project. By looking at this problem definition, in conjunction with the motivation of the project and the described scenarios, we can define a set of research questions that need to be answered in order to create a satisfactory solution to the defined problem.

In this section we state the problem definition that is the motivation for

this project. Finally, we create our main research questions, which will be the framework for the proceedings and goals of this project. By answering these questions we will also provide an answer to the defined problem.

Based on the motivation described in section 1.1, we see that the security measures for Internet banking system are not satisfactory. We have identified a general attack and attacker taxonomy, which we have specialized for our domain by three scenarios in section 1.2.3. These scenarios describe not the means of an attack, but rather the origin of one. This is important since different origins would have different properties and behavior. Further, we wish to use this information to make a contribution to improve the weaknesses and problems found in the motivation section.

We do not wish to create a new type of security measure which will replace any existing ones. Rather, we wish to make an addition which can be added to any system and would join and cooperate with the current security measures in order to increase the security level in general. By using the data material available from audit logs, one can create a solution that does not interfere with the system and security measures, while still having continuously updated information. This information could then be analyzed by an automatic processing unit in order to detect malicious behavior and fraud that has bypassed other security measures.

Such an automatic processing unit can be implemented with the use of a machine learning method that learns how the system normally behaves. It could then use its acquired knowledge to detect when strange and abnormal behavior appears, and alerts the system of this. Based on this we can define a problem definition of what we wish to accomplish:

> *Evaluate different audit logs and machine learning methods, in the context of anomaly intrusion detection, focusing on detection of fraud and malicious activity, limited by Internet banking applications, thus resulting in an evaluation of the various methods and audit logs.*

The project will analyze the available audit log data, in combination with machine learning methods, in order to see which of them contributes more significantly than others in detection of abnormal behavior. Furthermore, various sets of properties will be grouped into detection profiles, and their performance will be evaluated within this domain. The most important evaluation criterion is detection rate.

Based on this problem definition, and the attack scenarios, we are able to identify five basic research questions, which will need answers in order to fulfill these objectives of this project. The research questions are listed next.

RQ1: Does the information contained within audit logs contain enough information about normal behavior to create a profile defining all types of normal behavior?

RQ2: Is it possible to detect malicious behavior and fraud based on audit records from different levels in the system?

RQ3: Will malicious activity and fraud always be classified as abnormal data?

RQ4: Can malicious behavior be detected by the use of profiles?

1.4 Report outline

This report is organized as follows:

Chapter 2 consists of a study of background material concerning the essential parts of this project. Firstly, a study of the project's context is conducted. This includes information such as system overview and system threat picture. Secondly, we go into the field of fraud detection, beginning with a state of the art study of fraud detection research. Finally, we describe the area of machine learning and methodologies concerning this field. This includes a general introduction, as well as specific descriptions of the methods used in this project.

Chapter 3 tries to form a scientific basis for this project. It consists of research questions we wish to answer and a hypothesis we wish to test by our work. Moreover, we describe a plan for how this project should progress in order to reach our goals. We then describe the instruments used to carry out the actual research, including implementation and testing. Finally, we look at threats to validity that could influence the project's results, and means for reducing these threats.

Chapter 4 consists of the designs and configuration done in this project. It consists of a set of profile suggestions that are design to detect different kinds of fraud and malicious activity using different kinds of levels of abstraction. A subset of these profiles is then selected in order to have a manageable set to work further with. Then we start to prepare the available data so that it can be used my machine learning methods in an efficient way. Next, we find settings and configurations for the different machine learning models used to implement the selected profiles.

Chapter 5 consists of the results provided by the different profiles and machine learning methods. Here, we analyze the results and looks at how they answer the research questions. We also consider the validity of the results due to different threats.

Chapter 6 presents our findings and compares them to the background material. The hypothesis is tested in order to see how our results could prove or discard it. We then review the project work, and outline areas for further research.

Chapter 2

Background

Summary

In this chapter we describe the context of this project. The context consists of two parts. First, it contains a description of the system at hand, in conjunction with a review of the available data sources. Secondly, it describes a listing of security threats related to any online system. Next, we look at the state of the art in research on fraud and anomaly detection, in order to understand how mature these research areas are. Then, we provide a general introduction to intrusion detection systems, such that it can be seen how fraud and anomaly detection fit in a bigger picture. Finally we describe machine learning methodologies, in which both a general introduction to the field and a thorough review of different methodologies are given.

2.1 Context

We have established a cooperation with a large Norwegian bank, which offers an Internet banking application. They have provided us with insight into the technologies and methodologies involved in such a system, as well as actual log data from several layers of their banking application. In addition, we have collected information on Norwegian Internet banking applications in general. This section provides an introduction to the context of this project.

2.1.1 System overview

To our understanding, most Norwegian Internet banking applications adhere to the following superficial description, which is illustrated in figure 2.1. The user interacts with the banking application by means of a client, in most cases a Personal Computer (PC). When the user performs defined actions, the client sends requests to a corporate router, which forwards them to one of several application servers. These servers then process the requests, and determine responses to be sent back to the client.

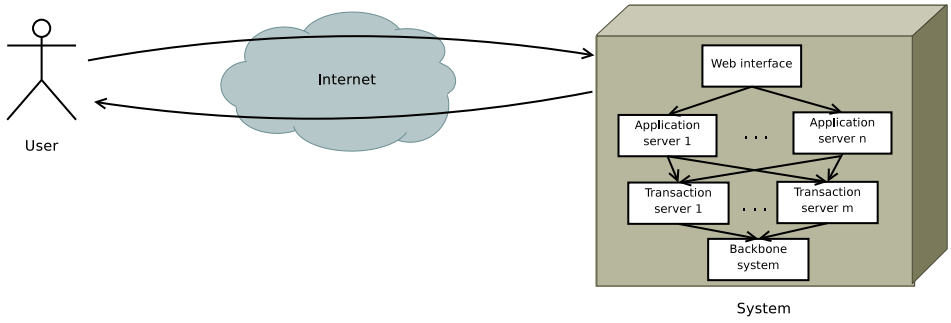

Figure 2.1: Overview of a general system

If the request from the client involves some transaction, a new request is generated in the application server, and placed in a queue. In batch mode, this queue is processed, and the requests are sent to a third party's server where the transactions are performed. This means that real-time detection of misplaced or fraudulent transactions is not critical as the transaction requests are queued for some time before the transactions are carried out.

An overview of the system we will work with is shown in figure 2.2. The structure is hierarchical with different layers concerned with different levels of the functionality. The data sources originate from these layers of the system. This means that data is collected at each layer, but information about the

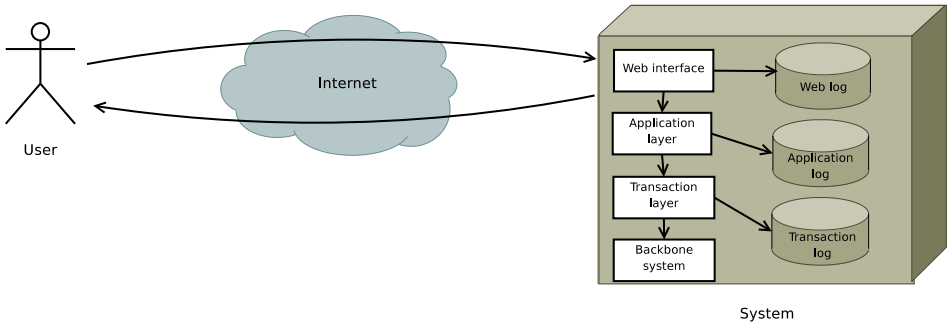

Figure 2.2: Overview of the project's system

activity of a user is separated between the logs. Collecting a complete picture of a user's activities would involve tracking the user through the different logs.

2.1.2 Data sources

The data available to this project is log data from the different layers of the system, as shown in figure 2.2, consisting of access logs from the web server application, application logs from the underlying application layer, and transaction logs from the transaction layer. There are also two special logs for each of these log types. They contain special incidents, where it is known that malicious activity has occurred. In the following we describe the properties and contents of the three types of logs.

Web log

The web log collects data about the user at the highest layer of the system. The basic properties of the web log can be seen in table 2.1. Every interaction with the system will be recorded in this log. As a result, a huge amount of data is collected at this layer. The interaction with the system is recorded by a standard logging scheme which provides structure and consistency of the instances. Since this log contains the web application's recordings it is a collection of HyperText Transfer Protocol (HTTP) requests, see section A.3. Most notably, it specifies the origin of the request and the requested resource, but also other information. A more formal review of the instances from this log is conducted in section 4.3.1.

Application log

The application log contains the recordings from the application layer. The application layer is where most of the core functionality of the system is lo-

Table 2.1: Web log properties

Web log	
Instances	$3.9 \cdot 10^7$
Time frame	120 hours
Number of fields	11
Size	1022 MB
# Sessions	225000
On consistent form	100%

cated. It provides a connection between the web interface and the transaction layer, and it provides a connection between the web log and the transaction log. Since the application layer contains much varied functionality, the log inherits this characteristic. It includes many different types of log instances, and does not follow a uniform format. However, instances of the same type follow a common format. The basic properties of the application log can be seen in table 2.2.

Table 2.2: Application log properties

Application log	
Instances	$2.3 \cdot 10^7$
Time frame	120 hours
Number of fields	6
Size	395 MB
On consistent form	47%

Transaction log

The transaction log is a more specialized log which collects information regarding the main functionality of the system, namely movement of an amount of money from one account to a second. Although this log contains a great amount of other information besides transactions, we have chosen to call it transaction log, since we consider this the most important information in it. Each transaction is recoded along with a set of required fields and a set of optional fields. It is obvious that the difference between instances is limited, but variance between attribute values could be large. It is also an important aspect that the transaction log contains values that are directly provided by the user, thus the accuracy and values cannot be trusted at the same level as

values provided by the application. The basic properties of the application log can be seen in table 2.3.

Table 2.3: Transaction log properties

Transaction log	
Instances	$3.1 \cdot 10^7$
Time frame	120 hours
Number of fields	18
# transactions	529000
Size	1600 MB

Tracking a user between the different logs can be a challenging task. This is due to the fact that there is no global identifier for a user across the different logs. Each log usually gives the session, request, or transaction its own unique identifier. In order to extract a complete set of data for a session, learning these identifiers, and mapping them together, is necessary. In the logs, users can be identified by for instance, username, IP address, session identifier, transaction identifier and thread number.

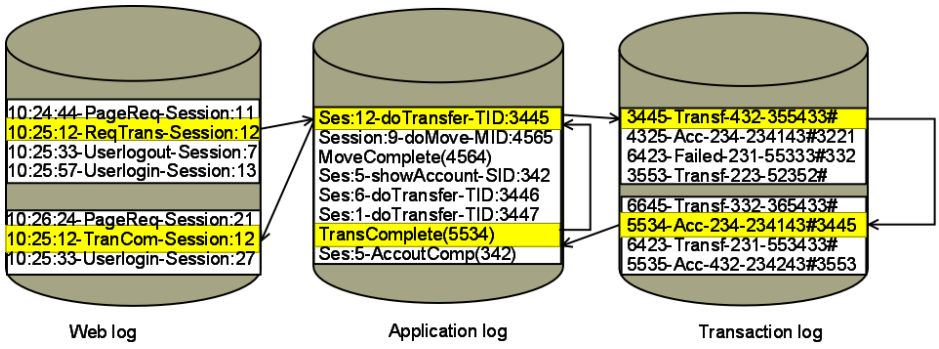

Figure 2.3: Tracking a single user between logs

Figure 2.3 shows how a user's interactions with the server are recorded in different logs, and how these interactions can be tracked across the logs. This example provides an illustration of the challenging task of tracking a user's session across the different logs. A single transaction is identified by four different identifiers during its course through the logs. The example, although only being a general example, has the same form and structure as to how users are identified in the log material we have available.

2.1.3 Threats

The Web Application Security Consortium: Threat Classification (WASC-TC) is a cooperative effort to clarify and organize threats to the security of a web site [Con]. It will be used as a basis for defining the possible aspects of attacks that have to be considered when creating profiles for normal behavior. It classifies threats into six different types: authentication, authorization, client-side attacks, command execution, information disclosure, and logical attacks. The WASC-TC creates a good foundation for structuring possible attacks. However, due to the scope of this project, two additional types of attacks will be added to the list: suspicious activity and fraud. Hence, the possible types of attacks are:

- Authorization
- Command execution
- Information disclosure
- Logical attacks
- Fraud
- Suspicious activity

The following sections consider each of these attack types; describe them, their characteristics and behavior, and what level of abstraction they could be detected on.

Authorization

Authorization considers attacks that threaten the system's method of determining if a user, service, or application has the necessary permissions to perform a requested action. For example, a user should in most cases only be able to view his own credentials and account information, transfer money only from his own accounts, etc.

Subclasses of authorization attacks consist of credential/session prediction, insufficient authorization, insufficient session expiration, and session fixation. These can be classified into two groups: hijacking of another user's session, and restriction of a user's access rights. Figure 2.4 shows a basic example of the authorization threat. An attacker uses false login credentials to gain access to restricted functionality of the system.

Detection of session hijacking could be done by looking at the client's IP address. A sudden change in the client's IP address could indicate that a

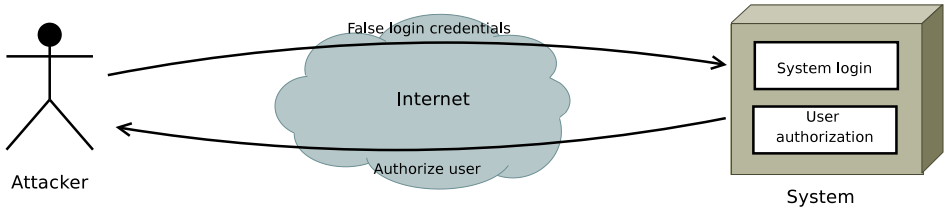

Figure 2.4: Threat concerning authorization

remote connection has stolen a user's session identifier and is using it to access restricted information. However, session hijacking could be done by other methods too, for example by running malicious software on the legitimate user's computer, thereby overtaking the session without the user being aware of the fact. This type of attack is harder to detect, but it may be done by looking at indication of robotic activity.

The second group, concerning access rights, is characterized by a user requesting abnormal information or services. By creating a baseline for normal requests for a user, one should be able to detect attempts of illegal access. This would involve analysis of attributes sent from the client that contains requests for information or services.

Command execution

Command execution covers attacks designed to execute remote commands on the system. This type of attack is possible whenever the system utilizes user provided input to complete requests. If the processing of such input is done insecurely, the system may be vulnerable to input that alters command execution. Attacks that fall within this category include buffer overflow, format string attacks, LDAP injection, OS commanding, SQL injection, SSI injection, and XPath injection[1]. Figure 2.5 shows a general example of how a command execution attack is conducted. The attacker submits input that includes commands. These commands are executed by a server side application, which then manufactures some result.

These attacks could be detected by analyzing the input passed by the user to the server. However, it is a difficult task to detect such attacks, as the commands may be hidden within data that are semi-random, such as names, account numbers, Customer IDs (KIDs), notes, etc., so a direct approach, i.e. trying to learn all possible values for such attributes would be unsuccessful.

[1]For further information about these types of attacks, we refer the reader to [PB05] and [NTGG+05]

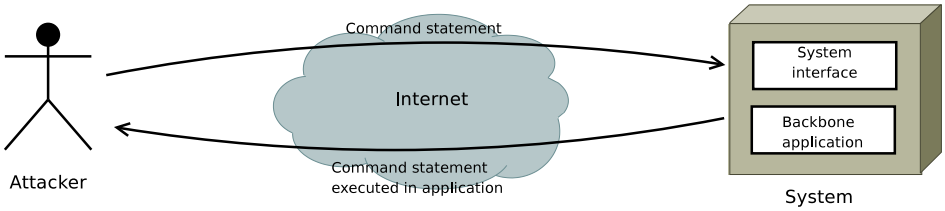

Figure 2.5: Threat concerning command execution

By taking an extra step however, the values could still be used to create a baseline for normal input.

By looking at the statistical properties of the input it should be possible to separate linguistic text, such as names and addresses, and semi-random numbers, such as account numbers and transaction sums, from malicious input. The statistical properties could be character distribution, attribute length, tokens used and attribute presence or absence. These properties have been proven successful when separating malicious input from valid input [KVR05].

Information disclosure

Information disclosure covers attacks designed to acquire system specific information, such as location of files, version numbers, backbone-server addresses, etc. The attacks in this category are directory indexing, information leakage, path traversal and predictable resource locations. In figure 2.6, an attacker requesting a file containing restricted information is illustrated.

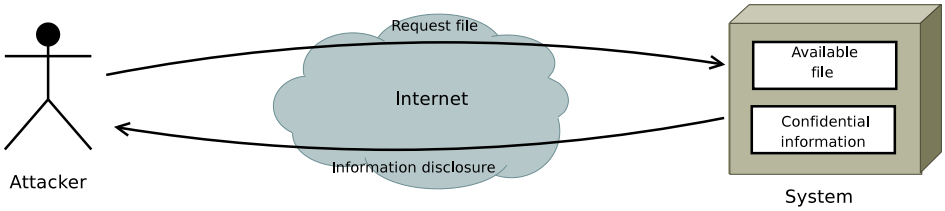

Figure 2.6: Threat concerning information disclosure

Prevention of information disclosure cannot be handled by analyzing normal/abnormal behavior alone. Sensitive information could be revealed, for instance, as comments within HTML code or by server configuration. These issues are not of interest in this context. The interest lies in the information disclosed that could be detected by looking at the interaction between the client and the server.

There are two places to look for information disclosure attacks: user requests and server responses. By defining the normal requested attributes and server responses, abnormal requests for items, attributes or resources should stand out and be detected. Since this could be due to logical flaws or configuration errors, it would be a good idea to look not only at the user requests, as with the other issues, but to also look at the information sent back to the client. These types of attacks entail the need to look at both streams of information between client and server.

Logical attacks

Logical attacks cover attacks that abuses or explores the web application's logical data flow. Application logic is the expected procedural flow used in order to perform a certain action. Attacks classified under this category are abuse of functionality, denial of service, insufficient anti-automation, and insufficient process validation. Figure 2.7 shows an example in which an attacker bypasses the login procedure in order to get authorized.

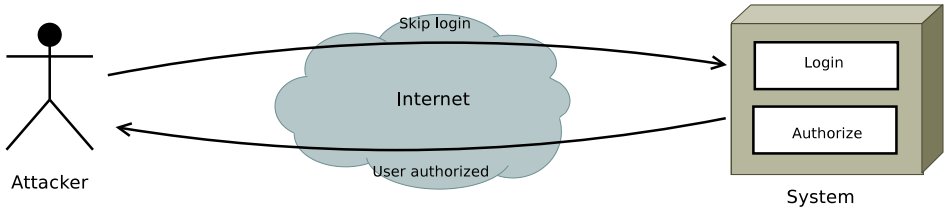

Figure 2.7: Threat concerning logical attacks

In order to detect such attacks one has to look at a higher level of abstraction in comparison to the previous attack types. One will not be able to find such an attack without looking at several requests in combination. There are two levels of abstraction to consider, single user attacks and multiple user attacks.

Single user logical attacks could be detected by looking at complete sessions. In this way, the normal traversal of pages, and navigation between them, can be learned, in order to detect when somebody tries to skip a critical step. Further, other statistics in the session data should be considered, such as number of request for each service and session time.

Multiple user attacks are coordinated attacks that compromise some part of the system's functionality. The most typical type of attack is the distributed denial of service (DDoS) attack, in which several clients, usually infected with a Trojan and controlled by remote users, repeatedly request time consuming services on the system, thereby causing an overload.

Denial of service attacks are also possible from a single user, but due to the size of the system at hand, a single user could probably not cause enough workload. In order to detect DDoS attacks one have to monitor the complete system for a given time frame. When a norm for the system's load and active sessions has been determined, abnormal values can be found that may indicate that a distributed attack is being performed. Illustrated in figure 2.8 is a group of attackers conducting a distributed denial of service attack on the system and overloading its capacity.

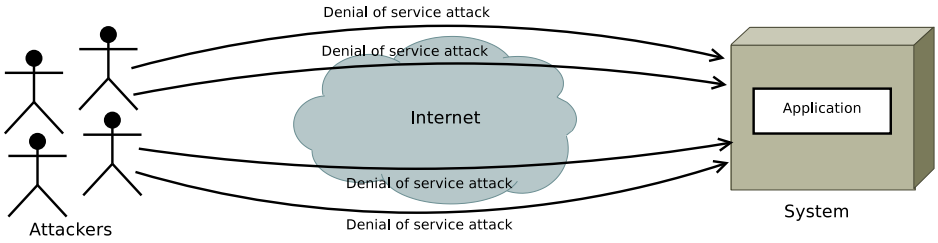

Figure 2.8: Threat concerning distributed attacks

Fraud

To classify fraud as an attack category is incorrect. However, it is an area which needs special attention due to the scope of this project. Fraud detection can function as a last line of defense if an attack should pass unnoticed by all other detection mechanisms. Fraud detection considers the transactions performed, and evaluates them with regard to previous transactions and other information collected about the user.

There is no simple answer as to what separates normal transactions from fraudulent ones, but there are several existing solutions that address this problem. By comparing a transaction with a set of learned rules based on historical transactions and knowledge about fraudulent properties, these solutions are able to detect such events with some degree of success. Figure 2.9 shows an example of a fraud indicative action involving a rare and strange transaction.

Suspicious activity

Suspicious activity is not an attack category, but a category consisting of elements that could indicate that a user is preparing an attack or looking for vulnerabilities. Since the only information available for analysis originates at the host, there is no way to detect classical network probing. However,

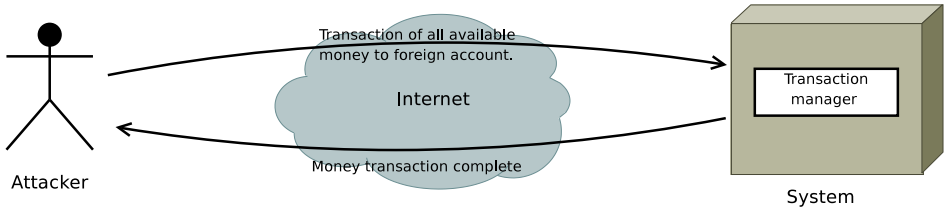

Figure 2.9: Threat concerning fraud

detection of application probing is just as relevant and indicative of such activity should be possible to extract from audit records. Figure 2.10 shows an example of a user probing the application by making an unusual request.

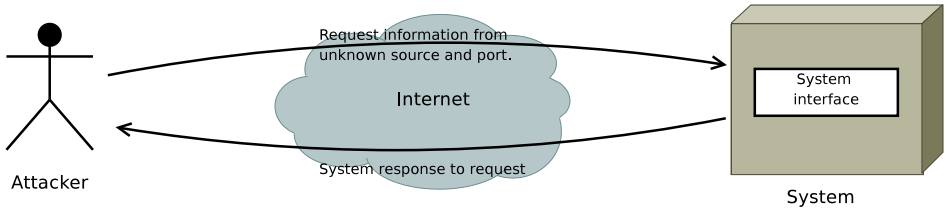

Figure 2.10: Threat concerning suspicious activity

There are several aspects to consider in terms of suspicious activity. The most relevant level would be a total collection of a user's activity. If only individual sessions are considered, it would be easy to evade detection by doing the probing across several sessions. Looking for probing activity by combining several users would be extremely complex and not relevant within this project's scope. However, we see a collaboration between several users as a future security issue. Whenever several malicious users are able to collaborate, there is a risk that they can behave in such a way that detection systems are taught to judge malicious activity as normal.

Other types of activities that fall under the category of suspicious activity are indications of automated activity and robotic activity. Several of the latest attacks have been done by Trojan software on legitimate users' computers [12]. Some research [PPLC06] indicates that activity done by robots would differ from human activity, and hence it would be possible to detect such activity. In an Internet banking system the tolerance for robotic activity is zero, and any sign of such activity would be a strong indication that malicious activity is being performed. One of the easiest parameters to test is the response time to the client, but also other properties such as Cascading Style Sheets (CSS) loading and mouse detection would give good indications.

Summary

The previous sections list different threats that have to be considered when trying to create profiles of normal behavior in order to detect anomalous events. As is seen, there is no universal profile which will find all types of attacks. It is necessary to look for different attacks at different levels of abstraction, from a single request's attribute values to a total view of all system activities within a given time frame. Section 4.1 presents a selection of possible profiles which could be used in order to detect abnormal behavior at different levels.

2.2 State of the art

The area of fraud detection has been in the spotlight for researchers for several decades. The research has led to many discoveries and new technologies, and is still under constant evolution. This section will provide a review of the state of art in fraud and intrusion detection with special emphasis on research that also involves unsupervised machine learning and intrusion detection systems.

2.2.1 Fraud detection

The field of fraud detection is concerned with detection of malicious actions that most likely would result in financial events. The following sections describe fraud detection in general and intrusion detection systems which can be used to detect malicious actions trying to achieve fraud. The methods used for fraud detection also apply for other real world problems, such as computer intrusion detection [MTV00]. This shows that these problems intertwine and research from one field is also valid for the other field.

Within fraud detection there seems to be three main research branches, credit card fraud, insurance fraud and telecommunications fraud. Published research done on fraud detection within the domain of banking applications seems to be limited. This is most likely not due to the absence of research, but rather the privacy, secrecy and commercial interests concerning this specific domain. However, we find that technology and research concerning fraud detection is in general of interest to this project and will therefore be studied. Hence, we find it more important to study which methods have been used on different types of data sets instead of what type of fraud is detected.

Many researchers have looked at the combination of fraud detection and machine learning. There are two approaches to machine learning. Supervised, in which the classification is known in advance, and unsupervised, in which it is not. Supervised approaches, trying to learn classification based

on data from only one class, have been the focus of much research in the last years. The first approaches to this type of fraud detection were published in 1997 by [Kok] and [AFR97]. [Kok] proposed the use of decision trees with Boolean logical functions to profile each legitimate user's behavior in order to detect deviations from the legitimate norm. Then, the norm for each user was used to perform cluster analysis to identify each legitimate user's credit card transactions. [AFR97], on the other hand, experimented with neural networks consisting of the same number of input and output neurons. These networks were used to learn the legal credit card transactions. These two researches proposed looking at single data instances to detect fraud. This was taken a step further by [MP99] who looked at profiles at different levels. They specified profiling schemes for single call, daily, as well as overall levels of normal behavior for telecommunication accounts. The final profiles were extracted by a clustering algorithm that found normal behavior across all accounts. Anomalies were detected by breaches on values for call duration, destination, and quantity.

[Fin03] implements a novel fraud detection method in five steps: First, generate rules randomly using an association rules algorithm and increase diversity by including time. Second, apply rules on known legitimate transaction database and discard any rule which matches this data. Third, use remaining rules to monitor actual system and discard any rule which detects no anomalies. Fourth, replicate any rule which detects anomalies by adding tiny random mutations. Fifth, retain the successful rules. This system has been under testing for internal fraud detection by employees within a retail transaction processing system.

A different branch of fraud detection uses unsupervised approaches together with unclassified data. The early attempts of unsupervised fraud detection were with the use of neural networks. [DC] built an online system for fraud detection of credit card operations based on a neural classifier, without the need for labeled data. To ensure proper model construction, a nonlinear discriminant analysis was used. The system is fully operational and currently handles more than 12 million operations per year with very satisfactory results. [BST01] use a recurrent neural network to form short-term and long-term statistical account behavior profiles. A statistical model is used to compare the two probability distributions, and give a suspicion rating on telecommunications toll tickets. [YTWM04] demonstrated the unsupervised SmartSifter algorithm, which can handle both categorical and continuous variables, and detect statistical outliers on medical insurance data. The SmartSifter algorithm uses a neural network to learn the underlying probabilistic model.

[CCCY05] used a multi-layer neural network with exponential trace mem-

ory to handle temporal dependencies in synthetic Video-on-Demand log data. [ELBJ03] propose fuzzy neural networks on parallel machines to speed up rule production for customer-specific credit card fraud detection. [ZY02] proposes support vector machine (SVM) ensembles with aggregation methods for telecommunications subscription fraud. [KPJ+03] proposes an anomaly detection scheme using a personalized profile for each credit card, based on the transaction information from all transactions. This scheme uses support vector machines in the detection of anomalies and reports satisfactory performance, but suggests negative data collection, i.e. collection of anomalies, to boost performance and detection rates.

2.2.2 Anomaly detection

Little change and progress has occurred in the field of misuse detection systems. It seems that the research has been more focused within the area of anomaly detection. This is most probably because of the potential of a successful anomaly detection scheme. Anomaly detection is divided into two main parts as described in the following section concerning intrusion detection systems. The two parts are host based and network based, divided by where the data is collected from. For this state-of-the-art study, we have chosen to focus on host based anomaly detection systems, with an emphasis on anomaly detection by machine learning.

A huge amount of various researches have been conducted within anomaly detection. The amount is still large when we limit the area to only contain anomaly detection which uses machine learning. The fields studied here include statistical models and unsupervised learning models.

In [Den87] a number of statistical characterization techniques for events, variables, and counters were first outlined. It used parameters, such as the central processing unit (CPU) load and the usage of certain commands, in order to flag anomalous behaviors. Examples of these early statistical models are:

- Threshold measures, or *operational models* [Den87], in which standards, or heuristically-determined limits, are used to flag anomalous rates of event occurrences over an interval, e.g. on the number of failed login attempts.
- Computation of mean and standard deviation of descriptive variables, in order to compute a confidence interval for "abnormality".
- Computation of covariance and correlation among the different components of multivariate measurements on a computer system.

Another interesting approach is the use of an incidence matrix between different commands and users, which is searched for statistical lows, representing rare commands [TS98]. More complex theoretical work, such as [YC01], has also followed this purely statistical approach, with some success. However, some clear limitations have lead to critics to this type of approach, mainly due to the fact that they do not take into account the sequence of events, just atomic events.

On the border line of both statistical based and unsupervised learning lies SVM. Anomaly based systems based on SVM have provided solutions that proves high accuracy on training data [HLV98]. Extensions to SVM has also been presented, which should lead to better detection of anomalies. Such an example is Robust SVMs [LT06] and feature selection SVMs [JY06].

Various unsupervised learning techniques have been used for host based anomaly detection. Some of the more advanced applications of statistics can also be defined as *learning algorithms*. For instance, some uses of Markovian process models can be ranked as such. Among these are clustering techniques to group similar activities or user patterns and detect anomalous behavior [Bac00].

The very first approaches dealt with the analysis of sequences of system calls for system processes. The first mention of the idea is in [FHSL96], where *normal sequences* of system calls are considered, without paying attention to the parameters of each invocation. Variants of [FHSL96] have shown improvements, as proposed in [MZI08] and [CLM01], but the general idea is still the same. Finite state machines have been used to express the language of the system calls of a program using deterministic or nondeterministic states [GSS99]. Hidden Markov models have also been used to model sequences of system calls [OMSH03], with better detection results, but with computational problems [YD02]. Markov chains have also been used to create state machines with probability rates for jumps between states [YC01]. This research has shown satisfactory results with rather simple methods [JW05].

Different kinds of neural networks have also been suggested in [GSS99] and [RLM98]. However, none of these methods analyzes the arguments of neither return values nor system call values. This is due to the inherent complexity of the task, however the arguments contain a wide range of information that can be useful for intrusion detection. For instance, mimicry attacks [WS02] can fool the sequence analysis, but it is much harder to devise ways to cheat both the analysis of sequence and arguments. In [WFP99] a detailed review of different approaches is presented, along with a comparative evaluation on live datasets. The conclusion is that the performance issues increase with the size of the data set, and that simpler models would be preferred when dealing with complete and consistent data sets.

It seems clear in this study that analysis of parameters and sequences is under-represented in the research, and would seem like a logical area to research further.

2.3 Intrusion detection systems

The idea of intrusion detection systems (IDSs) was proposed as early as in 1980 by J.P Anderson [And80]. He defined an intrusion attempt as the potential possibility of a deliberate unauthorized attempt to access information, manipulate information, or render a system unreliable or unusable. After this, the concept of IDSs has undergone much research and development. In general, one can say that an IDS monitors a system and the interaction with its users. Each interaction is checked against some rules or norms in order to verify that it is within the bounds that determines normal interactions. If an interaction breaches one of these bounds, it is considered an abnormal interaction, which should be reported. All abnormal interactions are not necessarily malicious interactions, but it is a strong indication that it could be.

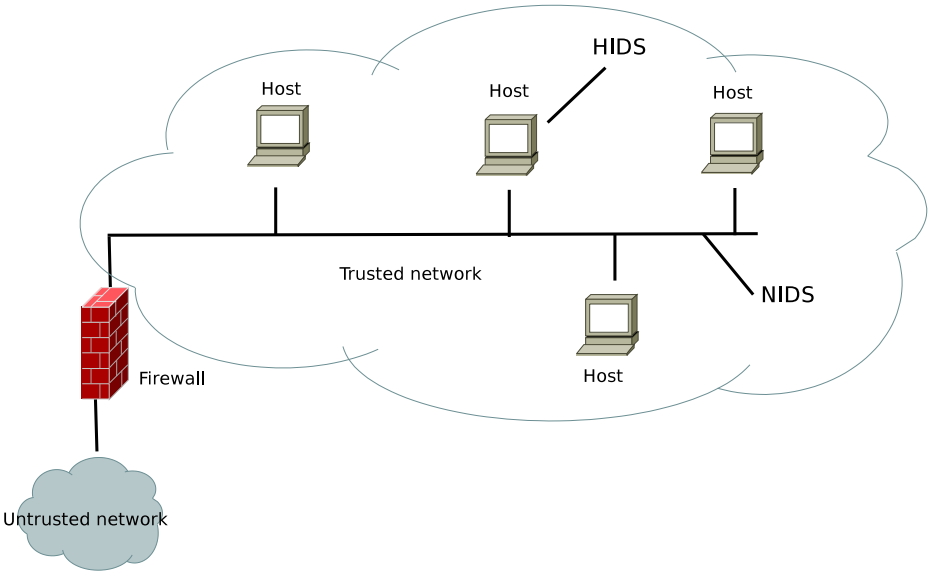

Figure 2.11: Locations of HIDS and NIDS

IDSs differ in many aspects of their functionality. The ones of most interest are location of information collection (host or network) and means of intrusion detection (misuse or anomaly). The following sections try to

describe some key aspects and properties of different IDSs with a final conclusion pertaining to the specifications of the IDS considered in this project.

Location

The difference between host IDSs (HIDSs) and network IDSs (NIDSs) is in where the information originates from. The HIDS looks at information on a single host, and is limited to only the information that can be traced by the operation system on the monitored host. The NIDS is connected to the network of a system and monitors the traffic, trying to detect packages which could be part of an attack. The different locations of the IDS solutions can be seen in figure 2.11. There are several advantages and drawbacks for each of the solutions. Usually, the strength of one is a weakness for the other.

The most prominent strength of NIDSs is that they can be installed and configured with small to no impact on the rest of the system. However a vital shortcoming is that NIDSs are unable to analyze and detect attacks in encrypted traffic. HIDSs, on the other hand, will affect the system they monitor, but provide the key strength that they can analyze encrypted traffic.

Table 2.4 shows a comparison of advantages and drawbacks between HIDS and NIDS. Here, one can see how different aspects of intrusion detection influence the different types. The advantages are or course preferable for both, but it is the drawbacks and limitations that usually tip the scale.

Table 2.4: Comparison of HIDSs and NIDSs

Host	**Network**
Centralized	Distributed
Large impact on system	Small to non impact on system
Needs specialization	General in nature
Possible bottleneck	No influence on system performance
Looks at the activity on a single host	Looks at communication for whole system
Can read encrypted data	Cannot read encrypted traffic
Can detect attack stages and success	Can only detect single package attacks

Detection method

Misuse based detection relies on knowledge of past attacks and the patterns and attributes of these attacks. The IDS uses this knowledge by analyzing

packages against a database of known attack indicators, referred to as *signatures*. This approach has been widely used by anti-virus software to detect malicious applications. The problem with this approach is that the detection engine is only able to detect known attacks, making it vulnerable for unknown and new approaches and methods of attacks.

Anomaly based IDSs look for behavior that deviates from what is defined as normal behavior. The underlying assumption is that malicious behavior deviates from normal behavior. An attractive property of these IDSs is their ability to detect previously unknown attacks.

Such systems operate in one of two phases, a training phase and a detection phase. During the training phase, normal behavior is established through processing and analyzing a data set. This set might be an application's log file, or data passed from an application to the IDS through defined interfaces.

In the detection phase, the data is analyzed to see if the events that occur are normal, in terms of a model created during training. If the new event differs from this model, by some metric measure by more than a defined threshold, the event is considered abnormal, and might represent a malicious action. Several approaches have been made to define normal behavior from the training data, such as decision trees, neural nets, and various statistical models. When deciding the threshold value[2], there is an essential trade-off involved, namely the trade-off between false positives and true positives. Whereas true positives denote abnormal events classified correctly by the IDS, false positives are harmless events mistakenly classified as malicious.

Table 2.5 shows the advantages and drawbacks between anomaly and misuse detection. Usually, an advantage of one detection scheme is a drawback of the other. This makes the separation between them clear and understandable.

Table 2.5: Comparison of anomaly and misuse detection

Anomaly detection	**Misuse detection**
Do not require updates	Require continuous updates
Long and complex training	No initial training
Tuning included in training	Need tuning
Can detect new attacks	Cannot detect new attacks
Vague alerts	Precise alerts
High false positive rate	Low false positive rate
Difficult to design	Easier to design

[2]The threshold defines the numerical border between normal and abnormal values.

Issues with intrusion detection

Generally speaking, there are some recurring issues that seem to trouble all types of IDSs. These are problems not directly concerned with the way intrusions are detected, but rather the IDSs' architecture, integration with existing systems, quality of service, and actions. In the following there is presented a description of six issues that concern IDSs in general, and which needs to be kept in mind when constructing one.

Comprehensiveness of model: This issue is concerned with how often the model is updated against new attacks and behavior. This is relevant for both misuse and anomaly detection, but would concern different data for each. For misuse detection it is important to update the model against new attacks as they are discovered. For anomaly detection the model must be updated whenever user behavior changes due to structural or functional updates to the monitored system. In general, it can be said that it is very important to keep the model as comprehensive as possible.

Zero-day attack recognition: This is clearly an issue concerning misuse detection. IDSs are intended as a complementary security measure which can detect failures of other mechanisms. The limitation of not being able to detect unknown attacks would seem unacceptable. Hence, one can state that a misuse detection scheme would not be able to provide the functionality of an IDS alone. This brings the anomaly detection scheme into focus with its strengths and weaknesses as stated in the previous section. One can ultimately state that in order to provide the proper functionality of an IDS, all the strengths of both misuse and anomaly detection are needed, while limiting the weaknesses.

Intrinsic security and survivability: This is an important issue. The security of the IDS itself also needs to be considered. All security mechanisms are of interest to attackers, since a successful attack on one of them could halt security. Therefore it is important that the IDS does not become a security threat itself. Second, it is important for it to provide as good performance as possible, even during an attack. This is called survivability and is concerned with the consequences of a direct attack against the IDS.

Flexibility and usability issues: This issue is concerned with how personnel interact with the IDS and how the IDS is made to adapt to the monitored system. Clearly, a security mechanism is not any stronger

than its weakest link, and hence, it is always important to provide proper education on the usage of an IDS to the security personnel. The IDS should also be as easy and intuitive as possible, providing a high level of usability. The most important part of this is to make the personnel knowledgeable of the types of alerts and security procedures related to each alert. It is also important to make the IDS flexible in the way in which it adapts to the system. This is a problem mostly concerning misuse detection since attack patterns are usually application specific, while anomaly detection uses the available data to create its models.

Scalability and throughput: In the evolving systems of today it is important to keep performance issues, such as scalability and throughput, in mind when constructions IDSs. It seems inevitable that data quantity and speed are ever increasing, hence it is important to monitor this such that the IDS does not become a performance bottleneck for the system in general.

Reactivity and intrusion prevention: This is probably the most important issue with IDSs since it concerns the goal of their existence. There are several issues regarding how the IDS should react to a detected intrusion. Reactivity deals with what to actually do, which is most often divided between active and passive reaction. The most relevant issues concern the active reactions. Firstly is the threat of denial of service due to false positives in the IDS. If an active IDS falsely detects an intrusion, it could lead to a denial of a legal service request. This would seem inevitable as long as IDSs produce false positives. The second issue is how one would integrate a prevention mechanism. Placing it at a key network point would seem like a good and easy choice, but would leave the system vulnerable for local attacks. Placing it on each host would prevent this, but would lead to a much larger and more complex integration.

As seen in the previous points there are several issues which concerns IDSs in general. Some of these points will most likely always be a part of IDSs, while others might be avoided as IDSs evolve. However, the list provides important issues which need to be kept in mind when designing and implementing an IDS with the technology available today.

2.4 Machine learning methodologies

An understanding of the basic principles of machine learning is needed before concrete methods can be studies. Hence, the following section introduces the field of machine learning. After this introduction we describe the actual machine learning methods used by this project to detect fraud.

2.4.1 General machine learning

Machine learning is the branch of artificial intelligence that examines how machines can mimic the humans' intelligent ability to learn. The term machine learning can in this context be defined as whenever a machine changes its model, data, or relations in such a way that its expected future performance improves, based on received inputs. This can be done either by supervised or unsupervised learning. If the output from the model is in a discrete form, it is called a classification problem. Otherwise, it is a regression problem. The interest lies not in managing to remember the examples used during learning, but using them to predict answers for cases not yet observed. The following sections contain information on the key aspects of learning algorithms used by anomaly IDSs.

Machine learning approaches has a clear division based on the type of data they use, this division is between supervised and unsupervised learning. The division is based on what type of information is available in the data set. If we know the classification associated with each data item, the task would be to create a model that learns this mapping. This is the supervised case. On the other hand, if we have no idea of the classification of items in the data set, a model has to be built that classifies the instances based on properties in the data. This problem is often referred to as *clustering*, and is primarily what unsupervised learning is concerned with.

Training

The goal of supervised learning is to create a function based on the training data. The training data consists of pairs of input objects together with its desired output. The output of the function can be a continuous value or can predict a class label of the input object. In the case of a continuous output value the learned function is called a regression function, and in the discrete case it is called a classification function. The task of the supervised learner is to predict the value of the function for any valid input object after having seen a number of training examples. To achieve this, the learner

has to generalize the presented data such that it can predict new unforeseen situations in a reasonable way.

The unsupervised learning method simply receives inputs, but does not obtain supervised target outputs, nor any other feedback based on its predictions. Since the unsupervised approach does not have the same data material as the supervised, it is clear that it does not have the foundation to make the same types of predictions. However, it is possible to develop a formal framework for unsupervised learning based on the notion that the machine's goal is to build representations of the input that can be used for decision making, predicting future inputs, efficiently communicating the inputs to another machine, etc. In a sense, unsupervised learning can be thought of as finding patterns in the data above and beyond what would be considered pure unstructured noise.

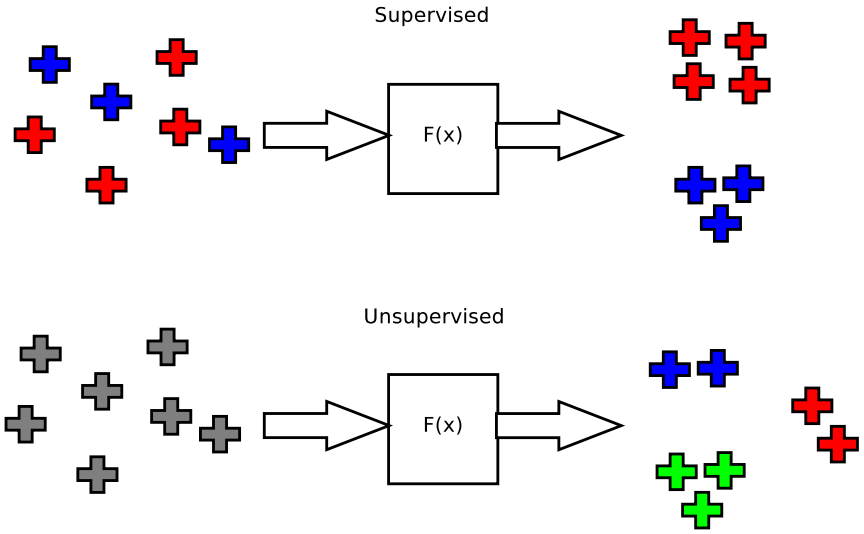

Figure 2.12: Supervised and unsupervised machine learning

Figure 2.12 visualizes the difference between supervised and unsupervised learning. For the supervised case, there exists a predefined classification of the input instances, and this classification is learned by the function $F(x)$. For the unsupervised case, no predefined information about classes or classification is available for the input instances, and the function $F(x)$ learns both the number of classes and classification of instances into these classes.

Measurements

The performance of different machine learning methodologies can be measured by many different means, all of which have different usage. In this

project we have decided to use the mapping between actual values and predicted outcome as a measurement basis. Based on this measurement we are also able to use the receiver operation characteristics (ROC) curve to evaluate the performance of our models [WBB08]. First, we describe the general theory behind actual and predicted values, before the ROC curve is explained.

Figure 2.13 shows a matrix that describes the relationship between the predicted outcome and the actual value. This is a simple one-class case where an instance is either classified, p', or not classified, n', and the actual value is either within the class, p, or outside, n. For each instance that is classified by a model, the outcome will lie within one of the four categories shown in the figure. In order to use this figure, knowledge about the actual value of the instances is needed, thus this method would seem inapplicable for unsupervised learning methods. However, this can be bypassed by evaluating the model on fabricated or modified data that contains classification labels.

		Actual value	
		p	**n**
Predicted outcome	**p′**	True positive	False positive
	n′	False negative	True negative

Figure 2.13: Mapping between actual values and predicted outcome

First, we look at the types of outcomes a learning algorithm can produce [All01]. Figure 2.13 shows that there are two correct outcomes and two false outcomes. First, the algorithm can classify an instance that is actually part of the class, a *true positive*. Second, the algorithm cannot classify an instance which is not part of the class, and hence produce a *true negative*. Then there are two types of false outcome that could be experienced. The first is when the algorithm classifies an instance which is not actually part of the class. This is called a *false positive*. In an anomaly detection scheme this would result in an anomaly instance being classified as normal. The second type of false outcome is when an instance, which actually is part of the class, is not classified by the algorithm. This is referred to as a *false negative*. This would be a normal instance being classified as an abnormal one. Clearly, models that have a high rate of *true positives* and *true negatives* are desirable, and as small rate as possible of *false positives* and *false negatives*. It is also important that one of the erroneous outcomes are preferred, compared

to the other. Clearly, it is better to classify a normal instance as malicious, then to classify a malicious instance as normal.

The ROC-curve is a graphical representation of the sensibility of a classifier as the detection threshold varies. The ROC can be represented by a graph, plotting the number of true positives and false positives for different levels of detection rate. The usual characteristics for a learning algorithm is that in order to gain a true positive rate close to 100 percent, the detection rates in general have to be increased, which leads to a higher false positive rate as well. The purpose of a ROC curve is hence twofold. First, it gives a visualization of the performance of different learning algorithms, and secondly, it gives a visualization of the trade-off between true positives and false positives.

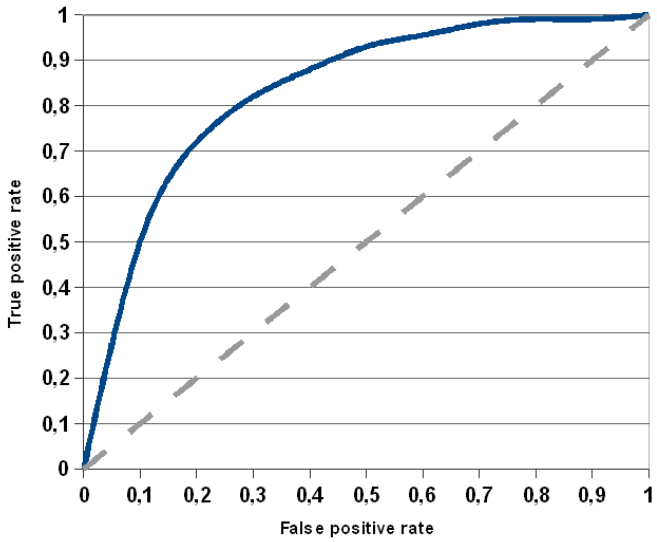

Figure 2.14: ROC-curve

Figure 2.14 shows an example of a ROC-curve. Here, the true positive rate is mapped along the y axis, and the false positive rate along the x axis. The different approaches experience varying levels of true positive rate as the false positive rate increases. Obviously, one would want the graph to give an as high true positive rate compared to the false positive rate as possible. In this example, there is however no obvious choice.

Feature selection

Feature selection is the technique, commonly used in machine learning, for selecting a subset of relevant features when building learning models [ZS04].

By removing the most irrelevant and redundant features from the data, feature selection can help to improve the performance of learning models by:

- Alleviating the complexity of increasing dimensionality.
- Enhancing generalization capability.
- Speeding up learning process.
- Improving model interpretability.

Feature selection is an important step for any learning application. The importance of correctly choosing features for machine learning problems has been widely discussed in the literature [GE03].

Feature subset selection (FSS) for multivariate time series is a well studied process. Unsupervised FSS techniques usually compute the similarity between features and remove redundancies in order to reduce the number of features. Usually, this is accomplished through partitioning or clustering the original feature set into partitions, each of which will be represented by a single representative feature to form the reduced subset.

However, no reliable method exists which takes into account categorical variables. Therefore, we resorted to a much simpler approach, testing different combinations of the variables. The provenance of the data used by this project would provide us with enough information to use common sense to remove redundant and useless features.

One class classification

The problem in one class classification is to make a description of a target set of objects and to detect which new objects resemble this set. The difference compared to conventional classification is that in one class classification, only examples of one class are available [MY01]. The objects from this class are called *target objects*. All other objects are per definition *anomaly objects*.

Obviously, the first application for one class classification is anomaly detection, detecting uncharacteristic objects from a dataset, examples which do not resemble the bulk of the dataset in some way [MY00]. In general, trained classifiers only provide reliable estimates for input objects resembling the training set. Estimates for unknown and remote regions in the feature space are very uncertain, hence we conclude in that target objects are classified correctly and anomaly objects are not classified as target objects [RP96].

Secondly, one class classification can be used for classification problems where one of the classes is sampled very well, while the other class is undersampled. For instance, possible false values from a machine would be such an

under-sampled set of values. Measurements of the normal working conditions of a machine are well represented and easy to obtain. On the other hand, measurements of false values would require the destruction of the machine in all possible ways. It is very expensive, if not impossible, to generate all faulty situations [Jap99]. Only a method trained on just the target data can solve the problem of monitoring the values of such a machine.

Figure 2.15 shows a general example of the one class classification problem. We see that we have a clear cluster of target objects located in the middle of the graph. These target objects have been classified by a boundary shown by the black line surrounding the objects. Outside this boundary there are some anomaly objects, which clearly fall outside the defined boundary, and hence are not classified as target objects.

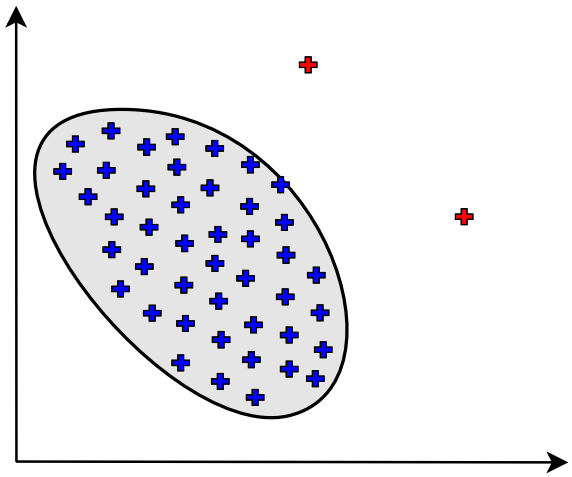

Figure 2.15: One class classification problem

2.4.2 Support vector machines

Support vector machines (SVMs) are a technique used for data classification. It originates from the work done on statistical learning theory by Vapnik and Chervonenkis, published 1974 [NC74], and was first proposed at a conference by Vapnik in 1992 [BGV92]. SVMs try to obtain the optimal boundary between two classes within a vector space, independently of the distribution of training vectors of each class. The foundation of this technique is quite simple: locate the boundary with is the most distant from the vectors closest to the boundary of both classes. The basic idea was for the boundary to be linear, but new research has introduced *kernel functions*, which can locate non-linear boundaries as well [CWHL]. The following sections describe

the basic theory behind the original SVM, followed by a description of the extension of kernel methods.

The simplest form of SVM is a linear SVM used on data which is actually separable [Bur98]. Figure 2.16 gives an idea of the workings of a SVM. In the figure, there is shown two types of data, blue and red dots, which can be separated by a linear border. The figure also shows three different border proposals. H_3 is clearly a wrong proposal, since it is unable to separate the two data types. On the other hand, H_1 and H_2 manage to separate the data. SVMs try to find the separation that provides the largest possible distance between the border and the data item closest to the border. Here, the distance between the border and the item closest to it is small for H_1, but maximized for H_2. So the result of a SVM classification on the example shown in the figure would be H_2.

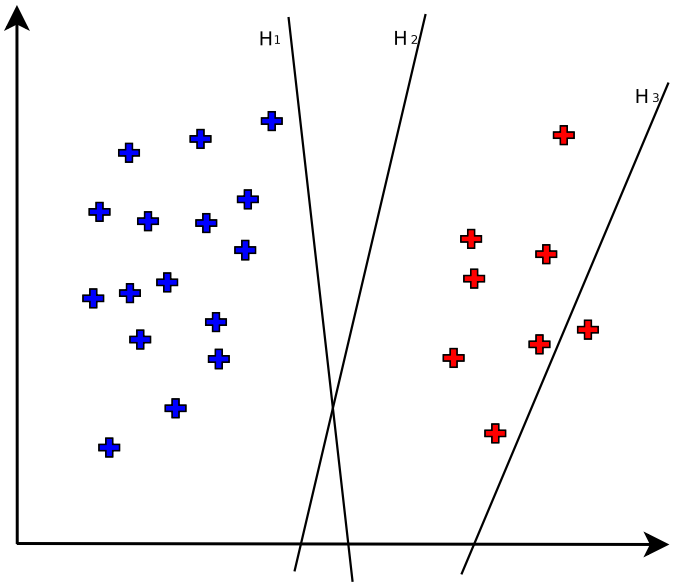

Figure 2.16: SVM two class classification problem

The following is a more formalized description of the theories behind SVM. Suppose we have a set of training data, consisting of n data vectors, $\vec{v}_1, \vec{v}_2, \ldots, \vec{v}_n$, and their associated truth values, $y_1, y_2, \ldots, y_n$:

$$(\vec{v}_1, y_1), (\vec{v}_2, y_2), \ldots, (\vec{v}_n, y_n)$$

The vector $\vec{v}$ consists of the data point values in a p-dimensional space, while the truth value y denotes the class of the data points. Now we assume that the training set contains a sufficient distribution such that the bound-

aries of the classes could be learned by a separating hyperplane[3] We want to find a classification function, $f(\vec{v})$, which takes as its input a vector, and correctly assigns a truth value to it as shown in equation 2.1.

$$y = f(\vec{v}) \tag{2.1}$$

The performance of the classifier is measured by a simple error function shown in equation 2.2.

$$E(y, f(\vec{v})) = \begin{cases} 0 & y = f(\vec{v}) \\ 1 & otherwise. \end{cases} \tag{2.2}$$

Further, we suppose that there are several classification functions that solve the task, and that these can be written as a function, f, with configuration parameter, p, denoted $f(\vec{v}, p)$. Using this, a risk estimate, $R(p)$, of each configuration can be defined, as shown in equation 2.3.

$$R(p) = \frac{1}{n} E(y, f(\vec{v}, p)) \tag{2.3}$$

These equations form the basic foundation for the work on SVM described in the following.

Basic linear support vector machine

Next we assume a two class data set which can be separated by oriented hyperplanes. The classification function from the previous section would then take the form of a hyperplane, as shown in equation 2.4

$$f(\vec{v}, p) = \vec{w} \cdot \vec{v} + b = 0 \tag{2.4}$$

The normal vector, $\vec{w}$, points orthographically[4] to the hyperplane separating the two classes and b is the bias term. The orthographical distance of the hyperplane to the origin is calculated by the offset parameter, b, divided by the length of $\vec{w}$. The hyperplane can be seen in figure 2.17, illustrating the normal vector, $\vec{w}$, normalized onto the hyperplane, and the bias parameter, b, showing the offset from the axes.

[3] A hyperplane is an $(n-1)$-dimensional subspace of an n-dimensional vector space.

[4] Orthographic projection is a projection which is made by drawing lines from every point to be projected, perpendicular to the plane of projection.

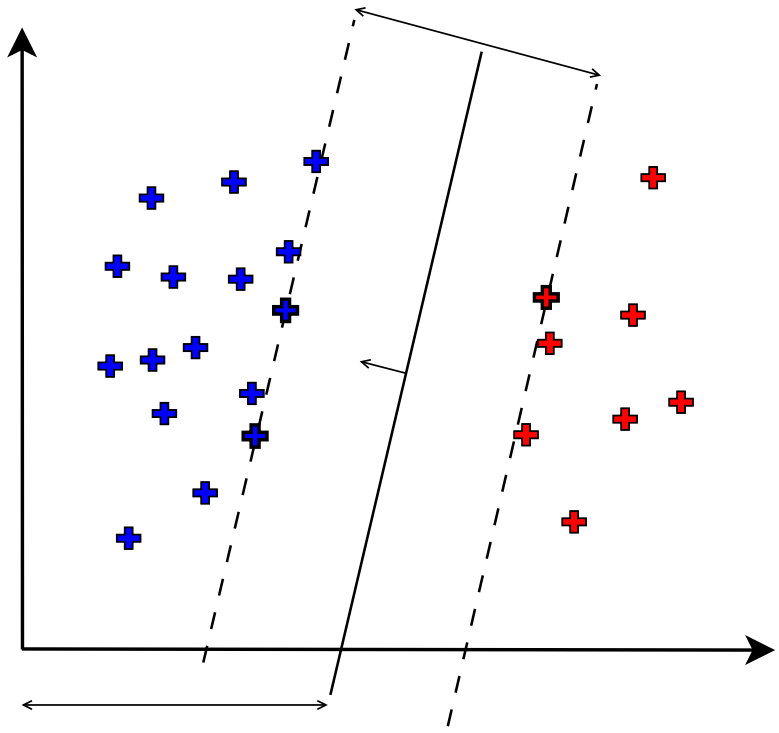

Figure 2.17: Support vectors creating the classification function

As mentioned before, the basic idea is to find a good separation by finding the boundary with the largest distance from the neighbors of both classes. This is because one wants to find a boundary with the best possible separation between the two classes. This follows logically from the risk estimate, where a large margin would result in a low risk. This solution is called the maximum margin solution and is described as the mean of two parallel hyperplanes, each passing through at least one vector of its representative class. These vectors are called support vectors and are described by equations 2.5 and 2.6.

$$\vec{w} \cdot \vec{v} - b = 1 \tag{2.5}$$

$$\vec{w} \cdot \vec{v} - b = -1 \tag{2.6}$$

The values 1 and -1 are the usual representations of the truth value, y, where 1 equals one class and -1 equals the other class. The two hyperplanes are selected by two criteria: there are no data points between the hyperplanes, and the distance $\frac{2}{|\vec{w}|}$ is maximized. This can be put together further to obtain equation 2.7.

$$y_i(\vec{v}_i \cdot \vec{w} + b) - 1 \leq 0 \qquad \forall i \tag{2.7}$$

This maximization problem should be optimized to find the largest possible distance between the support vectors, $\frac{2}{|\vec{w}|}$, hence minimizing the value of $|\vec{w}|$. This distance can be seen in figure 2.17, as the distance between the two support vectors. The optimization problem is solved by using a Lagrange formulation[5]. The mathematical foundation and description of such a transformation is outside the scope of this project, interested readers are recommended to read [SPST+01] for a thorough review. After the transformation of the Lagrange formulation, the problem looks as in equation 2.8.

$$Max \sum p - \frac{1}{2} \sum p_i p_j y_i y_j (\vec{v}_i \cdot \vec{v}_j) \qquad \forall i, p_i > 0, \sum p_i y_i = 0 \tag{2.8}$$

Solving equation 2.8 provides values for all p, b is found using equation 2.7 and $\vec{w}$ is calculated using equation 2.9.

$$\vec{w} = \sum p_i y_i \vec{v}_i \tag{2.9}$$

[5]The Lagrange formulation provides a way to solve functions aiming to maximize or minimize a given cost function. It is widely used to solve optimization problems.

Non-linear SVM using the kernel method

So far, the SVM classifier can only have a linear hyperplane as its decision surface. This formulation can be further extended to build a non-linear SVM. The motivation for this extension is that a SVM with non-linear decision surface can only classify non-linearly separable data. This is done by transforming data into a higher dimension, where it is separable, by a transformation function, $\vec{v} \rightarrow \Phi(\vec{v})$. This is the basis for the kernel method which aims to transform the data to a higher dimensional space, where it is linearly separable. Figure 2.18 shows an example of how data can be transformed into a higher dimensional space.

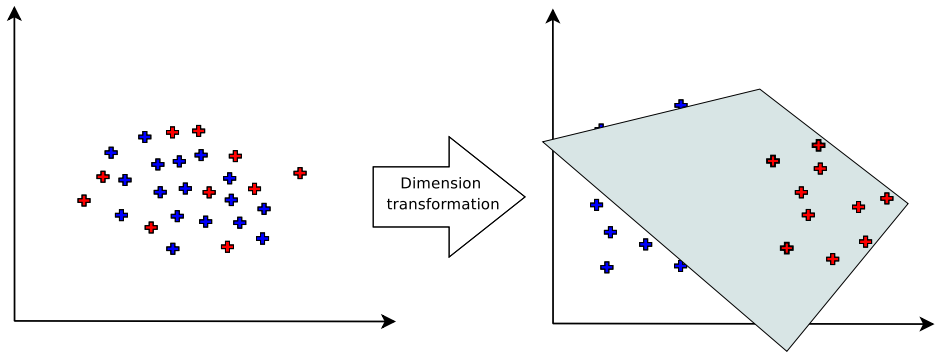

Figure 2.18: Illustration of how higher dimension transformation eases the classification task

Let F be a function we wish to transform to a higher dimensional space. The kernel function, $K(\vec{v_i}, \vec{v_j})$, is introduced and combined with equation 2.4. The kernel function always satisfies the condition shown in equation 2.10.

$$K(\vec{v_i}, \vec{v_j}) = \Phi(\vec{v_i}) \cdot \Phi(\vec{v_j}) \tag{2.10}$$

The above equation indicates that the kernel function is equivalent to the distance between $\vec{v_i}$ and $\vec{v_j}$ measured in the higher dimensional space. If we measure the margin by the kernel function and perform the optimization, a non-linear boundary is obtained [Gun98]. The boundary in the transformed space is obtained by equation 2.11.

$$\vec{w}\Phi(\vec{v}) + b = 0 \tag{2.11}$$

The kernel function can have several different forms with different properties. This project uses the Gaussian Radial Basis Function (RBF) kernel, which has the form shown in equation 2.12. γ is a variable that defines the boundary in the transformed space.

$$K(\vec{v_i}, \vec{v_j}) = exp(\frac{-|\vec{v_i} - \vec{v_j}|^2}{2}\gamma^2) \tag{2.12}$$

One class SVM

One class SVM is a special case of SVM classification. It originates from the theoretical work done by Scholkopf and Smola [SSWB00] and [SPST+01]. Supposing there is a dataset, drawn from an underlying probability distribution, P, an estimation of a simple subset, S, of the input space is needed, such that the probability of a test point from P lying outside S is bounded by some a priori specified $\nu \in (0, 1)$. The solution for this problem is obtained by estimating a function, f, which is positive on S and negative on $\neg S$. In other words, Scholkopf et al. developed an algorithm which returns a function f that takes the value $+1$ in a small region, capturing most of the data vectors, and -1 elsewhere. A basic example of one class SVM classification with the use of a RBF kernel can be seen in figure 2.19.

To separate the training set from the origin, one needs to solve the quadratic equation described in equation 2.13 [GD03] and [HSKS].

$$\min_{\vec{w} \in F, \xi \in \mathbb{R}^l, p \in \mathbb{R}} \frac{1}{2}|\vec{w}|^2 + \frac{1}{\nu l}\sum_{i=1} \xi_i - p, (\vec{w} \cdot \Phi(\vec{v_i})) \geq p - \xi_i \qquad \xi_i \geq 0 \tag{2.13}$$

ν is an upper bound on the fraction of anomalies, that is, training points outside the estimated region. l denote l-dimensional vectors whose components are labeled using a normal typeface. ξ_i describes the slack from the item $\vec{v_i}$ to the detection threshold. If $\vec{w}$ and p solve this equation, then the decision function shown in equation 2.15 will be positive for most $\vec{v}$ contained

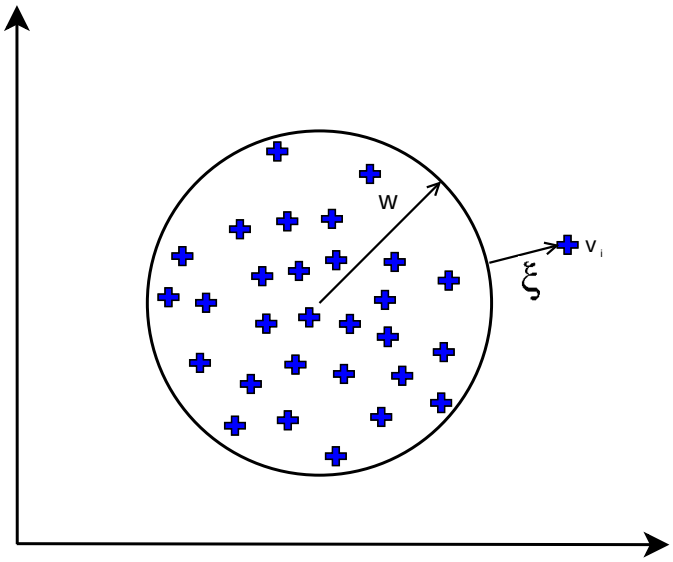

Figure 2.19: Example of one class classification problem solved using RBF kernel

in the training set[6].

$$f(\vec{v}) = sgn(\vec{w}\Phi(\vec{v}) - p) \tag{2.15}$$

2.4.3 Artificial neural networks

An artificial neural network (ANN) is a computational model based on the biological neural networks. They can be understood as weighted directed graphs, consisting of *neurons* (nodes), *synapses* (edges), and a set of weights. When presented with input, they produce some output value. During training, the output value produced by a given input can be influenced.

An ANN can be organized in different ways. As with other machine learning methodologies, they can be both supervised, and unsupervised. Since we

[6]The sgn-function returns an integer indicating the sign of the number as follows:

$$sgn(x) = \begin{cases} x < 0 & = -1 \\ x = 0 & = 0 \\ x > 0 & = 1 \end{cases} \tag{2.14}$$

do not have the material to employ supervised learning, the ANNs considered here are unsupervised Unsupervised neural networks are *two-layered*, which means that they only consist of a set of input neurons and a set of output neurons. The input neurons are connected to the output neurons by synapses. Each synapse is directed from the input neuron to the output neuron, and has a weight associated to it. As such, each output neuron can be said to have a *weight vector*, and the network as a whole a *weight matrix*. An example two-layered neural net, with three input nodes and four output nodes, is illustrated in figure 2.20. Its corresponding weight matrix is shown in table 2.6. If we consider the rows, we can say that output node O_1 has weight vector $\vec{W_1} = [w_1, w_2, w_3]$, O_2 has weight vector $\vec{W_2} = [w_4, w_5, w_6]$, and so on.

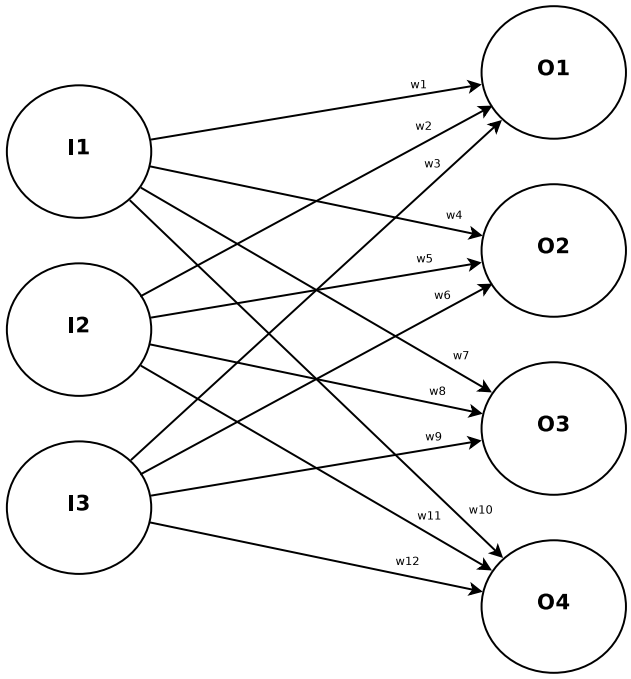

Figure 2.20: Two-layered neural network

The input to such an ANN is a vector, i.e. one value for each input neuron. Based on the weights of the synapses, one output neuron is calculated to be the winner, also called the best matching unit (BMU). This is the output neuron with a weight vector closest to the input vector. The index of the winning neuron is the output from an unsupervised ANN.

Before ANNs can produce meaningful output, they must be trained. The way this is performed varies between different types. In the case of unsupervised networks, it involves feeding data from a training set to the network,

Table 2.6: Weight matrix for a neural net

	$\mathbf{I_1}$	$\mathbf{I_2}$	$\mathbf{I_3}$
$\mathbf{O_1}$	w_1	w_2	w_3
$\mathbf{O_2}$	w_4	w_5	w_6
$\mathbf{O_3}$	w_7	w_8	w_9
$\mathbf{O_4}$	w_{10}	w_{11}	w_{12}

and adjusting the weight vectors. When the network is trained, the weight vectors represent a set of reference vectors with regards to the particular data set.

The trained network is then used to classify input vectors as belonging to one of the output neurons.

Learning vector quantizer

Learning vector quantizer (LVQ) [KHK+96] is one of the most frequently used unsupervised clustering algorithms. There exist several versions of LVQ. We consider the unsupervised version, LVQ-I.

LVQ-I aims to construct clusters of similar input vectors. Similarity is often measured in terms of the Euclidean distance. Figure 2.21 provides a scatterplot visualization of an example output from a LVQ-I with two input nodes and three output nodes. In the figure, the input values to the input nodes are mapped along the horizontal and vertical axes. In that way, the black squares represent input vectors, and the circles represent the weight vectors of the output neurons. The blue line starts at the last input vector and ends at the BMU for that vector.

In our context, this can be used to construct clusters of normal values. There can be several clusters of normal values, each defining a target cluster. Any input not within some threshold from the learned clusters are anomalies.

Training

The training phase of a LVQ-I consists of six steps:

1. Initialize weights
2. Feed the network with a vector from the training set
3. Find the BMU
4. Find the radius of the neighborhood of the BMU

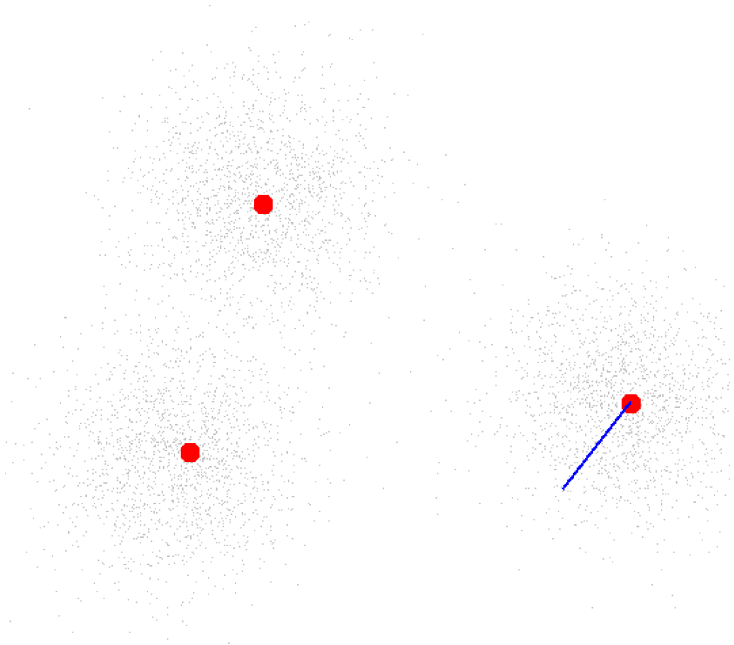

Figure 2.21: Output from a LVQ-I

5. Adjust the weight of each node in the neighborhood.

6. Repeat the procedure from step 2

These steps will be described next. The training algorithm for LVQ-I does not specify the functions that should be used. The functions in the following discussion are from [13], [KHK+96] and [Eng07].

1. Initialize weights Before training begins the weights of the connections between the neurons are initialized. This is done either by setting the weights equal to the first input vector, by randomizing the weights, or setting the weights to a predefined value. The outcome of the training phase is often influenced by this choice, so it may be a good idea to try several methods.

2. Feed the network with a vector from the training set During training, an input vector is chosen from the training set and presented to the network. The vectors can be picked randomly, sequentially, or by other methods. To make the trained model more general, randomly choosing the input vectors may be a good idea.

3. Find the BMU Based on the current weights on the connections, find the best matching unit, i.e. the neuron with the weight vector that is

most similar to the input vector. The measure of similarity may be any function, but the Euclidean distance is often employed. For an input vector, $\vec{V} = [v_1, v_2, \ldots, v_n]$, and a weight vector, $\vec{W} = [w_1, w_2, \ldots, w_n]$, the Euclidean distance, d_E, is calculated as in equation 2.16.

$$d_E(\vec{V}, \vec{W}) = \sqrt{\sum_{i=1}^{n}(v_i - w_i)^2} \tag{2.16}$$

4. **Find the radius of the neighborhood of the BMU** In addition to updating the weight vector of the BMU, the weight vectors of the neurons in the neighborhood of the BMU is often altered, although this is not specified in [KHK+96]. The function used to calculate the neighboring neurons may be chosen freely. It is often preferable to use a decaying function. In that instance, when training starts, all neurons may be in the neighborhood of every other neuron. As training progresses, the neighborhood is decreased, so that in the end, the neighborhood is only a small subset of the neurons. The radius, σ, as a function of time, t, of the neighborhood of the BMU can be calculated by the exponential decay function shown in equation 2.17 [13].

$$\sigma(t) = \sigma_0 \epsilon^{\left(-\frac{t}{\lambda}\right)} \qquad t = 1, 2, 3, \ldots \tag{2.17}$$

σ_0 denotes the radius of the neighborhood at time t_0, i.e. when training starts, and the time, t, is the current iteration of the loop. λ is dependent on σ_0 and the number of iterations chosen for the algorithm to run, N, as shown by equation 2.18.

$$\lambda = N / \log \sigma_0 \tag{2.18}$$

When the radius of the neighborhood is determined, the neurons which lie within can be found by Pythagoras' equation.

5. **Adjust the weight of each node in the neighborhood** The neurons within the neighborhood of the BMU gets their weight vectors updated proportionally to their distance from the BMU. A suitable

function to calculate the weight vector, $\vec{W}$, for time $t+1$, given the weight vector at time t, $\vec{W}(t)$, may be as in equation 2.19.

$$\vec{W}(t+1) = \vec{W}(t) + \Theta(t)L(t)(\vec{V}(t) - \vec{W}(t)) \qquad t = 1, 2, 3, \ldots \quad (2.19)$$

in which t represents the time step, $\vec{V}$ is the input vector, and L is the learning rate.

The purpose of Θ is to make the adjustments of the weight vectors proportional to the distances of the neuron from the BMU. A possible function for Θ is described in equation 2.20.

$$\Theta(t) = \epsilon^{-\frac{d^2}{2\sigma^2(t)}} \qquad t = 1, 2, 3, \ldots \quad (2.20)$$

in which d is a node's distance from the BMU.

Also, it is usually beneficial that the learning rate, L, decreases as a function of the time, e.g. by equation 2.21.

$$L(t) = L_0\epsilon^{-\frac{t}{\lambda}} \qquad t = 1, 2, 3, \ldots \quad (2.21)$$

6. Repeat the procedure from step 2 At this step, one of three choices are made. If the network is not yet sufficiently trained, the process is repeated from step two. If it is, the training phase can be said to have completed a *cycle*. If more cycles are to be performed, training starts over from step one. If not, the training is completed. Several criteria can be used to determine if a training cycle is finished. Training may be terminated when a certain number of vectors are given to the network, when the adjustments to the weight vectors for each input vector are below some threshold, or when the quantization error is small enough. The quantization error, Q, is defined in equation 2.22 [Eng07].

$$Q(t) = \frac{\sum_{i=1}^{t} ||\vec{V}(i) - \vec{W}(t)||_2^2}{t} \quad (2.22)$$

in which, t again is the current time step, $\vec{V}(i)$ is the input vector at time i and $\vec{W}(t)$ is the weight vector of the BMU at time t.

2.4.4 Markov chains

A Markov chain is a statistical model first applied by Andrey Markov in 1906 [14]. Markov chains model processes inhibiting the *Markov property*, which indicates that given the present state, future states are independent of past states.

A Markov chain is described by a set of states and probabilities for changing from one state to another. The changes of state are called *transitions*, and the probabilities associated with the various transitions are referred to as *transition probabilities*. It may be described by a weighted, directed graph, in which the nodes and weights respectively represent the states and transition probabilities. Commonly, the transition probabilities are represented in a matrix, referred to as a *transition matrix*. A simple Markov chain is illustrated in figure 2.22. The arrows are labeled with their respective transition probability. If in state 1, the probability for moving to state 2 in the next transition is 1. When in state 2, the probability for moving to state 3 is also 1. However, when in state 3, there are two possible transitions. The probability for moving to the states 1 and 2 are $\frac{1}{3}$ and $\frac{2}{3}$, respectively. Its corresponding transition matrix can be found in table 2.7. The rows represent the current state and the transition probability for moving to the other states can be found in the corresponding column.

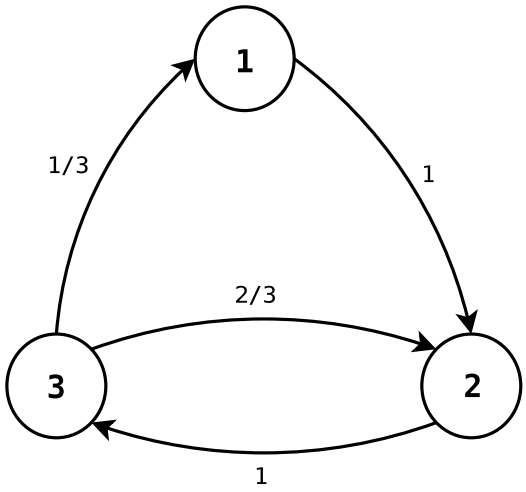

Figure 2.22: Markov chain

Mathematically, for a stochastic process, $X(t)$, the Markov property is

Table 2.7: Transition matrix

	1	**2**	**3**
1	0	1	0
2	0	0	1
3	1/3	2/3	0

described in equation 2.23.

$$P(X(t_{n+1}) = x_{n+1}|X(t_n) = x_n, \ldots, X(t_1) = x_1) = \\ P(X(t_{n+1}) = x_{n+1}|X(t_n) = x_n) \tag{2.23}$$

for a given sequence of time points $t_1 < t_2 \cdots < t_n < t_{n+1}$ [15].

A stochastic process inhibiting the Markov property is called a Markov process, and a Markov process in which the state space is discrete is often referred to as a *Markov chain*.

Many systems can be modeled by Markov chains, even if they initially do not appear to exhibit the Markov property. Even if a process' dependence on previous states does not satisfy the Markov property, it may depend only on a finite *memory* of past states. By considering *state spaces*, instead of single states, the process can be considered Markovian [MT93]. For a process which depends on its last m states, equation 2.23 becomes equation 2.24.

$$P(X(t_{n+1}) = x_{n+1}|X(t_n) = x_n, \ldots, X(t_1) = x_1) = \\ P(X(t_{n+1}) = x_{n+1}|X(t_n) = x_n, \ldots, X(t_{n-(m-1)}) = x_{n-(m-1)}) \tag{2.24}$$

A Markov chain of order m, thus, considers the m past states in the probability for future states. If $m = 1$, it is a regular Markov chain.

Markov chains are widely used within pattern recognition and time series analysis. In our context, they can be used to monitor user behavior. By building a transition matrix from the various requests issued by users, abnormal request sequences can be discovered and flagged as abnormal.

Chapter 3

Method

Summary

This chapter provides a foundation for further work of this project. First, we elaborate on the main research questions in order to decompose them into smaller and more manageable sub questions. The later work will be guided by the need to answer these questions. The contents of these research questions are united into a hypothesis, which will be tested based on the answers of the research questions and the results of this project. After this, we describe the remaining operations that need to be carried out in order to finalize this project and furnish the desired results. Finally, we conduct a thorough analysis of threats to the validity of the results to this project.

3.1 Research questions

In the following we will describe the research questions we wish to test in this project. A research question is a statement that identifies the phenomenon to be studied. We have chosen to list our research questions here as it provides a way to identify the areas we wish to research. It also groups the work that has to be done into main topics, with a decomposition of some of them into sub-questions. This provides a better structure and overview of the problem area, and makes the structuring of the research easier. The research questions are numerated for the sake of reference, not in order of importance. However, sub-questions are considered less important since they are divisions of the main question.

RQ 1: Does the information contained within audit logs contain enough information about normal behavior to create a profile defining all types of normal behavior?

- **RQ 1.1:** Does the audit logs collect enough information about the user's interaction with the system to enable detection of fraud?
- **RQ 1.2:** Is it possible to separate normal and abnormal behavior in audit logs based on SVMs?
- **RQ 1.3:** Is it possible to separate normal and abnormal behavior in audit logs based on ANNs?
- **RQ 1.4:** Is it possible to separate normal and abnormal behavior in audit logs based on Markov chains?

RQ 2: Is it possible to detect malicious behavior and fraud based on audit records from different levels in the system?

- **RQ 2.1:** Is it possible to detect malicious behavior and fraud based on transaction logs?
  - **RQ 2.1.1:** Can research question 2.1 be answered by the use of SVMs?
  - **RQ 2.1.2:** Can research question 2.1 be answered by the use of ANNs?
  - **RQ 2.1.3:** Can research question 2.1 be answered by the use of Markov Chains?
- **RQ 2.2:** Is it possible to detect malicious behavior and fraud based on application logs?

RQ 2.2.1: Can research question 2.2 be answered by the use of SVMs?

RQ 2.2.2: Can research question 2.2 be answered by the use of ANNs?

RQ 2.2.3: Can research question 2.2 be answered by the use of Markov Chain?

RQ 2.3: Is it possible to detect malicious behavior and fraud based on web logs?

RQ 2.3.1: Can research question 2.3 be answered by the use of SVMs?

RQ 2.3.2: Can research question 2.3 be answered by the use of ANNs?

RQ 2.3.3: Can research question 2.3 be answered by the use of Markov Chain?

RQ 2.4: Is it possible to detect malicious behavior and fraud based on information from a combination of logs?

RQ 2.4.1: Can research question 2.4 be answered by the user of Markov chains?

RQ 3: Will malicious activity and fraud always be classified as abnormal data?

RQ 3.1: Will research question 3 be true for SVMs?

RQ 3.2: Will research question 3 be true for ANNs?

RQ 3.3: Will research question 3 be true for Markov chains?

RQ 4: Can malicious behavior be detected by the use of profiles?

RQ 4.1: Can malicious behavior be detected by the use of a profile looking at each user request?

RQ 4.2: Can malicious behavior be detected by the use of a profile looking at sequences of requests?

It is a goal of this project that each of these research questions be answered in a thorough manner. The questions should be answered in light of the results and analysis of the part of the project that is most related to each question. By subdividing each main research question into sub-questions, we are able to provide a more thorough and detailed answer.

3.1.1 Hypothesis

The hypothesis is a statement. It is a prediction or proposed solution to a problem based on prior knowledge or information gathered [Hor89]. It is an educated guess about the outcome of the experiment. A hypothesis must be able to be tested. Here, we propose a hypothesis we wish to answer with this report. The work done by answering the research questions will all contribute to answering and verifying the hypothesis. The hypothesis is as follows:

> *If it is possible to detect novel fraud and misuse by using anomaly detection on audit data from Internet banking systems, then it is possible to do so with the use of machine learning methods on audit data.*

The goal of proposing this hypothesis is to provide a statement to be tested by this research, so it can be either verified or falsified. The results of this project are the answer to the tests of this hypothesis and the research questions.

3.2 Operations

This project has been carried out in different stages, as described in the following list. There has not been a linear transition between the different stages, as some of them have been executed simultaneously, while others have been under progress the whole project. They are not necessarily separated in time, but they are separated in domain and scope. The instruments used by this project in order to carry out these stages are described in appendix B.

Technology research is the first thing that needs to be conducted. This consists of arriving at an understanding of the technological field in which we wish to work. Basically, it consists of a study of the field in general and an analysis of the state of the art of the current solutions to problem areas lying close to the problems we wish to address in this project.

System research needs to be conducted in order to gain knowledge of the foundation of the system. This stage will consist of a thorough examination of the system with a special emphasis on the available data sources. When trying to gain knowledge of the normal behavior of a system it is important to understand the general structure and functionality. While conducting the system research it is also important

to collect as much information as possible, which may be useful in the later stages. Quantity of information is more important than quality at this stage of the project.

Available data research is a finer review and refinement of the data collected in the previous stages. Here, we seek to find the data sources we wish to examine for fraud with a choice of detection technologies. A special emphasis is put on analysis and understanding of all data fields available from the data sources. This understanding is crucial in order to make qualified choices of which data material to use.

Development of hypothesis and research questions can be done once a proper foundation on the available information has been laid. The formulation of the hypothesis is important in order to state a goal for what we wish to test with this project. The research questions will aid us in accomplishing this by dividing objectives into smaller questions, which all need to be answered in order to fully test our hypothesis.

Framework development is the first work done on the system we wish to develop. The theoretic foundation determined earlier is used, in conjunction with the hypothesis, as a basis for the desired functionality. Then, all information on the data sources is taken into account, and an advantageous way of combining this with the selected methods is found, to achieve the wanted results.

Log parsing is the first step that needs to be conducted by an implemented solution. The log data needs to be parsed into a consistent and lucid format. Mechanisms are needed which can easily extract the desired information, and provide an interface that allows the rest of our solution to function smoothly.

Data processing and production would be the next stage, when the data is available through the interface of the log parser. Data processing consists of transforming data to desired formats and, in some cases, normalizing it to get a better effect of the available data. Data production also includes generation of additional data attributes. This is true in particular for string values that cannot be used by machine learning methods.

Model development and implementation can commence once the data is ready for the machine learning algorithm. It is important to note that different machine learning algorithms take different inputs, and this has to be synchronized with data processing and production. This stage

consists of implementing the machine learning algorithm and making it work with the data.

Model testing can begin once the model is finished. Model testing is concerned with testing that the model actually manages to separate normal and abnormal data in a proper way. The model is also tested to see if it can have different thresholds correctly. Since an unsupervised learning method is used, the model's performance on detection of fraud cannot be examined. However, this can be accomplished by the use of data sets that have synthetic fraud actions inserted or data sets which contains actual known fraud.

Results analysis will be conducted after a model has been tested. In the case of several models, the results from different models can be compared to each other to find similarities and differences. The results should also be analyzed with regard to further improvement of the profiles.

Results discussion and conclusion is the final stage of this project. The results are evaluated in relation to what we expected to find. Answers to all the research questions are discussed, together with a final conclusion on the result of the hypothesis' test. The conclusion summarizes the results found in this project, together with a discussion related to our findings.

Reflection and development of a platform for future research is an additional stage, in which it is attempted to evaluate the work done, the choices made, and what could have been done better. A reflection on the work done provides valuable information for others conduction similar projects. We also wish to create a platform for others who wish to bring this work further. By creating such a platform we can use the experience gained during this project to emphasize areas that are most beneficial to research further.

3.3 Threats to validity

An important and fundamental part of doing research is to evaluate the validity of the results. In order to plan and achieve valid results, validity should be considered as early as in the planning phase of a project. The following sections describe the threats to validity we have considered and evaluated for the results of this project.

3.3.1 Validity evaluation

There are two important aspects of validity with respect to the results. The first is that the result should be valid within the population from which the samples are drawn. Secondly, the results could be generalized to be valid within a larger population. Relating this to the population in this project, the results first need to be valid for the specific bank logs used, before they may be generalized to be valid for bank logs in general. The results are valid to an adequate degree if they are valid for the generalized population [WRH+00].

In the literature there may be found several classification schemes for all types of threats to validity of a project. We have chosen to use Cook and Campbell's list of threats [Coo79]. They define four main threats, and state that every threat is a special case of one of these. The main threats are threats to conclusion, internal, construct, and external validity.

In [WRH+00] a set of more specialized threats are described for each of the four main threats. We have chosen to use this set as a foundation for our work to reduce the threats to validity. Although some of the threats listed in this set are not of direct relevance for this project, we find it important to list these as well to make explicit the fact that they are considered.

3.3.2 Conclusion validity

This validity is concerned with the relationship between the treatment and the outcome. We wish to ensure that there is a statistical relationship with a satisfying significance. Threats to conclusion validity are concerned with issues that affect the outcome of an experiment. These issues include, for example, choice of statistical tests, choice of sample size, care taken in the implementation and measurement of an experiment.

Low statistical power The power of a statistical test is the probability that a test will reject a false null hypothesis [MDW+95]. This type of error is referred to as a false negative error. The statistical power depends upon statistical significance, the variance in the populations and the sensitivity of the data. The possibility for low statistical power is an issue that needs to be addressed in this project. The research done in the field of anomaly detection has shown that the false negatives might exist, but we will try to keep this rate as low as possible. There exists a clear trade-off between the rate of false positives and false negatives. It is more important to keep the rate of false negatives as low as possible instead of having false positives, due to the fact that it is worse to classify an actual intrusion as normal than the opposite.

Violated assumptions of statistical tests Since certain tests assume properties in the data, it is important to maintain these assumptions in order to avoid miscalculations and erroneous conclusions [MDW+95]. The impact of a miscalculation, and ultimately a wrong conclusion, would be fatal and should be avoided by all means. Hence we need to pay special attention to the assumptions of statistical models before using them.

Fishing and the error rate This threat consists of two separate threats. First there is the threat of fishing, or searching, for a specific result. The second is concerned with the actual significance level of a test, and how this can be wrong when performing multiple analyses [MH96]. For this project we find the first threat the most important and the second of lesser importance. Fishing for results is obviously a destructive thing to do. Therefore, we need to have an objective approach when evaluating results and interpret the result as little as possible as to direct the result in one way or another.

Reliability of measures Measurements could be imprecise and unreliable. This could result in inaccurate results and false conclusions. As a basic principle, the same result should be acquired when measuring a phenomenon twice. For this project, all measurements are done automatically and stored digitally, so the measurements should already be quite reliable. However, in order to leave all doubt aside, all measurements and calculations should be done twice. Because this is done automatically, it should not lead to neither significantly increased workload nor other complications. The basic principle should always hold.

Random irrelevance in experimental setting This threat is concerned with elements outside the observed setting that could disturb the results. For our project we might find irrelevant data in the logs originating from occurrences not related to the actual actions in the log. This could be things such as power-out, restarts or software updates. We need to keep this in mind when analyzing the log to prevent irrelevant actions from being classified as malicious behavior.

Random heterogeneity of subjects When doing statistical tests it is always a risk that the population's individual differences are larger than the effect to be measured [JSDJ01]. For our case this would for example mean that the difference in behavior between individual users is in general larger than the difference between normal and abnormal users. After evaluating this threat we have found that this should not be a

problem for our case. The very definition of abnormal behavior is to find those cases where a user deviates beyond the normal deviation that is seen inside the group of normal users. Hence, random heterogeneity is taken into account and is not an issue that could affect the validity of the outcome.

3.3.3 Internal validity

If a relationship is observed between the treatment and the outcome, we must make sure that it is a causal relationship, in contrast to being a result of a factor of which we have no control or have not measured. In other words, we must ensure that the treatment causes the outcome. Threats to internal validity concern issues that may indicate a causal relationship, although there is none [SEF08]. Factors that impact the internal validity include how the subjects are selected and divided into different classes, how the subjects are treated and compensated during the experiment, and whether special events occur during the experiment. All these factors can make the experiment present a behavior that is not due to the treatment, but rather to the disturbing factors.

History It is important to note that the measurements could be performed at different times, and that different circumstances could lead to different outcomes. This could lead to inaccurate results and misleading data. The historical threat is something we know with certainty is present in our data sets. The data will be affected by system updates and other occurrences that could lead to change in the behavior patterns. Because we are dealing with log data, it is impossible for us to minimize this threat. We have no way of knowing the historical changes besides from what information is present in the logs. This is a threat that will be highly relevant for this project and could lead to a loss in validity of the results.

Maturation Maturation is the effect that the same user could react differently as time passes by and he learns the system. When trying to create profiles of user behavior, this is something we need to take into account. It would be normal for the user to be more effective and accurate as the system's functionality is learned. The maturation threat should be present in system logs, such as those we are dealing with. Hence we need to keep in mind the possibility that users change behavior over time.

Instrumentation Instrumentation looks at how and what data is collected. Here, there are many threats that could lead to loss in the validity of the results. The most important threat is that important data may not be collected, which leads to the data set missing important elements that affect the rest of the data. Another threat is that the data is stored inaccurately, for instance with loss of precision or by generalization. The instrumentation threat is very valid for our project, but it is also impossible for us to remedy. Hence, we have to recognize it as a threat and look at it as a possible affect that could threat the validity of our results.

Selection Selection is concerned with the fact that the result could be different depending on how subjects are selected from a large group. To relate this to our project, we possess information on a lot of different users. Some of them are single users, which would be rather easy to extract from the log, while others are more advanced users, for which more work is needed in order for their behavior to be fully extracted. Based on the selection threat we see that it is important to make sure that we are able to extract as many users as possible in order to get the best possible selection from the population.

Mortality Sometimes users quit before they are finished, leaving behind an incomplete data trail. For a statistical evaluation this could lead to results being affected, but for our project this will not be a problem. As each log record is part of a session that is initiated when a user logs in, we are able to detect when this session terminates, and hence, where the user exits. This should then not be a problem and should not influence the results.

3.3.4 Construct validity

This validity is concerned with the relation between theory and observation [PS00]. If the relationship between cause and effect is causal, we must ensure two things. Firstly, we must make certain that the treatment well reflects the construct of the cause, and secondly, that the outcome well reflects the construct of the effect. Threats to construct validity refer to the extent to which the setting of the experiment actually reflects the construct under study. For example, the number of courses taken at a university of computer science may be a poor measure for a subject's experience in a programming language. This has poor construct validity. An example of better construct validity may be the number of years of practical experience.

Inadequate pre-operational explication of constructs This threat simply means that constructs, i.e. systems, are not sufficiently described and explained [16]. If the understanding of the system and its functionality is unclear, then its analysis cannot be clear either. In order to avoid this we have taken precautions and thoroughly studied the system and its log material. We have also received training and lectures by specialists who work with the system on a regular basis.

Mono-method bias This threat is concerned with the consequences of only using a single type of measures or observations [16]. This is because there is no way of verifying the results by, for instance, a cross validation with other results. This threat is partly present in our data, although it is probably not a big threat. We are given data from one source alone, and have no way of checking the data against logs other than the ones provided. On the other hand, we have a large data set with several different logs at different levels of the applications, thus based on these, some validation could be performed by checking the consistency of the data across the different logs.

Experimenter expectancies The researchers conducting the experiments could bias the results in several ways based on their expectations. For instance, leaving out important data or reformulation of test could be done based on the expectations of the results. We find it important to work and evaluate in an objective manner in order to not let our expectancies influence the results.

3.3.5 External validity

The external validity concerns generalization [SEF08]. We wish to see if the results of our study can be generalized to a larger scope. In order to do so we must verify that relationships between construct and cause are valid within the new scope. Threats to external validity concern the ability to generalize results of experiments beyond the experiments' populations. External validity is affected by the chosen design for the experiment, by the objects in the experiment and the subjects chosen. There are three main risks: having wrong participants as subjects, conducting the experiments in the wrong environment, and performing it with a timing that affects the results.

Interaction of selection and treatment This is a threat affected by having a subject population not representative for the one aiming to generalize to. We feel that this is not an issue with our population. Users

of an Internet banking application for a certain bank should be representative for users of Internet banking applications in general.

Interaction of setting and treatment This threat is affected by the lack of representative material and experimental settings. Since our work is conducted on real world data we find this threat negligible.

Interaction of history and treatment Since time and date could influence the results, this threat looks at the timing of the experiments. For instance, the day before Christmas should not be regarded as a normal day for money transactions. It is a threat that definitely is relevant for the data used by this project. Since we use real world data we expect to see differences in data properties based on both time and data. It is hard to deal with this problem on a general basis, but hopefully, we can cope with this by generalizing the data over larger intervals than single hours or days.

Chapter 4

Design

Summary

In this chapter we describe the steps which need to be carried out in order to implement a functioning anomaly detection system. It begins with the formulation and design of different profile suggestions. They are designed in consideration of capturing user behavior at different levels of information abstraction. These profiles are then evaluated based on a set of criteria, in order to select a manageable subset for further use. We then look at the available data sources and design log parsing solutions that can provide consistent information for the machine learning methods. Finally, we describe the machine learning methods that will be used to implement the selected profiles. We find the configurations for these methods that will perform in the best best manner to our needs.

4.1 Profiles

When considering profile building in anomaly based intrusion detection, the aspects to consider are numerous and varied. Due to the level of customization and specialization of servers and applications, little information about profile building can be deducted on a general basis. The solutions available are mostly ad-hoc solutions serving for a server or application. However, some efforts have been made to create a framework for construction of features and models for intrusion detection systems [LS00]. Mining audit data for automated models for intrusion detection (MADAM ID) is the most applicable framework available. It aims at using data mining techniques to extract useful models from large stores of data. This framework is of interest based on one of the basic assumption made [Lee99]. *"A basic premise for intrusion detection is that when audit mechanisms are enabled to record system events, distinct evidence of legitimate activity and intrusions will be manifested in the audit data."* This assumption is also one of the fundamental assumptions made by this project.

MADAM ID proposes three types of data mining algorithms that are particularly useful for generating anomaly detection models: *classification, link analysis and sequence analysis*. Classification pertains to mapping the data into one of several predefined categories. Link analysis searches to find relations between fields in a database's records. Sequence analysis discovers which time-based sequences of audit events frequently occur together. The properties these algorithms detect give valuable information of what types of properties to look for in the data.

The following sections provide a selection of possible profiles which could be used in an anomaly intrusion detection scheme. The profiles look at the data at different levels of abstraction in order to gain knowledge and insight into possible criteria where attacks could be performed, ranging from an analysis of a single parameter's value to a total picture of user behavior.

4.1.1 Profile 1: Request structure

The first profile looks at the structure of a request. The entity in this profile is a single user request to the server. The profile consists of six different models, that look at the structure and logic of a request, in order to detect faulty and malicious request structures. A detailed description of this profile can be seen in table 4.1.

Table 4.1: Models for profile 1: Request structure

Request structure	
Model	Request structure
Description	Looks at the structure of the request.
Detects	Illegally crafted requests.
Learning	Learns grammar for legal structure for requests.
Detection	Checks each request against the learned grammar for requests. If a request does not fit the grammatical structure, it is considered an anomaly.
Model	Path
Description	Looks at which path is accessed.
Detects	Directory indexing and predictable resource locations.
Learning	Empirical list of legally accessed paths.
Detection	Requests for paths not in the list are considered anomalies.
Model	Parameter order
Description	Looks at the order in which parameters are passed in a request.
Detects	Crafted attacks not aware of application logic or structure.
Learning	An attribute precedes another attribute when they appear together in the parameter list of at least one query and comes before in the ordered list of attributes of all queries where they appear together.
Detection	The detection process checks whether the attributes of a query satisfy the order constraints determined during the learning phase.
Model	Parameter
Description	Looks at what parameters are present in the requests.
Detects	Attribute probing, backdoors and logical attacks.
Learning	Valid relations between path and parameters are learned for each unique path.
Detection	If a request contains a parameter not related to the path, it is flagged as an anomaly.

Continued...

Table 4.1: Models for profile 1: Request structure – Continued

Model	Value length
Description	Looks at the length of each parameter's value.
Detects	Buffer overflow, injection attacks demanding more space than provided.
Learning	Approximates the mean and variance of the real attribute length distribution by calculating the sample mean and the sample variance.
Detection	Given the estimated query attribute length distribution, with parameters mean and variance as determined by the learning phase, it is the task of the detection phase to assess the anomaly of a parameter with a certain length, given a threshold value.
Model	Value type
Description	Looks at the character types of the parameters' values.
Detects	Injection of invalid data.
Learning	Learns the value type by looking at a sample of value types for each attribute, e.g. string, Boolean, integer, and double.
Detection	Checks the value type against the learned type for the attribute.

4.1.2 Profile 2: Request values

Profile 2looks at the values of different parameters set by the user, in order to detect abnormal values, e.g. malicious code. This profile goes beyond a traditional input validation by learning character distribution and structural inference. It searches to detect all kinds of attacks that need to pass data on to the server, thereby leading to a security breach, such as injection attacks. A detailed description of this profile can be seen in table 4.2.

Table 4.2: Models for profile 2: Request values

Request values	
Model	Attribute character distribution
Description	Looks at the distribution of characters.
Detects	Injected program code that deviates from normal legal text by the distribution of characters.
Learning	For each observed request attribute, its character distribution is stored. The idealized character distribution (ICD) is then approximated by calculating the average of all stored character distributions.
Detection	Given an idealized character distribution, the task of the detection phase is to determine the probability of the character distribution of a request attribute being an actual sample drawn from its ICD.
Model	Structural inference
Description	Looks at the structure of the request attributes.
Detects	Injected code with different structural properties than natural grammar would inhibit.
Learning	When structural inference is applied to a request attribute, the resulting grammar must be able to produce at least all training examples.
Detection	Once a model has been built, it can be used by the detection phase to determine the probability of request attributes.
Model	Token finder
Description	Looks to see if a value of an parameter is drawn from a limited set of values.
Detects	Injected special characters.
Learning	The classification of an argument, as either enumeration or random value, is based on the observation that the number of different parameter values is bound by some unknown value in the case of an enumeration, while it is unrestricted in the case of random values.
Detection	Once it has been determined that the value of a query attribute consists of tokens drawn from an enumeration, any new value is expected to appear in the set of known values.

Continued...

Table 4.2: Models for profile 2: Request values – Continued

Model	Attribute presence or absence
Description	Looks for regularities in attribute values.
Detects	Changed or injected special characters.
Learning	Create a model of acceptable subsets of attributes that appear simultaneously in a query. This is done by recording each distinct subset of attributes that are seen during the training phase.
Detection	The algorithm performs, for each query, a lookup of the current attribute set.

4.1.3 Profile 3: Server response

The server response profile searches to detect if sensitive information is falsely being disclosed to a user. It tries to learn how the server responds to different user requests, in order to detect when a user gains a server response that is not intended. A detailed description of this profile can is shown in table 4.3.

Table 4.3: Models for profile 3: Server response

Server response	
Model	Response
Description	Looks for relationship between requests and responses.
Detects	Detects mismatches in request response pairs.
Learning	Learning the normal pairs of user request and server response, and use these pairs to calculate the statistical probability of each new pair.
Detection	Detects all pairs of user request and server response that have a sufficiently low probability.

4.1.4 Profile 4: Session structure

The session structure profile looks at each session as an entity for learning. A session is defined as a sequence of interactions between a user and the server, from login to logout, or time-out. During this interaction it should be possible to learn a pattern of how a user normally interacts with the system,

enabling detection of abnormal patterns of interaction. Furthermore, the session can be analyzed to detect types of malicious actions searching to bypass application logic or searching for non-public resources. A session provides a good collection of information in order to detect robotic activity as well [PPLC06]. A detailed description of this profile is shown in table 4.4.

Table 4.4: Models for profile 4: Session structure

Session structure	
Model	Inter-request time delay
Description	Looks at the time delay between successive client requests to discover abnormal request times.
Detects	Automated attacks and robotic activity.
Learning	A distribution of normal time delays between successive client requests is created.
Detection	During the detection phase, a distribution of time delays between successive requests from a client is compiled. The goal is then to determine the probability that the observed time delays between successive requests is a sample from the learned distribution.
Model	Invocation order
Description	Looks at the sequence in which a session requests the different resources and services of the site.
Detects	Logical attacks bypassing functionality.
Learning	The learning phase starts with a non-deterministic automaton that outputs exactly the strings that represent the sessions, with their corresponding sequences of requests.
Detection	During the detection phase, a query is associated with its corresponding session. When this session can be derived from the automaton that was built during the training phase, the output from the model for the query is normal.

Continued...

Table 4.4: Models for profile 4: Session structure – Continued

Model	Invocation frequency
Description	Looks at the frequency by which a session invokes the different services and requests.
Detects	Detects attacks that seek to stress or probe the system, for instance DoS)
Learning	The learning of invocation frequencies consists of creating a model of different pages and services, with a mean invocation frequency and variance for each of them.
Detection	Each new invocation from a session will be collected and the collection will continuously be evaluated against the mean frequency learned. If a session's collection breaches the legal variance of invocation frequencies, it is abnormal.
Model	Session time
Description	Looks at the time length of a session.
Detects	Looks for sessions with abnormally long durations, perhaps prolonged by some mechanism, to perform malicious actions on valid sessions.
Learning	The learning consists of learning the mean session time, together with the variance, in order to create an upper bound for session times.
Detection	Any session time breaching the upper bound for sessions will be considered abnormal.
Model	Session time per invocation
Description	Looks at session time per invocation.
Detects	Looks for sessions prolonged by limited activity in order to prevent timeout and keeping the session valid.
Learning	Learns the normal ratio between session time and invocations, together with the variance, for normal sessions.
Detection	Any ratio deviating more than the learned variance is considered abnormal.

Continued...

Table 4.4: Models for profile 4: Session structure – Continued

Model	Invocations per session time
Description	Looks at the number of invocations per session time entity.
Detects	Looks for sessions with a high rate of invocations within a short period of time, in order to detect automatic activity which does not breach the detection bound for single requests.
Learning	Learns the normal ratios between invocations and session time, together with the variance, for normal sessions.
Detection	Any ratio deviating more than the learned variance is considered abnormal.

4.1.5 Profile 5: User profile

In order to detect when a malicious user has gained illegal access to an account, one needs to learn how the legal user normally acts. When a baseline of normal user interaction is created, it should be possible to detect deviations from this norm compared to other users. Using a user's behavior characteristics, it could be possible to detect masqueraders who have gained access to an account. A detailed description of this profile can be seen in table 4.5.

Table 4.5: Models for profile 5: User profile

User profile	
Model	User statistics
Description	Determines statistics about the user's behavior, such as login times, session durations, services used, number of transaction made per session, and number of logins per week.
Detects	Detects actions that deviate from the statistical norm.
Learning	Learning consists of generating the statistical values for the attributes chosen.
Detection	Actions that differ from the user's statistical values will be defined as abnormal.

Continued...

Table 4.5: Models for profile 5: User profile – Continued

Model	Usage patterns
Description	Looks at the usage pattern of a user.
Detects	Finds usage patterns indicating abnormal, and possibly malicious, user activity.
Learning	Creates a state chart of different activities and the transition between them.
Detection	Unusual or strange state transitions.
Model	Locations
Description	Looks at the locations from where a user is logged in.
Detects	Detects abnormal locations within a time frame, indication that the user's credentials are being used by another user.
Learning	Needs no learning phase, but needs to keep track of user locations within a given time frame.
Detection	If a user's locations deviate more than what is possible for a user to move within the time frame, it is a strong indication that one of the two locations is by a malicious user.
Model	Transaction amount
Description	Looks at the amount that are transacted to and from an account.
Detects	Abnormal transaction amounts may be a indication that a fraudulent transaction is being performed.
Learning	The learning phase establishes the normal transaction amounts for the users, and determines a threshold for normal values.
Detection	Detection consists of checking transaction amounts against the threshold, flagging all transactions that breach it.

Continued...

Table 4.5: Models for profile 5: User profile – Continued

Model	Transaction frequencies
Description	Looks at the frequencies of transactions involving an account.
Detects	Abnormal transaction frequencies may indicate attempts to bypass fraud detection by performing many transactions.
Learning	The learning phase needs to learn the frequencies of transactions by a user within different time frames, e.g. days or months, together with variances of these frequencies.
Detection	Detection consists of checking transaction frequencies at the different time frames, notifying if the frequencies breach a threshold.
Model	Transaction recipients
Description	Looks at the number of recipients of transactions involved in an account.
Detects	Abnormal number of transaction recipients may signal an attempt to bypass fraud detection by performing transactions to several recipients.
Learning	The learning phase needs to learn the number of unique recipients, and new recipients within different time frames, together with their variances.
Detection	Detection consists of checking transaction recipients against the number of recipients within the different time frames, notifying if some threshold is breached.

4.1.6 Profile 6: System overview

In order to detect attacks done by several users the system as a whole must be considered. Distributed attacks usually aim to stress and overwhelm the system, such that regular users loose, or get limited, functionality or response. DDoS attacks could be detected by a sudden raise of service requests or by an unusually high system load in general. A detailed description of this profile is shown in table 4.6.

Table 4.6: Models for profile 6: System overview

System overview	
Model	System load
Description	Looks at the total load of system resources.
Detects	Detects situations where the system resources' workload reaches abnormal levels due to abnormal user activity.
Learning	During the learning phase, the normal load of the system resources, in terms of the number of users and their activity, is learned.
Detection	Detection consists of monitoring the load of the system resources and comparing it to the normal load values learned. When the load reaches abnormal values, it may be a sign that some kind of abnormal, resource demanding, activity is in progress, such as a DDoS.
Model	Service requests
Description	Looks at the total number of service requests on the system within a time frame.
Detects	This model would detect attacks searching to overwhelm the system by requesting a high number of services, and hence, limiting the responses from these services to other legitimate users.
Learning	This model needs to learn the average number of service requests by a user, which with a variance can be used to calculate the total expected request distribution.
Detection	When the service requests breaches the largest accepted value per user, it is considered an abnormal event.
Model	Access frequency
Description	Looks at the frequencies at which services and pages are requested by the total number of users.
Detects	This model searches to find distributed attacks bypassing the previous models, by requesting different pages and services without any special pattern, such as, for instance, requesting only specially demanding services.
Learning	Statistical charts of access frequencies for different time intervals.
Detection	Statistical abnormalities in the chart.

4.2 Profile selection

After a brief review of the created profiles it became clear that we had to select a subset of them in order to keep the work amount within the limitations of this project. In order to find the suitable subset of profiles to proceed with, we had to evaluate each profile. The evaluation was carried out in two steps. Firstly, we compared the profiles to the available data material, and estimated the work amount needed to create them. This was then measured up against advantages and drawbacks for each profile in order to see if the work effort would be worth the cost.

Secondly, we conferred with the administration of the Internet bank providing the data material used by this project. We interviewed them about which types of profiles they thought would be the most appropriate for us to select. It became clear that the Internet banking system had several security measures already, and they wanted to use profiles that would complement these measures.

The different profile suggestions were evaluated by these two steps in order to find our final subset of profiles. The evaluation of each profile, and the conclusion, are described next.

4.2.1 Profile evaluation

The profiles were evaluated by the criteria we found the most relevant: complexity, match with available data, functionality, and feedback from the administration. Based on these evaluation criteria, we were able to analyze each profile to find the appropriate subset. In order to not rush into any premature conclusions, we tried to analyze all profiles individually before comparing them to each other. We wanted to end up with a subset of non redundant profiles that could be implemented and tested within the determined frames of this project.

The profiles were evaluated by the criteria which we found the most relevant. These were complexity, match with available data, functionality, and feedback from the administration. Based on these evaluation criteria we were able to analyze each profile to find the wanted subset. In order to not rush into any premature conclusions, we tried to analyze all profiles individually before comparing them up against each other. We wanted to end up with a subset of non redundant profiles that could be implemented and tested within the determined frames of this project.

Profile 1, *request structure*, looks at how the request from a user is structured. An analysis of the request structure could find deviations indicating that a user tries to pass additional values, skip mandatory values, or by other

means attempts to trick the system into allowing unauthorized functionality. This profile looks at the cornerstone of the available data, namely the instances, and hence, the importance of analyzing an instance at this level is high.

One of the main advantages of the request structure is that it is basic in both functionality and structure. It should be easy to implement a working solution for this profile. Another important aspect is the fact that it exactly matches the available data as it does not necessitate rearranging, restructuring, or transforming the data. It is also a relevant profile for assuring consistency and quality of the data before it is used by other profiles. This results in a further strengthening of this profile's importance.

A drawback of this profile is that it is very simple, perhaps too simple to be given any significance. To give the request structure any significance we should analyze the deviations more thoroughly, indicating that profile 2 could be involved. It is also important to notice that since we do not have any information about the audit mechanisms, we do not know if the instances are transformed into a common format, thus loosing structural information. This presents a threat to the validity of the profile, but not however a threat to its relevance and functionality.

A final drawback of the request structure profile is that it looks at an instance without taking into account the context in which the instance appears. This could lead to misinterpretations and a higher level of false positives than necessary. Even though we rely on a fair amount of consistency in the logs, there may be incidents that deviate from the normal format. Examples of this are error messages and system failures. A summary of the evaluation of this profile can be seen in table 4.7.

Table 4.7: Evaluation of profile 1: Request profile

Profile 1:	**Request structure**
Advantages	Easy to implement. Cornerstone functionality. Good match with data.
Drawbacks	Too simple. No context. No significance alone.
Conclusion	Basic functionality.

Profile 2, *request values*, looks at, and analyzes, the attribute values of each instance. This is a refinement of profile 1 that discards the structure and focuses on the values. The advantage of such a profile is that it can analyze each attribute's value more thoroughly. It can go further than the basic properties, such as length and character type, and look at more advanced

properties such as character distribution and structural inference.

Another advantage is that such a profile can be customized for each type of attribute, leading to a better understanding of an attribute's properties and values. This further results in a greater possibility of detecting malicious or fraudulent values. This also means that it would be harder to bypass security measures, as all values passed by the user to the server would be analyzed by this profile.

One of the drawbacks of such an approach is actually also one of its advantages. The need for a customized approach for each attribute would imply a large amount of work, due to the specialization. A possibility that limits this drawback is to generalize the different attributes to such an extent that the work amount would be within a satisfactory range.

Another drawback of this profile is that it does not consider how attributes relate to each other and hence how the values are influenced by other values. This is a property that could be useful to analyze, but which would inflict a heavier work magnitude. For a summary of this evaluation's main points see table 4.8.

Table 4.8: Evaluation of profile 2: Request values

Profile 2:	**Request values**
Advantages	Specialized. Analyze data at the lowest level. Customizable for different attributes.
Drawbacks	Specialization. Probable high workload. No relationship between attributes' values considered.
Conclusion	Highly customizable.

Profile 3, *server response*, looks at pairs consisting of user requests and server responses. It would seem a good idea to look at these pairs to find requests that seems normal, but which receives strange server responses. Such pairs could indicate that some sort of malicious request has lead to an unintended server response. This would be a very useful profile if one was able to implement it. The main advantage of such a profile is that it is fairly simple. The simplest approach does not have to consider values and attributes, just the primitive request and response types. However, this approach could be conducted in a much more complex way depending on how much information is included.

A drawback of such a profile is that it is unknown what level of significance the pairs carry. It would be difficult to find a sufficient abstraction level for how much information to take into account for each pair. An overly

simple approach would only be able to detect anomalies on a limited set of properties, while a too complex approach would become complex and lack statistical significance. Both profile 1 and 2 could be used within a complex version of profile 3.

Another drawback of this profile is that user requests and server responses do not appear clearly in the available data, hence the need for functionality that can extract pairs is apparent.

A final drawback for this approach is that it neglects parts of traffic between the user and the system. This is because not all user requests result in a server response. This reduces the completeness of the detections done by this profile alone. The summary of the evaluation of profile 3 is found in table 4.9.

Table 4.9: Evaluation of profile 3: Server response

Profile 3:	**Server response**
Advantages	Looks at an important aspects. Easy in its most basic approach.
Drawbacks	Very complex when more information is taken into account. No direct match with data. Low significance.
Conclusion	Complexity issues.

In many ways can profile 4, *session structure*, be looked at as an extension of profile 3. However, in this profile it is more likely to only look at the user requests. However, a possible extension would be to take server response into account as well. Profile 4 looks at how a user navigates through the system's web interface. This profile can find strange jumps between web pages that could indicate malicious acts such as bypassing server logic or authorization escalation.

The main advantage of this profile is that it can detect attacks that need some history and context to be detected. It takes into account how the system usually is navigated through, and uses this information to create a baseline for normal activity. This is an easy way to learn the regular user's behavior in the system.

It also became clear that this profile was one of the preferred choices by the system administration. They believed that this kind of profile would elaborate on security issues of high interest.

The main drawback is that such a profile would quickly become complex and the complexity would escalate exponentially as the number of states increase. Complexity would further increase if one wants to look at sequences

of transitions between states. A possible way to lower the complexity would be to generalize the states, but in a system as comprehensive as the one under study, this benefit would be limited.

Another drawback with this profile is that it generalizes users into a common system interaction model. Every user counts equally in this profile, which would lead to even more unorganized and useless data. Clearly, a system can have several different user types, ranging from novices to more advanced users. The borders between user types are probably very vague, due to user maturation, which again would lead to even more vague interaction patterns. A synopsis of this profiles' evaluation can be found in table 4.10.

Table 4.10: Evaluation of profile 4: Session structure

Profile 4:	**Session structure**
Advantages	Includes history and context. Finds attacks that the previous profiles are unable to. High interest to administration.
Drawbacks	Becomes quickly very complex. Exponential growth. Generalizes a large quantity of different users.
Conclusion	Generalizable, includes context.

There are several similarities between profile 4, *session structure*, and profile 5, *user profile*. They consider the same data, at the same levels of scope, but different subsets. Profile 5 analyzes the behavior of single users in order to find deviations from *this* user's behavior, which would suggest that the behavior is that of some other person than the legitimate user. The main advantage with this type of profile is that we can look at a single user at a time in order to learn how he interacts with the system. Based on this we can learn what to expect from this user and use this to detect stolen credentials or Trojan activity.

Sadly, this approach has some large drawbacks. Firstly, is the fact that in order to learn a user's normal behavior, we must collect a considerable amount of data for each particular user. This is because we need to be able to lay some statistical significance to the behavior profile, in order to detect deviations from it.

The second drawback is a natural property all users have. A user will mature and learn the system, leading to natural changes in the usage pattern. This is a natural course of change that would make this user profile even more difficult. With regards to statistical relevance and data validity, a considerable amount of continuously updated data would be needed. This

is obviously not a preferable situation. For a summary of the evaluation of profile 5, see table 4.11.

Table 4.11: Evaluation of profile 5: User profile

Profile 5:	**User profile**
Advantages	Specialized, looks at single users. Can detect changes in behavior.
Drawbacks	Low statistical relevance. High need for data validity and updates.
Conclusion	Potentially low statistical relevance.

Profile 6, *system overview*, looks at the total performance of the system in order to detect cooperating users attacking and draining system resources. The main advantage of this profile is that it can detect attacks originating from several cooperating users. This is also the only approach that can deal with such attacks that we consider in this project.

This profile has a set of prominent drawbacks. Firstly, the fact that cooperation attacks, such as DDoS, attacks the performance of the system, trying to overwhelm it into being unable to respond at all. The audit processes would also be influenced by such an attack, making the data source compromised and unsuited for use to detect such attacks.

Secondly, that there is no data items in the available logs that include information that directly could be used by such a profile. This means that the logs need to be analyzed in order to find the extract the needed data, and then used to find system performance.

A final drawback is that this profile received little interest from the system administration, who thought that other mechanisms would be more suited for detection of cooperation attacks. The summary of the evaluation of profile 6 can be found in table 4.12.

Table 4.12: Evaluation of profile 6: System overview

Profile 6:	**System overview**
Advantages	Detect attacks from cooperating attackers.
Drawbacks	No support in data. Not of interest to administrators
Conclusion	Unfortunate data source.

4.2.2 Profile comparison

After evaluating each profile independently we compared them up against each other to find a non-overlapping subset of profiles that could be used further in this project.

Firstly, profile 6, system overview, was eliminated based on the complexity and relevance of the model, and difficulty with using the data in such a profile. This left us with five profiles, with overlapping contributions and clear strengths and weaknesses.

Profiles 1 and 2 seemed to complement each other in an attractive way. As profile 1 is very basic in its approach, profile 2 searches to analyze the attributes more thoroughly. This lead to the conclusion, that both profiles were highly relevant for further work. However, alone they seem to have clear limitations, so we proposed a new profile which combined the functionality of both. This profile, *user requests*, is our first choice for further work.

Profiles 3, 4, and 5 have a lot of common responsibilities. It seems that profile 4 lacks most problems related to profile 5, while still being able to keep most of the strengths and relevance. A session structure profile would be generalized and not need the amount of data leading to the complexity issues to a user profile. It would also have a large pool of data to draw from and the model would surely have statistical significance. However, by generalizing data into one model, we would end up with one complex model, instead of several simpler ones. Based on this we propose a basic implementation of profile 4 that can be expanded to include elements from both profile 3 and 5 in later stages of development.

Table 4.13: Selected profile 1: User requests

User requests	
Increment 1	Request structure. Parameter order. Parameter. Value type. Value length. Token finder.
Increment 2	Path Attribute character distribution. Structural inference. Attribute presence or absence.

If a successful model of the user's traversal through the systems web pages can be made, it can be specialized further to include different kinds

of user groups and data related to user requests and system responses. It is important to emphasize that the specialization only should be worked on if the basis of profile 4 is implemented successfully. This leaves us with two non overlapping profiles that will be used further in this project: the *user requests* and *session structure* profiles.

Table 4.14: Selected profile 2: Session structure

Session structure	
Increment 1	Invocation order. Usage patterns.
Increment 2	Inter-request time delay. Session time. User statistics. Response.

Table 4.13 and 4.14 shows a summary of the models each profile will consist of. The details for each model are found in the preliminary profiles described in section 4.1. As seen, the profiles consist of two increments. Increment 1 represents the main functionality wanted by the profile while increment 2 represents additions that can be added if time should allow it.

4.3 Data preparation

In order to use different machine learning methods to build the chosen profiles, we must prepare the available data. The preparation must be performed in order to transform the data to a format that can be utilized efficiently by the learning methods. Two key points are instance integrity and attribute data type. The instance integrity is important since the learning methods view instances as equal. If the data instances do not have a common format, they need to be transformed into one. Attribute data type needs to be specified for each attribute value, so the learning method knows what data it is receiving. These two points can be achieved by a log parsing scheme that process the data sources before they are handed over to the machine learning method. In the following text we describe the available data sources and the data parsing that is conducted.

4.3.1 Data Source

The data for the creation of the user request profile originates from two different sources. Each of them will have a personal model for detection of

anomalies. The first is the transaction log, which consists of all the transactions performed by the users. The second is the web log, which logs the interactions between users and the system interface. Each of them will be described in more detail in the following text.

We also had another log available, the application log. The application log is written to by several sources, meaning that it contains data material written in different formats. Hence, the application log has a very low consistency among the different instances. A review and evaluation of this log was conducted in a prior project [KK07]. We refer the reader to this project's report for a more thorough review and analysis of this log. However, the conclusion of this evaluation was that the application log was unsuited for direct use by machine learning methods.

The transaction entries from the transaction log consists of eighteen fields, made up by nine numerical, three textual, and two date fields, in addition to four flags, i.e. boolean fields. Each of the fields has a different range of values. In order to keep the confidentiality of the log at a satisfying level, we can only describe them in general terms. The numerical values range from zero to the magnitude of 10^{20}. The two date fields have different formats. The first contains values describing a day, while the other specifies the date and time with the granularity being milliseconds. The textual fields provide written information, some provided directly by the user. Finally, the flags have a binary representation, i.e. either *on* or *off*. The general structure of each transaction instance is as follows:

integer, integer, integer, date, integer, integer, integer, integer, integer, date, integer, string, string, string, flag, flag, flag, flag.

Most of the fields' meanings are known to us by analysis of their contents. However, some of them are hard to discover without a better documentation of the log than that available to us. This has led to difficulties in attribute selection, due to the fact that we have limited knowledge of the importance and significance of some fields. Hence, we have decided not to remove any fields with unknown meaning as a precaution. This is because we do not wish to remove any vital information that could lead to weakened detection performance.

The formats for the rest of the entries, those not directly involving transactions, are somewhat different. However, the strict formats for these entries are not considered important in this project.

The web log records the interaction between the user's browser and the system's interface. It approximately follows the log format referred to as

combined log format, see section A.2. The combined log format is an extension of the more known common log format (CLF), see appendix A.1. The general format of these log instances are:

remotehost rfc931 authuser date "request" status bytes "user_agent" integer session_id

An example of an instance written in this format would look like this:

125.125.125.125 - - 2007-08-17T16:15:05 "GET /index.html HTTP/1.1" 200 1043 "Mozilla /4.05 [en] (WinNT; I)" 125 asdadJGKJHGS8765sakdjh

For the explanation of the different fields we refer to the appendix. The web log provides a much more structured and lucid data representation. Each field is known and follows a common format. This makes the work done on this kind of logs easier and more efficient.

4.3.2 Data properties

Even more interesting than the data sources and structure are the data properties. In this section we describe these properties in terms of different statistics. These statistics are of importance since they describe both the quality and the completeness of the data. *Min* and *Max* is defined by the highest and lowest values of an attribute. The *Mean* is the average of all values of one attribute and *Unique* shows how many unique values an attribute contains. The *Null* value defines the percentage of instances where the attribute has no value. The letters *NA* in any of the cells indicate that the specific property for that field is not available.

Table 4.15: Data properties of web log

Attribute	Min	Max	Mean	Distinct	Null
remotehost	*NA*	*NA*	*NA*	13549	0%
request	*NA*	*NA*	*NA*	6900	0%
status	200	500	221	9	0%
bytes	0	1463000	9200	25822	47%
user_agent	*NA*	*NA*	*NA*	2653	1%

Table 4.15 shows an overview of some statistical properties concerning a set of selected attributes. As we can see there exists consistency within standard attributes, such as the request attributes, this is indicated by the low

number of distinct values. For obvious reasons not all statistical values can be calculated, for instance the minimum values for string values. However, the table shows some characteristics that could be expected in a web log. The set of requested items is drawn from a specified set of items and could be used to determine what items a user requests from the system. From the web log we were able to extract all instances with a success rate of 100%. This means that the web log provides very consistent information and we do not have to be concerned with loss of information due to consistency issues.

Table 4.16: Data properties of transaction log

Attribute	Min	Max	Mean	Distinct	Null
integer1	1	42	10	41	0%
integer2	1	2866000	460000	85924	0%
integer3	1030	9898	3781	20	0%
integer4	0	142	5	117	71%
integer5	0	99000000000	47600000000	84950	15%

Table 4.16 shows the statistical properties for a selected subset of attributes from the transaction log. The selected attributes are of such private nature that they are masked for the sake of confidentiality. As seen, all attributes contain integer type values and most of them seem to have a defined range of values. The number of unique values also indicates that the attributes are of a less structured nature than the ones seen in the web log. However, some of these attributes have such low numbers of unique values that it indicates a clear structure of the values. From the transaction log, we extracted 72% of the log lines, since the rest of the lines, by manual inspection, were considered to carry no valuable information. Among the extracted lines were the transaction entries, of which we were able to use all.

4.4 User requests

The user request profile searches to analyze the user's interaction with the system at the level of single instances. In our case this means at the level of single transactions and single web page requests. The motivation for this profile is to detect fraud and malicious actions by looking at the requests that differ severely from the normal user requests to the system. These types of actions could be detected by change in the general structure of a request, by strange or false input to attributes, or by values that differ from the normal value range. This profile would need to cope with all these aspects in order to function properly.

The functionality of SVMs and ANNs are thoroughly described in sections 2.4.2 and 2.4.3. The main limitation when using these methods on the described data set is only being able to use numeric and categorical values. Both data sources contain string values which could contain relevant information. In order to preserve some of the information contained in string values we had to transform the information from the string over to numerical values.

The transformation of a string to numerical values can be done in a number of ways. What is of most importance is that the transformation brings the most important information over to the numeric values. Types of transformations have been discussed in the suggested profiles. The transformed numerical values could be string length, character distribution, existence of special characters, or consistency to grammars. We chose to transform the string values over to four numeric values, as listed below.

1. *Length:* the length of the string.
2. *Unique:* the number of unique characters within the string.
3. *Rate:* the rate between the number of unique characters and the length of the string.
4. *Token finder:* the number of characters and words that matches a predetermined list[1].

4.4.1 Support vector machines

The configuration of a SVM consists of three variables that determine different aspects of the calculations for the determination borders. The gamma variable, γ, determines the bound of the radius to the *hyper space* surrounding a class. The epsilon variable, ϵ, determines the tolerance of the termination criterion. In other words, it determines how large error is allowed when specifying the detection border. The nu variable, ν, determines the ratio of points that are allowed to lie inside the class defined as normal[2]. The library

[1]The predetermined list consisted of characters and words that are used to reach backbone systems. It could, for example, be command words searching to execute some backbone command or database control characters searching to alter the effect of a query.

[2]The variable ν is determined by three conditions, as defined in [17]:

(i) ν is an upper bound on the fraction of margin errors, and hence, also on the fraction of training errors.

(ii) ν is a lower bound on the fraction of support vectors.

we used comes with a set of default values for these variables, which we used as an initial configuration. These default values were $\gamma = 0.5$, $\epsilon = 0.001$, and $\nu = 0.5$.

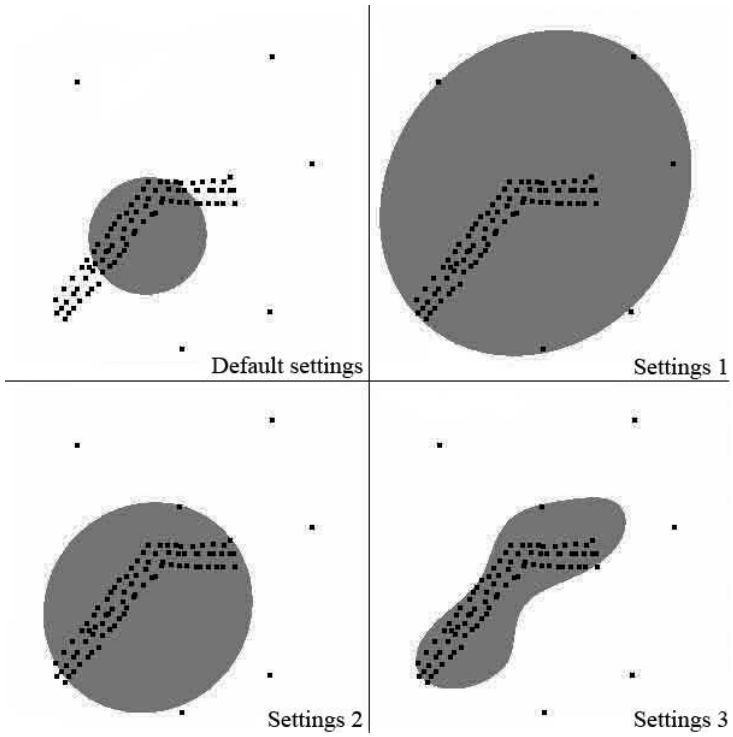

Figure 4.1: SVM's classification space at various configurations.

Figure 4.1 shows how the classification space changes for different configurations, illustrated by a two dimensional space. The first picture shows the default configuration, setting 1 illustrates the configuration with $\gamma = 1$, $\nu = 0.01$, and $\epsilon = 0.01$, setting 2 with $\gamma = 1$, $\nu = 0.1$, and $\epsilon = 0.01$, and setting 3 with $\gamma = 15$, $\nu = 0.1$, and $\epsilon = 0.01$. One can see how the different variables influence the subspace. The following text will describe how these variables are determined in order to gain the optimal classification space for our data.

(iii) Suppose the data $(x_1, y_1), \ldots, (x_m, y_m)$ were generated independently and identically distributed from a distribution $Pr(x, y) = Pr(x)Pr(y|x)$, such that neither $Pr(x, y = 1)$ nor $Pr(x, y = -1)$ contains any discrete component. Suppose, moreover, that the kernel used is analytic and non-constant. With probability 1, asymptotically, ν equals both the fraction of support vectors and the fraction of margin errors.

The best values for these settings are unknown and unique for each data set. The only way to find the best combination is by testing each and everyone, tweaking the system until a satisfactory result is achieved [CWHL]. We found that the best way of doing this was by systematically evaluation different combinations of values. In this way we were able to identify the areas of values which we needed to look closer on. By doing this constriction on the variables in two iterations, we were able to find a configuration that performed well.

Since this is an unsupervised machine learning approach there is no way to measure the detection rate for the different configurations. Hence, a different success criterion was needed. Based on this we started to look at the classification rate of the different configurations. It would seem that one-class classification was not as easy for most of the configurations due to difference in the data. This raised one clear success criterion that needed to be fulfilled. The configuration needed to be able to classify most of the instances within one class. However, it should not classify all instances within the same class. This is because it would lead to too wide boarders to the class and consequently too low anomaly detection.

The evaluation of different configurations was performed by varying one variable at the time to see how the performance consequently altered. In the first iteration we used the default values on the constant values. In the second iteration, we used the values that seemed to perform the best in the first iteration as constant values.

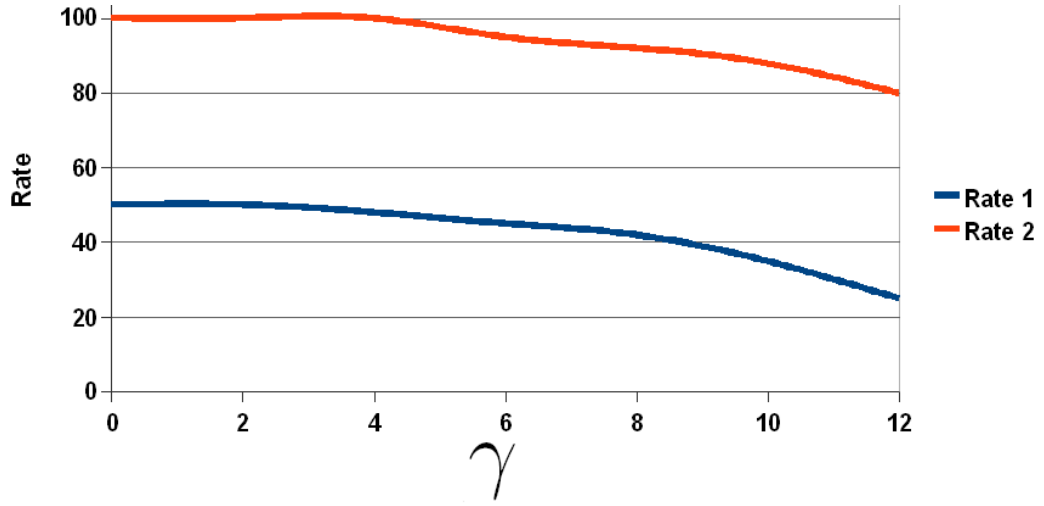

Figure 4.2: Variation of detection rate for SVM due to variations in γ

Figure 4.2 shows the detection rate for different values of γ. The blue

line shows the first iteration with default values on the constant variables, and the red line shows the second iteration, with improved values on the constant variables. One can see how poor the γ performs overall due to poor performance on the default values on the other variables. On the second iterations one can see a clear improvement in classification rate due to the improvement of selecting new values based on the first iteration. One can also see the effect of an increasing values of γ. When increasing the γ values the classification hyperspace is made to fit closer to the data so that less jitter is allowed in the data that is classified. Based on this evaluation we can say that we need to select a γ which fits the data closely, but not so close that it eliminates the possibility to normal variations.

The first iteration shows a tendency of weakened performance for γ values above 3. Based on this observation we chose 3 as the new default for the second iteration. In the second iteration we saw that the tendency was observed for values above 4, so this value was chosen as the default for the rest for this project.

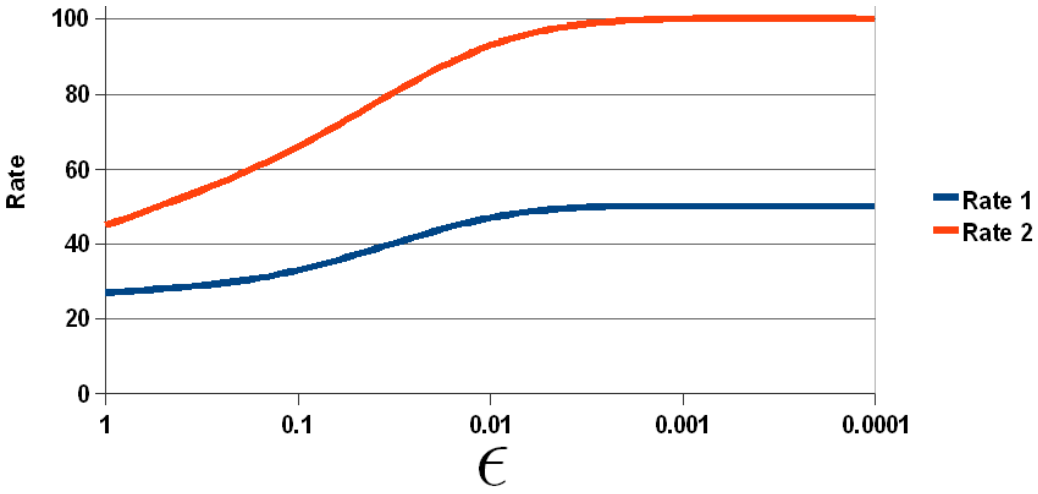

Figure 4.3: Variation of detection rate for SVM due to variations in ϵ

Figure 4.3 shows the effect on the detection rate as ϵ varies. In order to find varying effect with ϵ we had to chose large differences between values. In the figure, the ϵ is reduced to one tenth of the previous value. Now we can see how ϵ influences the classification rate of the SVM. Lower values accept less error in the classification, and hence, are of interest to us. We see that ϵ seems to flatten out in the interval between 0.01 and 0.001. Based on this, ϵ was in both iterations set to 0.001.

Figure 4.4 shows how the detection rate varies for different values of ν.

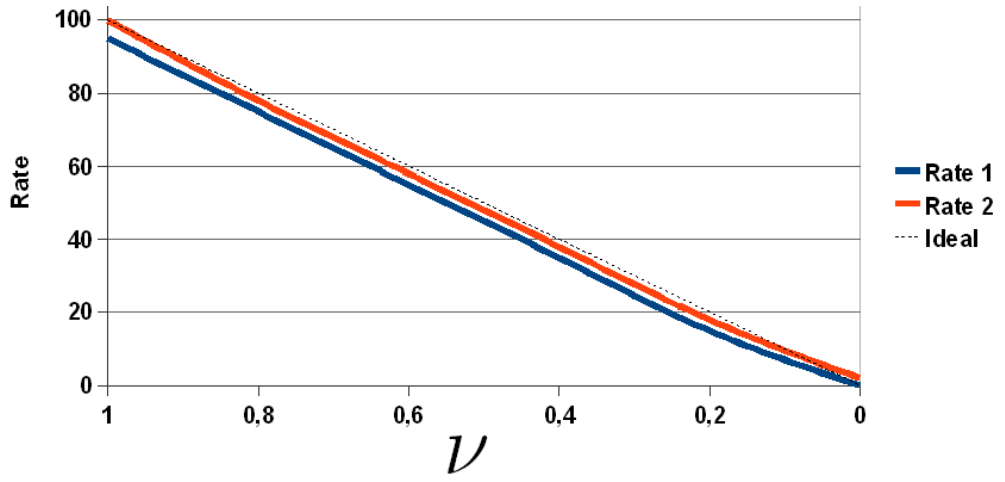

Figure 4.4: Variation of detection rate for SVM due to variations in ν

One can see that it exists a clear correlation between the detection rate and ν. This is because ν defines the rate of instances which should be classified within the class. However, we can also see that the other variables have some effect on the detection rate, but clearly ν is the most significant. This is an important observation since it would allow the model to define its detection rate. It also introduces a challenge since we have no information about the actual rate of malicious instances in the data set. Based on this we have to select a value for ν that is large enough, such that anomalies cannot be classified as normal, and low enough so anomalies are detected at all. In this project we have chosen to use $\nu = 0.01$. However, this value can be changed with ease if it is discovered that another value would be better suited.

Based on this evaluation we can see that some configurations are better than others. The following values were determined to be our standard configuration for the rest of this project: $\gamma = 4$, $\nu = 0.01$, and $\epsilon = 0.001$. This configuration seemed to give a low and accurate detection rate on anomalies of about 1%. However, we kept notice of other configurations that would provide other detection rates. We saw the possibility of being able to set the model at different detection levels could be a valuable feature to the system.

4.4.2 Classification

After we determined a satisfactory configuration to the model we started to test it up against the different data sets. We experimented with different sized training sets and different sized testing sets. The emphasis on this stage of the evaluations was to find the detection rates and anomaly rates in

respect to the sizes of training and testing sets. The goal of this stage was to find a proper trade-off between training set sizes compared to testing set detection rates. It is an inevitable fact that the larger the training set is, the more complex the model becomes. It is also a consequence that the time taken to build the model increases together with the size of the training set.

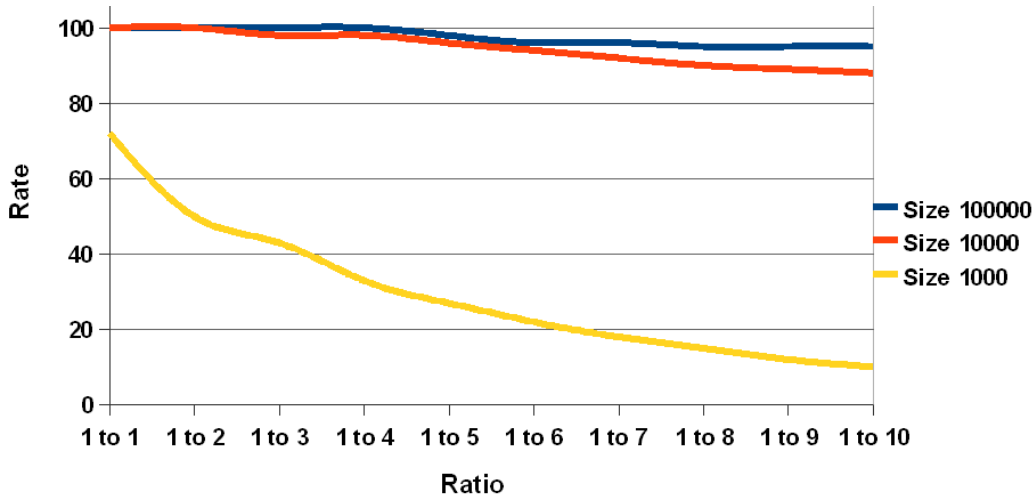

Figure 4.5: Variation in detection rate for SVM due to variations in training to test set ratio, for sizes of training set of 1000, 10000, and 100000

Two important aspects of model training is the ratio between the data used to train the model and the data that is classified. Figure 4.5 shows how the detection rate varies for different ratios and different sizes on the data sets. One can see a slight decrease in performance as the ratio between training set and testing sets increases. However, an even more important discovery is that the loss of performance is less for larger data sets. This would indicate that it is not the ratio between training set and test set that is of importance, but the size of the training set. Figure 4.6 shows the detection rate based on different sizes of training data. It is tested against a large data set, representing several days of log information, consisting of approximately 500000 log instances. This is because we wish for our model to provide proper functionality for a continuously stream of data without the need for continuously maintenance.

Figure 4.6 shows a clear tendency. Too low training sets will result in low detection rates. However, it is clear that the improvement in detection rate flattens as the training sets get larger. This would indicate that one can build a model based on a subset that will perform just as good as the whole data set. The graph seen in the figure exposes that a training set of

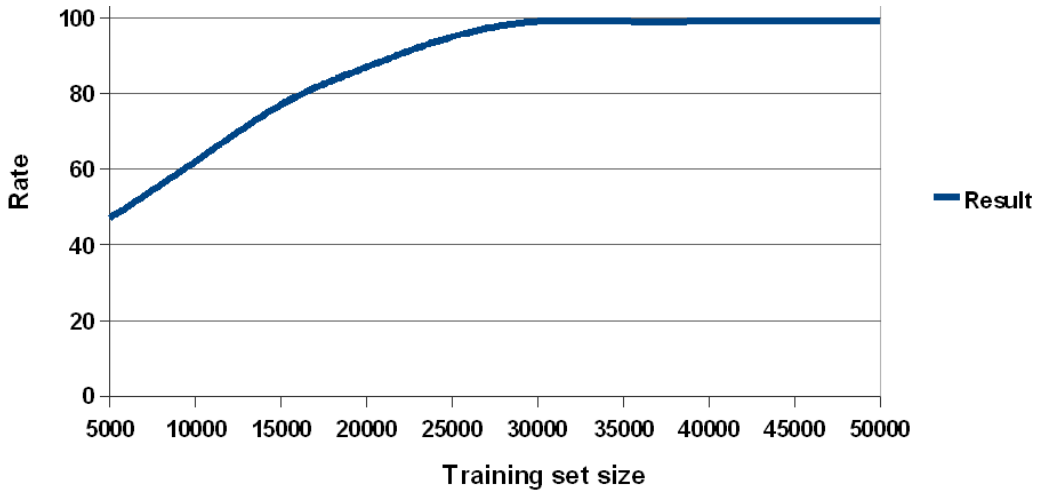

Figure 4.6: Variation of detection rate for SVM due to variations in training set size

approximately 30000 instances would be sufficiently to keep the detection rate at proper levels.

We wished to evaluate the performance of our classification models with actual malicious log entries. By doing this we could get an idea of a very important aspect of anomaly detection, namely if the malicious log entries are detected as anomalies. By doing such a test on real malicious data we could also evaluate the false rate and tolerance of our models.

When testing was about to commence we noticed errors in the log parser due to instance inconsistency. It became clear that some sort of update had changed the log format of the logs making them impossible to read by our log parsers. It would seem like a quick thing to change by reimplementing the log parser, but this would not be possible. The incident logs containing the malicious log entries were of a small quantity consisting of log entries directly invoked or close to a malicious act. The rest of our available data was on a new format, hence we did not have any training data to use for classification on the incident logs.

This was a setback for the evaluation of the detection models, but we found it relevant to perform an alternative evaluation of the detection rate. When actual incident data is not available, it is normal to create synthetic incident data with the known properties and characteristics of fraud. The different threats described in section 3.3 would be a logical place to start to look for different kinds of threats. However, we found that the result provided by such synthetic instances would be very limited. This is due to the fact that we easily could create instances that would and would not be

detected by our models. This is based on the fact that we know what the model looks for in the instances and what would be neglected.

4.4.3 Neural networks

As explained in section 2.4.3, we considered unsupervised neural networks. We did not encounter a suitable implementation of these algorithms, so we had to implement it ourselves. The core elements of the implementation are shown in section C.2.1.

We wanted the net to be able to be trained and used in real-time. This means that data can be given to the network as soon as it is available, instead of assuming that there is a large static data set which the network should be trained on, before the set is classified. This imposed several restrictions on the implementation of the network. Referring to the enumeration of the steps involved in training an unsupervised neural net in section 2.4.3, the following decisions were taken when determining the design of the training phases of the neural net.

Initialize weights Several approaches have been employed when initializing the weight matrix of an unsupervised neural net. In some cases the weight vectors are set equal to some randomly chosen input vectors from the training set. However, we want the net to be prepared to make only one pass through the available data sets, so that not all data needs to be available at the time of initialization. Therefore, the weight matrix is initialized to random values between 0 and 1.

Feed the network with a vector from the training set Also the order in which the input vectors from the training set are presented to the network varies. Since the net needs to be able to train with incrementally available data, this order is sequentially.

Find the BMU The BMU is the output unit which has the weight vector closest to the input vector. The function that determines the distance between the input vectors and the weight vectors is normally the Euclidean distance. This is a good measure of this matter in most cases, and is also implemented in our network.

Find the radius of the neighborhood of the BMU This step is sometimes implemented to make each input vector impose a greater influence on the network. This was not implemented, in accordance with [KHK+96].

Adjust the weight of each node in the neighborhood When the previous step was not implemented, this involves adjusting the weight vector of the BMU alone. This is accomplished by equation 2.19, without Θ, as in [KHK+96], as shown in equation 4.1:

$$\vec{W}(t+1) = \vec{W}(t) + L(t)(\vec{V}(t) - \vec{W}(t)) \qquad t = 1, 2, 3, \ldots \tag{4.1}$$

The learning rate, L, is implemented as in equation 2.21, with an initial value, L_0, of 1. However, the definition of λ, see equation 2.18, is altered. When defined as in equation 4.2, it resulted in a more suitable decrease in the learning rate for our dataset. The value 8 was found to be suitable by experimentation.

$$\lambda = N/8 \tag{4.2}$$

Repeat the procedure from step 2 The number of repetitions of the training cycle was set to be configurable. However, we found that the upper limit of the recommendation from [KHK+96], at 50 - 200 times the number of output nodes, n, was a suitable default value.

We implemented a simple visualizer for the neural network, of which figure 2.21 provides an example. This visualizer was used to interpret the effect of adjusting the number 8, in equation 4.2, the number of training cycles, as well as the initial learning rate, and to determine suitable values for our data.

4.4.4 Classification

We implemented a classification layer on top of the neural network, i.e. a classifier that uses the neural net to classify instances as either target or anomaly. This classifier is configurable with regards to two central attributes:

Number of output nodes The number of output nodes affects the accuracy of the model, and is determined by experimentation. Generally, the more output units used, the better the generated model will fit the training data. However, the point is not to classify the training set, but unknown data. In addition, more output units increase the computational load.

Anomaly threshold This is the threshold that determines whether a given instance is regarded as target or anomaly. This value is the maximum distance an instance can have to its BMU without being classified as an anomaly.

When running the classifier, the input values were normalized to values between 0 and 1. This was done by determining the largest values for each attribute, and dividing each input by the corresponding value. This ensures that attributes with small values has equal impact on the result as attributes with large values.

Suitable values for the number of output nodes and the anomaly threshold were determined by experimentation. The weight matrix was initialized to random values before the training started. To see which effect this had on the classification, the classifier was dry-run several times with the same configuration. It became obvious that this had little effect on the trained model. Nevertheless, we ran the classifier five times with each configuration to discover any deviations from this. The numbers for the following figures were taken to be the best values obtained, meaning the lowest number.

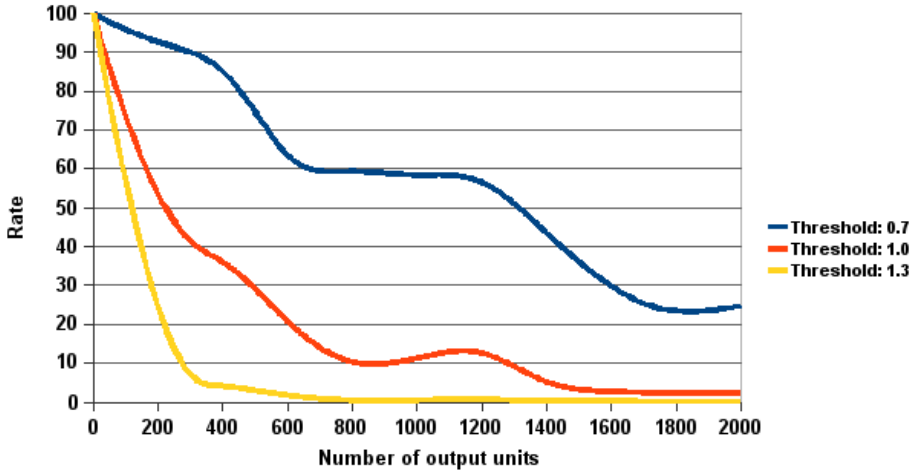

Figure 4.7: Variation in detection rate for ANN due to variation in number of output units, for thresholds 0.7, 1.0, and 1.3 and $N = 200n$.

Figure 4.7 shows the percentage of anomalies for various numbers of output units and three different anomaly thresholds. This figure is generated with a configurations in which the number of training cycles, N, is the number of output nodes, n, multiplied by 200. Obviously, a network with 0 output units does not make sense. In this case, we define the percentage of anomalies to be 100%, since no BMU can be found for any input vector.

We had 525919 instances in the set. To be able to fully train a network with 2000 output units, 400000 instances were set aside for training, leaving 125919 instances in the test set. All models, except the one where $n = 2000$, were thus trained on a subset of the training set.

Generally, this figure confirms that more output nodes mean fewer anomalies. However, in the range around 800 to 1200 there is a slight increase in the number of anomalies, for each of the three classification thresholds. By analyzing the output from the model, it appears is if it, in this interval of N, is more sensitive to the values of the initial weight vectors. This behavior should thus be reduced if the number of times the model is trained is increased. However, the exact values for each configuration were not vital.

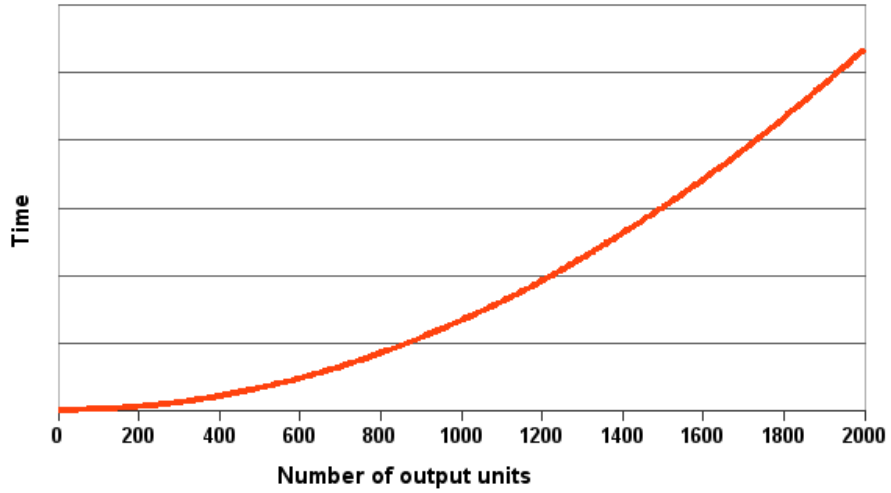

Figure 4.8: Variation in training time for ANN due to variation in number of output units, n, for $N = 200 * n$.

The relative training times for the various numbers of output units are shown in figure 4.8, for $N = 200 * n$. The implemented algorithm is not optimized with regards to performance, nor do we consider the available hardware comparable in this context, so the actual time spent training the model is not relevant. However, the relative training times for models with different number of output units are of interest. Obviously, we would like a model to impose as little as possible load on a system. Evidently, the time taken to train the model increases above linearly for each output unit added to the network. This is as expected, since each new output node causes the model to be trained for further 200 cycles, in addition to that the distance from the added node's weight vector to each input vector must be calculated. This is not a fair measure of the implementation's performance,

since the number of cycles increase for increased number of output units, but it is a relevant measure, since it models the actual time increase needed to successfully train the model.

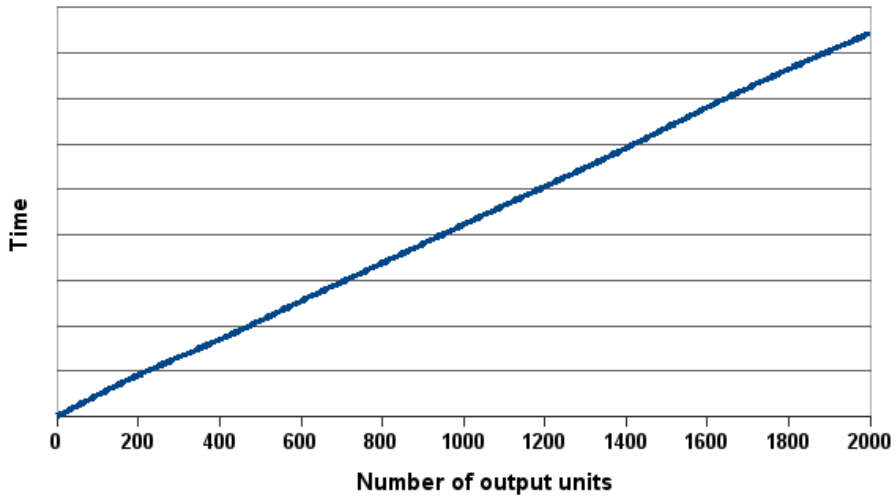

Figure 4.9: Variation in classifying time for ANN due to variation in number of output units, n, for $N = 200 * n$.

As expected, figure 4.9 shows that the time taken to classify the same number of instances with varying number of output units increases linearly. One added output unit demands one more distance calculation for each input vector.

From figure 4.7, it can be seen that the same anomaly rate can be achieved with different configurations. For instance, a configuration with $n = 1800$ and $threshold = 0.7$ generates approximately the same amount of anomalies as a configuration with $n = 600$ and $threshold = 1.0$. There may be differences in which instances are classified as anomalies between the configurations. However, we do not have any foundation to claim that one is more correct than the other, in that the instances classified as anomalies by one configuration are more outlying than the ones classified as anomalies by the other. Thus, the configuration we choose to proceed with may as well be the one with the lower number of output units, since this leads to a model which can be trained faster.

As for SVM, different configurations can be utilized to allow different anomaly detection rates. We chose to proceed with a configuration with the number of output units equal to 600 and the threshold at 1.3. This provides a low and accurate detection rate of just above 1%.

4.5 Session structure

The purpose of the session structure profile is to attempt to create a baseline for normal navigation through the system's resources. In that way, unusual sequences of requests, which may, for instance, indicate attempts to bypass application logic will be classified as anomaly.

To analyze sequences of states, we chose to utilize a Markov chain. A regular Markov chain, i.e. one with a memory of just one state, will fail to consider a great deal of important information, for instance inter-request time delay and other states than the current and next. We believe that these would contribute a great deal to such a model. However, incorporating such information into this profile increases the complexity of the models by a great amount. Therefore, we consider the Markov chain in its simplest form, without any extra information beyond state transitions.

4.5.1 Markov chains

Markov chains are introduced in section 2.4.4. Since we did not discover a suitable implementation of a Markov chain, we implemented one ourselves. Excerpts from this can be found in section C.2.2.

We considered only first order Markov chains. These models are rather simple, and the only variable is the number of training cycles the model should perform. For each cycle, information about the current transition is incorporated into the transition matrix. This is done by maintaining a matrix where the numbers of transitions between all nodes are kept. When a new transition is added, this matrix is updated, and the new probability for this transition can be calculated.

4.5.2 Classification

As with the ANN, we implemented a similar classifier for the Markov chain. This classifier also allows a threshold to be set. This threshold defines the minimum probability a transition needs in order for that transition to be considered normal. If the probability for that transition is below this threshold, it is considered an anomaly.

A Markov chain is a rather general model, in that all it needs are unique names for a set of states, and a set of transitions from one of the states to another. We ran this model on both the transaction logs and the web logs.

A model treating the structure of sessions needs more training data than one looking at single requests. To level with any changes in users' behaviors during different hours of the day, the model was trained with all log data

from an entire day. Due to the simplicity of the model, its design is not evaluated in the same manner as for the previous methods. However, we should still be able to answer the research questions related to the Markov chain. We do this by establishing the transition matrices from whole days of log data and examine the resulting probabilities.

Transaction log

Markov chains needs a set of states and a set of transitions between them. A central decision is what to define as states. In the transaction logs, each request causes both an internal request and response to be manifested in the log. The requests are caused by some function being invoked. The name of this function is present in the logs, and so defining the states as the names of these functions seems as a reasonable choice.

One days worth of log data from the transaction log, parsed and suitable for the Markov chain, means about 3700000 instances.

Web log

When running the model on the web logs, we simply defined the states to be the requested resources. This means that any request made by the user will be treated by the Markov chain. It also means that the Markov chain has to handle a relatively large number of states and state transitions. To keep the number of states and transitions to a more manageable level, we exclude requests for graphical elements in this situation. Another simplification could be to take certain irrelevant web pages out of consideration. However, we do not have enough insight into the banking application to decide which pages may be excluded. Therefore, all pages are considered. Any parameters were removed, as different values for these were not considered to be separate states. As with the transaction logs, the model were trained on data from one day, in this case about 8200000 instances.

4.6 Summary

As seen in this chapter, we proposed six different profiles that could be used to detect fraud and malicious actions. Based on an evaluation of these profiles we decided to limit the set of profiles and focus on two profiles, of which one of them was a combination of two original ones. The profiles we chose to work further with were the user request profile and the session structure profile.

Data preparation came as a logical next step. We needed to prepare the available data material so that it could be used by our selected machine

learning methods. This consisted in rearranging the data instances to a common and consistent format, together with transformation or extraction of unusable data types to a better suited type.

The user requests profile looks at the data at a single instance at the time. The emphasis is on attribute values and instance structure. This profile was implemented using two different machine learning methods, SVM and ANN. The behaviors for both of these were similar and positive. Both machine learning methods were able to detect anomalies at different thresholds in the data set.

The session structure profile treats the data at a higher level. It considers sequences of interactions between the user and the application. However, to limit the complexity, only sequences of two interactions were considered. A Markov chain was used to establish a model of normal sequences.

Chapter 5

Results and analysis

Summary

This chapter consists of the results provided by our implemented profiles, based on the different machine learning method, and used on the available data logs. These results are analyzed in order to provide insight into and understanding of their meaning and impact. We look at how the results and information discovered by the analysis answers the research questions. Additionally, we analyze how the validity of the results is maintained, considering the relevant threats to validity.

5.1 Profiles

After implementing the profiles we can evaluate their performance in terms of how well they detected anomalies. The following analyzes the results based on the two profiles which were implemented. The user requests and session structure profiles were implemented using different data sources and machine learning methods. These aspects, concerning data source and machine learning methods, will be analyzed later in this chapter, but a more formal analysis of the profiles is conducted here. Especially, we wish to look at the properties of the logs compared to the properties captured by the profiles.

5.1.1 User requests

The theoretical foundation for the user requests profile, described in section 4.1, and the described threats to the system, in section 2.1.3, provided a basis for this profile. An important foundation was that malicious behavior would manifest itself in data types, value ranges, and characters present for different attributes provided by the user. The user requests profile would detect activity that falls under this category, however the results provide no indication that the analyzed data set contained such activity. This is probably due to other security mechanisms detecting such attacks and limits these kinds of interaction from reaching the log.

The most apparent property of the user requests profile is both a strength and weakness, namely that this profile looks at the data at a very high level of detail, and is limited to only look at one instance at the time. This results in two things. Firstly, the user requests profile is able to analyze and evaluate each instance with high accuracy, since we look at a very limited set of data at the time. This makes room for special analysis of attribute values, structure and other types of specialized analysis of each instance or value. Secondly, due to the fact that this profile only looks at one instance at the time it loses a valuable information source, namely the context in which the instance appears. Obviously, we can think of many types of fraud that can only be detected with an analysis that includes the context surrounding the action. This is the main limitation of the user requests profile.

5.1.2 Session structure

The theoretical foundation for the session structure profile was defined in section 4.1. The key aspect for the session structure profile is that we are able to create a model for transitions between different states in the system. When the states and the transitions between them can be modeled we can

calculate the possibility for the different state transitions. The potential of this profile is large since there is a large amount of information that can be taken into account. The basic approach can learn how a user traverses through the system's web interface, detecting strange and illegal transitions. More complex approaches can take sequences of transitions and time between transitions into the model.

An apparent property of the session structure profile is the complexity of it. If one considers the model to consist of n states, the model would then have to provide an nxn matrix just for the basic state transitions. If the model should include more information one can see that the growth of complexity would be considerable.

The most interesting aspect of the session structure profile is that it can potentially look at the complete interaction between user and system, i.e. from login to logout. This creates a total picture of the user's behavior that leads to detection based on a general behavior rather than attribute values.

5.1.3 Research questions related to profiles

RQ 4.1: *Can malicious behavior be detected by the use of a profile looking at each user request?* In theory, this profile would be able to detect all kinds of attacks which would require changes in attributes' values. Many kinds of attacks would have such properties. However, it seems to be a valid observation that security measures already operational in the system provides a similar functionality. The tests performed on the user request profile do not give any indication that it can detect malicious behavior outside the scope of anomalous attribute values.

RQ 4.2: *Can malicious behavior be detected by the use of a profile looking at sequences of requests?* It is known that several attacks must exhibit properties that would be detected by such a profile. These include attacks that search to bypass application logic by altering normal sequences of requests. A central issue is how much information to include in such a profile. Increasing the model's complexity may lead to increased detections. In any case, theoretically, such a profile could detect malicious behavior.

5.2 Machine learning methods

This project has used three different machine learning methods to achieve its results. SVMs and ANNs were used to create the *user requests* profile and

Markov chains were used to create the *session structure* profile. Following is an analysis of the results provided by the different methods.

5.2.1 Results and analysis of SVM

The results provided by the SVM implementation of the *User requests* profile showed that it was possible to create a classification model that could classify new instances as normal or abnormal. In this analysis we search to find how the SVM classifies the instances and on what basis. We also look at the validity of the results as discussed in section 3.3 and how the results answer the research questions presented in section 3.1.

Classification by SVM

An overview of how the anomalies are distributed into the normal data can be seen in the plot diagram in figure 5.1. It shows different attribute data spaces, where blue represents instances classified as normal and red instances classified as abnormal. The figure shows that some dimensions provide better separation between normal and abnormal instances. One can also see that some of the dimensions provide a clearer structure of normal behavior and a higher similarity between normal instances.

Figure 5.2 shows a more detailed graph of two of the plot graphs shown in figure 5.1. Here, we can see that the instances form a pattern of value ranges that define the normal behavior. These collections of blue data points within the same area, clusters, are an indication that the values within these clusters are legal. Further we see that a few anomalies, represented as red data points, exist in the higher data range of the horizontal axis. It is also important to notice the anomalies that have fallen within the areas that we have described as legal behavior clusters. This is due to the fact that anomalies are detected in all dimensions, not just the two dimensions shown in this graph.

The plot graph shown in figure 5.3 displays a similar picture of data clusters as in figure 5.2. However, there is a significant difference. This difference lies in the vague cluster shown in the middle of the horizontal axis and bottom of the vertical axis. This cluster is distinctive enough to have normal instances, but vague enough to contain anomalies as well. By analyzing this picture manually it would be tempting to look at the cluster as an anomalous cluster. However, the SVM classification has defined this cluster as normal. This can be seen since there are normal instances in the cluster, and hence the border for normal classification lies further out on the axis. The anomalies in this cluster are detected by deviations in other

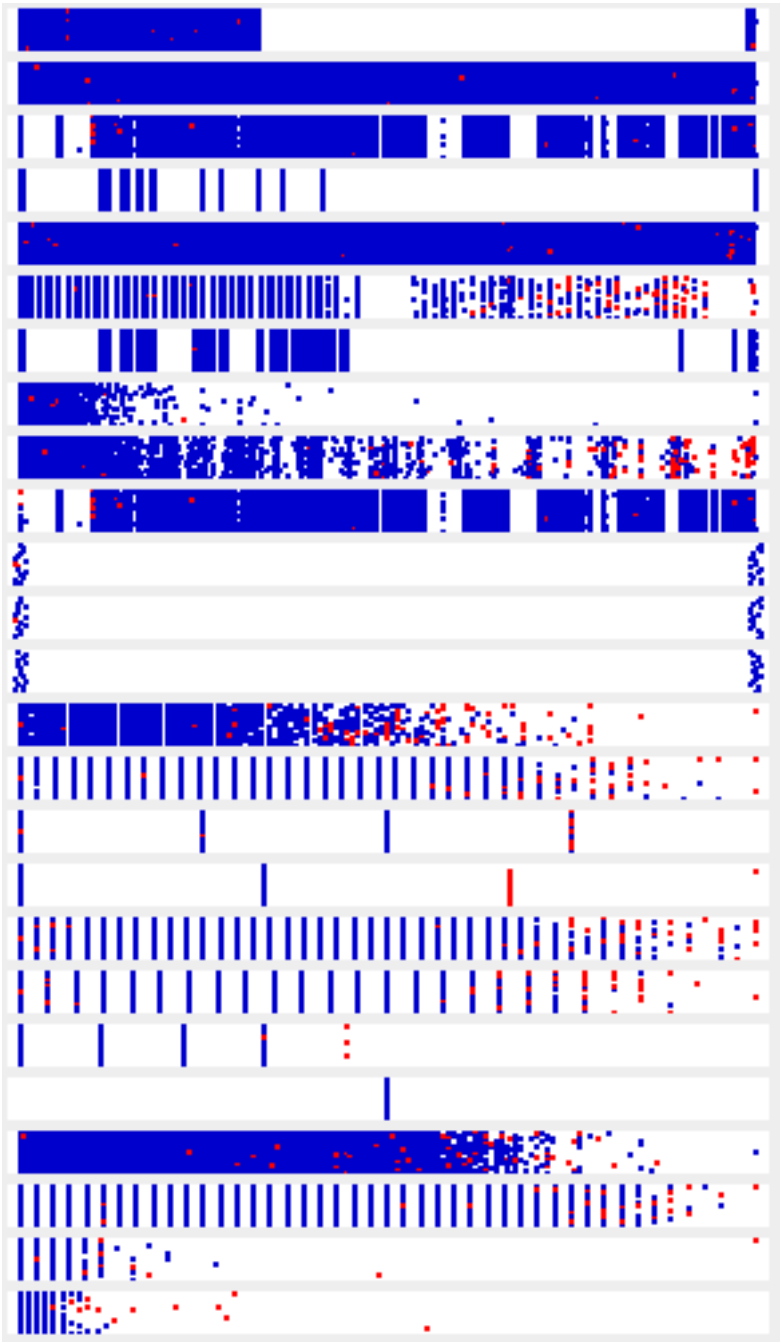

Figure 5.1: Classification overview for SVM

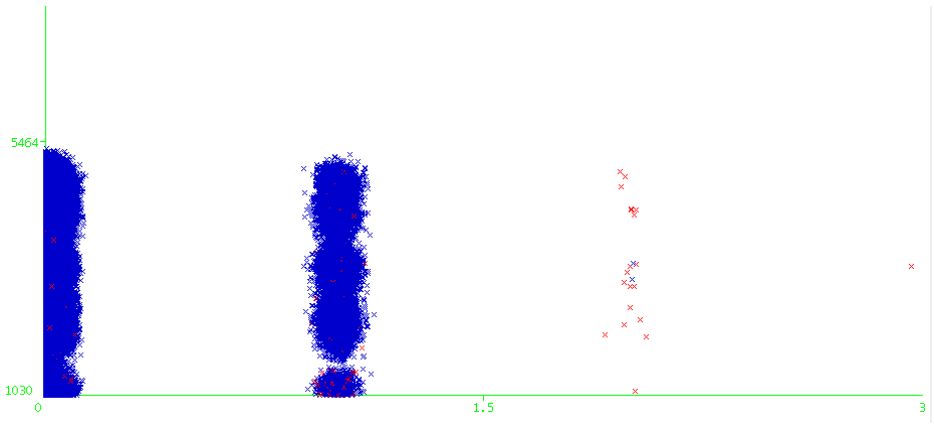

Figure 5.2: Classification plot for SVM

dimensions than the two shown here. A valuable lesson can be learned from this graph. Anomalies seem to deviate more from the norm in general than the normal instances, also within dimensions other than where it is classified as abnormal.

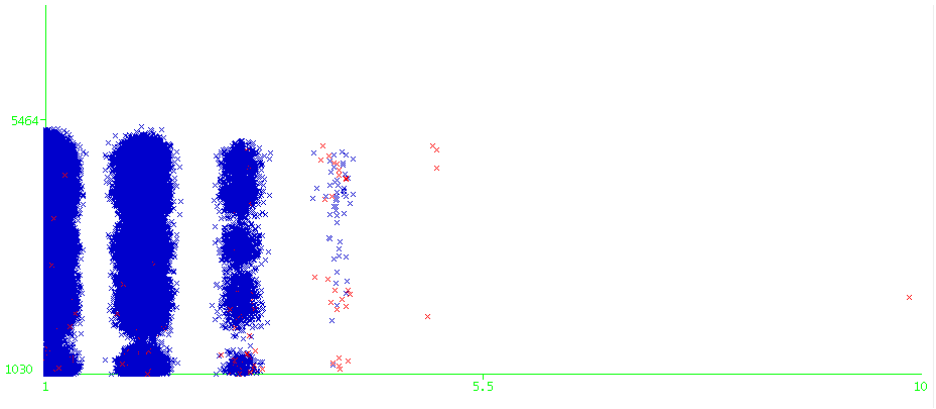

Figure 5.3: Classification plot for SVM

Figure 5.4 presents a quite different data distribution than the previous illustrations. Here, it can be seen that the data distribution is more normalized than in the other plots. No clear border of normal behavior can be found in the horizontal axis. The vertical axis has, however, a clear abnormal subspace in the middle of the axis. However, in this case on anomalies lies in this subspace. This plot is a good example of that not all dimensions provide clustering between each other. As this case the data does not seem very well structured and the anomalies are distributed equally among the normal instances.

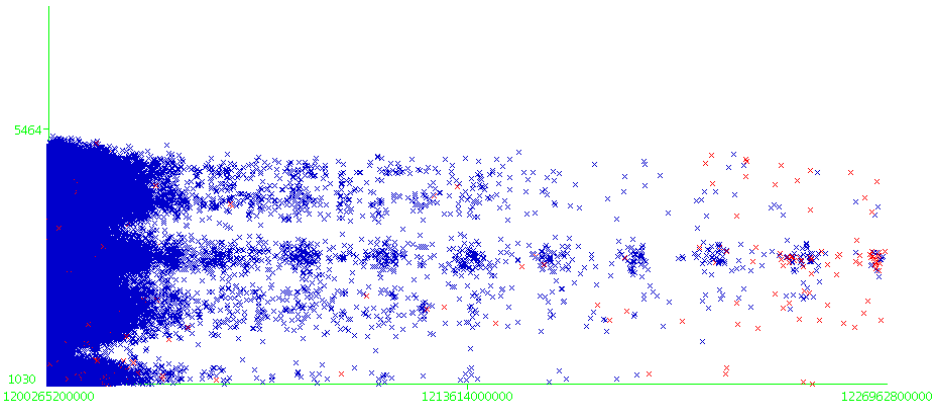

Figure 5.4: Classification plot for SVM

The performance of classification by the SVM model, together with the analysis of the detection level provided in this chapter, shows that the data set contains both clear clusters of normal behavior and correctly classified anomalies. We see that while some dimensions show high density clusters made up by instances classified as normal, other dimensions show little to none patterns of behavior or clustering. A general observation from the graphs is that instances classified as anomalies have a tendency to lie close to the detection perimeter of other attribute dimensions as well, besides the ones that are breached.

Research questions related to SVM

An important part of the analysis is to review how the results has provided answers to the research questions suggested earlier in this report. In the following we will analyze how the SVM model has provided answers to the relevant research questions. It should be clear that the SVM can only provide answers to the questions directly related to it.

RQ 1.2: *Is it possible to separate normal and abnormal behavior in audit logs based on SVMs?* The results provided by the SVM leaves little doubt that the SVM is capable of separating the abnormal instances from the normal. The further analysis of the results proofs that the SVM is actually separating out the most abnormal instances from the data set.

RQ 2.1.1: *Can research question 2.1 be answered by the use of SVMs?* The transaction log contains actually all the information one would need to document fraud. The problem is however to separate the legal actions

from the illegal ones. This is an important question, whether fraud can be detected in an environment where all types of transactions are legal. Even if a transaction to a foreign account, involving the total balance of an account, is fraudulent in most of the cases, it is still a legal action for any user to do. Based on the results provided by the SVM on the transaction logs, we see that the SVM finds anomalies that are rare transactions. So the answer to research question 2.1.1 is no, SVMs is not able to detect fraud. SVMs are, on the other hand, able to find rare transactions that could indicate fraud, or at least transactions that have a higher possibility of being fraudulent.

RQ 2.2.1: *Can research question 2.2 be answered by the use of SVMs?* See answer for research question 2.2 in section 5.4.2.

RQ 2.3.1: *Can research question 2.3 be answered by the use of SVM?* This is a more problematic question to answer. As we analyzed the different kinds of threats to an Internet banking system, there is little doubt that many types of attacks will manifests themselves in the web log, and would have properties that a SVM model would detect. The typical injection attacks and traversal attacks contain properties that make them distinguishable from the normal. However, it should be noticed that these are not a complete representation of all types of malicious behavior. The Trojan scenario, described in section 1.2.3, does not consistently deviate from the normal usage patterns, and hence, would these types of fraud attempts not be detected by a SVM model alone. Thus, the answer to this question is positive, even though SVM is unable to detect all types of attacks.

RQ 3.1: *Will research question 3 be true for SVMs?* We see that the SVM is able to classify abnormal instances correctly. Hence, it would be possible to use SVM to detect fraud and malicious behavior. The work conducted on SVMs shows that the limitations is not due to the properties of the SVM, but rather due to the data evaluated by the models. SVMs should be able to detect some sort of attacks, fraud, and malicious behavior, but it would not classify all types. This is due to the fact that SVM only looks at a subset of the available data and only analyzes this subset based on a limited set of properties.

5.2.2 Results and analysis of ANN

The ANN is capable of separating abnormal instances from normal ones. However, it is not clear on what foundation the ANN decides which instances

to consider abnormal. This section looks deeper into the results provided by the ANN.

Classification by ANN

As for SVM, figure 5.5 shows an overview of the classification performed by the ANN. The various attributes involved in the classification are stacked on top of each other, with each attribute's distribution of values illustrated horizontally. The figure shows how the anomalies, the red dots, are distributed among the normal instances, the blue dots. Within some attributes, the anomalies are in the midst of all other normal instances, while in other, the red dots have considerably different values than the cluster of blue dots. Also evident in this figure is that many normal instances appear far more abnormal than the anomalies within a single attribute. These observations confirm that the ANN does not classify based on values of single attributes, but rather on a combination of all attributes.

Figure 5.6 visualizes two of the attributes from the previous figures, one is mapped along the X-axis, and one along the Y-axis, and the crosses are then where one of the values intersects with the other. Again, in this and the proceeding figures, the anomalies are indicated by a red cross and the normal instances by a blue cross.

It is not possible to see a clear classification of the abnormal and normal instances when considering these two attributes. The sheer majority of the instances have at least either X-value or Y-value close to zero. The abnormal entries seem to loosely follow the same distribution as the normal ones. Keep in mind that only close to 1% of the entries are classified as anomalies.

Some pairs of attributes exhibit a more obvious clustering of their values, as figure 5.7 illustrates. This relates to the nature of the attributes' values. Again, the distribution of the abnormal entries seems similar to that of normal ones. At the far right, there is a small cluster of entries in which, oddly enough, most is considered normal by the ANN. In contrast, in the upper left corner, there is an even smaller gathering of entries, in which most is classified as anomalies. This appears as a more reasonable case.

In the final illustration, in figure 5.8, there seems to be a more logical distribution of the anomalies compared to the normal entries, in that they are farther away from the main groupings. There are some entries to the far right, which mostly are considered anomalies. Further there is some clustering of the entries. Among the instances which are the farthest from the cluster's centers, a relatively high number of these are considered anomalies by the ANN.

This analysis clearly shows that, generally, no attributes alone, nor only

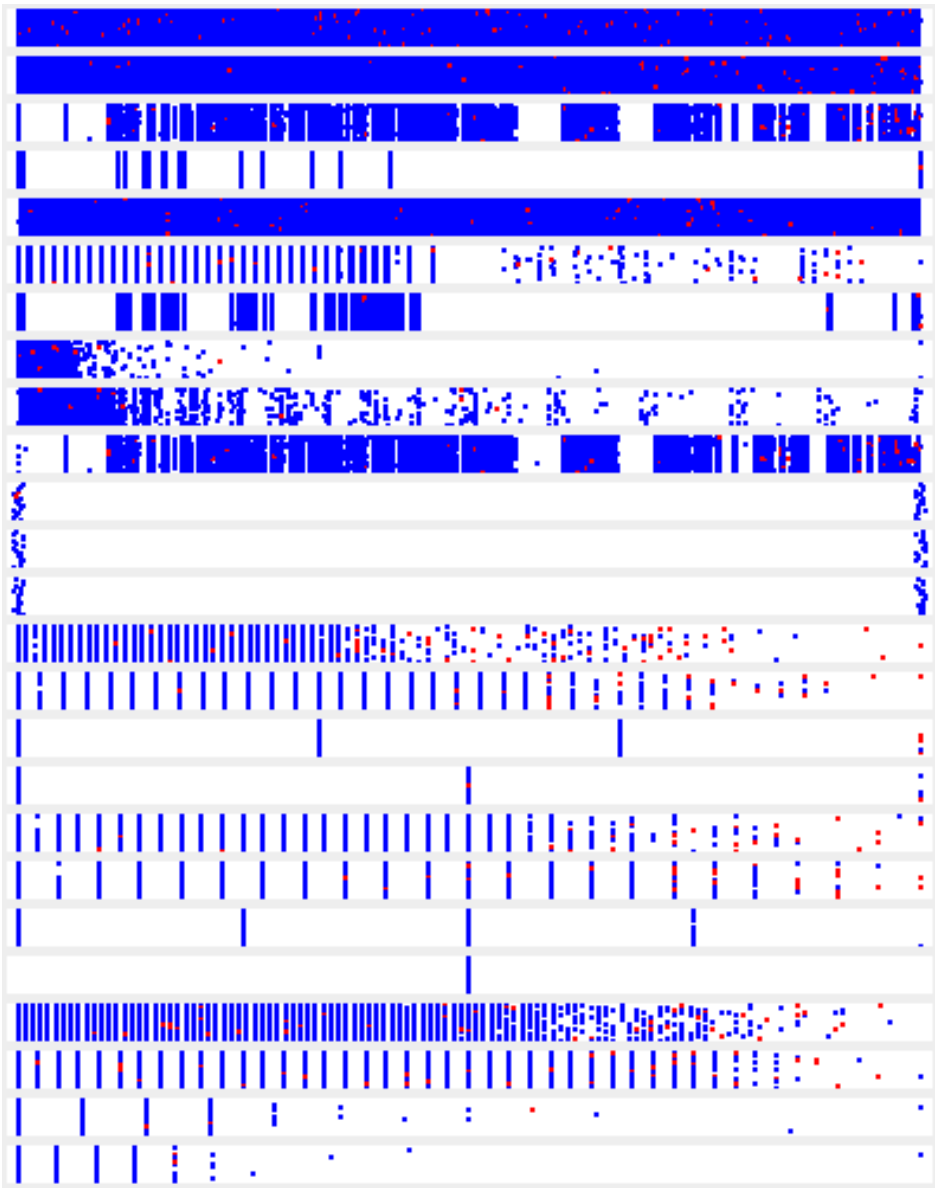

Figure 5.5: Classification overview for ANN

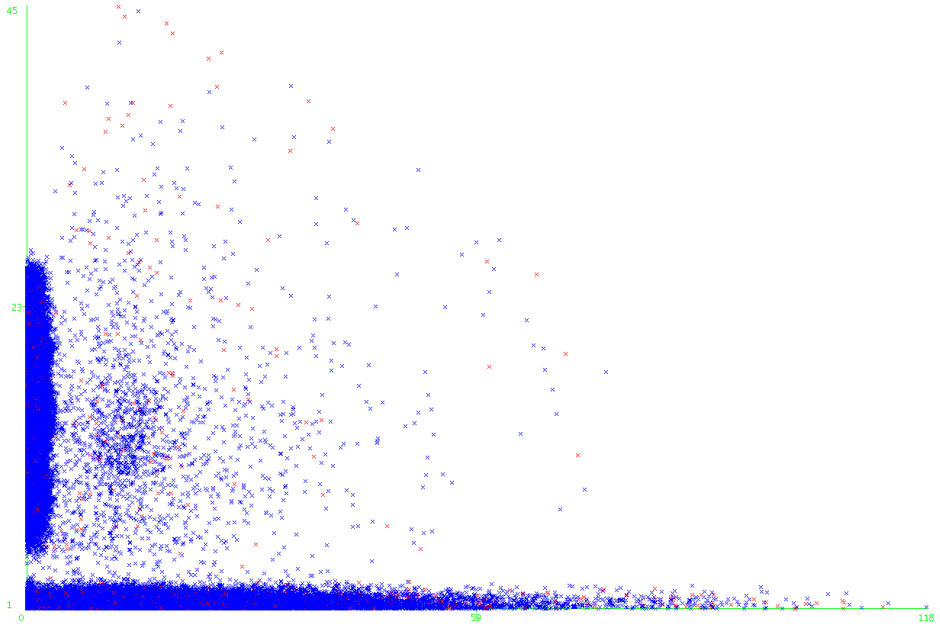

Figure 5.6: Classification plot for ANN

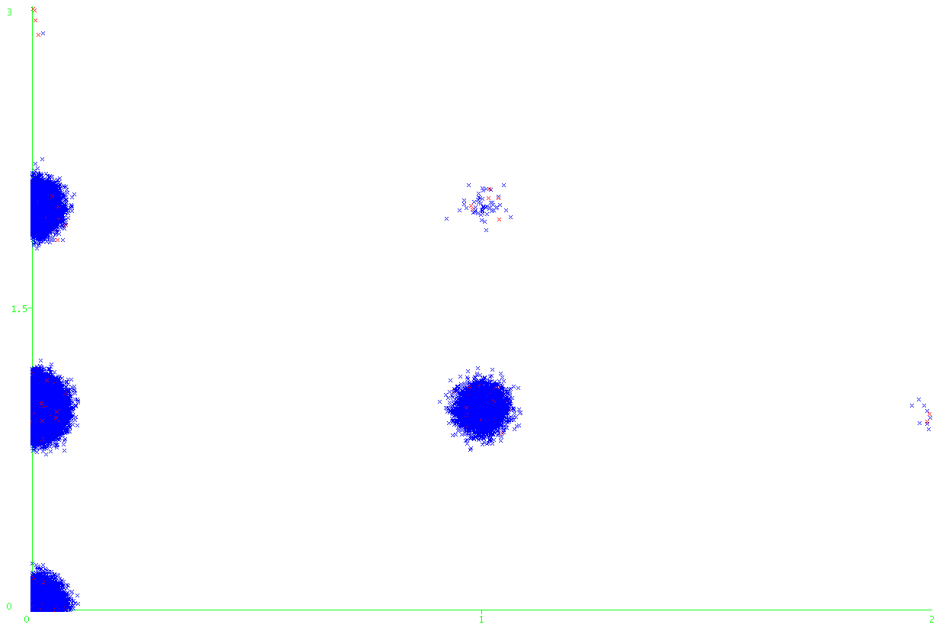

Figure 5.7: Classification plot for ANN

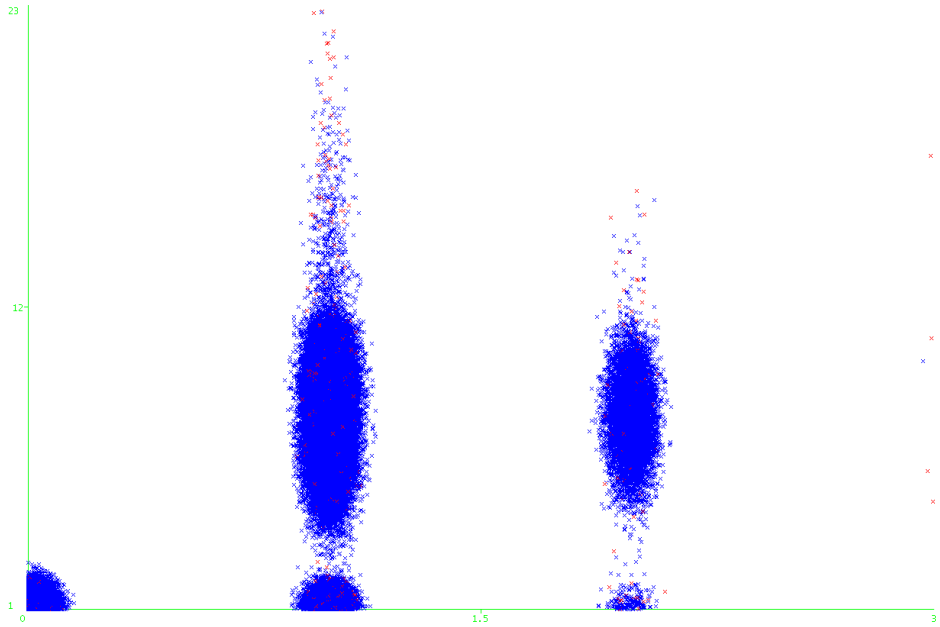

Figure 5.8: Classification plot for ANN

two attributes, are decisive to cause the ANN to classify an instance as abnormal. Rather, it proofs that the ANN takes all attributes into consideration when deciding which instances are anomalies. Attributes with only one or two abnormal values are classified as normal, as long as the rest of the values of this instance's attributes are normal. This is to be expected, considering the distance function of the implemented ANN, which provides the measure of similarity. The more attributes included in the classification, the less influence each attribute gets. It may appear that the ANN considers the collections of all values as a group to a greater extent than the SVM. These observations imply that a proper attribute evaluation and selection might be beneficial for the performance of the ANN, since we cannot expect that malicious instances deviate from the normal ones in every attribute. It is also possible to give some of the attributes a higher weight than others, simply by adjusting the factors used for normalization. This can only be done after a thorough analysis and evaluation of the various attributes' importance.

Research questions related to ANN

Here, we attempt to answer the research questions related to the ANN, based on the results obtained. Since the ANN and the SVM both implement the same profile, the answers are similar to those of the SVM.

RQ 1.3: *Is it possible to separate normal and abnormal behavior in audit logs based on ANNs?* It appears as if what is considered abnormal by the ANN differs from what is considered abnormal by the SVM. The ANN considers the whole set of attributes together, to a greater extent, than the SVM, thereby classifying entries that are abnormal in a few dimensions as normal, as long as the rest of their attributes are normal. Still, the ones classified as anomalies are, in fact, anomalies, in terms of the ANN. We cannot say which classifies more correctly than the other. In any case, the ANN is capable of separating abnormal instances from the normal.

RQ 2.1.2: *Can research question 2.1 be answered by the use of ANNs?* Similarly to the answer of RQ 2.1.1, ANNs cannot determine which transactions are fraudulent and which are not, since any transaction may, or may not, be illegal. They are still able to find rare entries, which often is the case for fraudulent transactions.

RQ 2.2.2: *Can research question 2.2 be answered by the use of ANNs?* See answer for research question 2.2 in section 5.4.2.

RQ 2.3.2: *Can research question 2.3 be answered by the use of ANNs?* Some fraudulent activities leaves traces in the web log which are detectable by the ANN. Considering some of the known attacks and their characteristics, one observation is that the SVM might be better suited in this matter, since it is more sensitive to a subset of the attributes. Then again, a more adapted ANN, better tuned with regards to which attributes to consider and which weight to assign to them, might provide other results. Still, the ANN cannot tell which entries are fraudulent, and which are not.

RQ 3.2: *Will research question 3 be true for ANNs?* No, ANNs alone will in many cases fail to classify malicious activity and fraud as abnormal. Several known attacks and fraudulent activities lack the properties to make them be classified as abnormal by the ANN.

5.2.3 Results and analysis of Markov chains

We successfully implemented and trained a Markov chain on both the web logs and the transaction logs. Hence, a transition matrix was created for the state transitions for both these logs. In the web log, these transitions correspond to resource requests, which in most cases mean page requests. In the

transition logs, they represent invoked functionality based on the requested resources.

Classification by Markov chains

Figures 5.9, 5.10, and 5.11 illustrate the Markov chain, trained on data from the transaction logs, in the form of graphs. The numbered nodes represent states, and the transitions are represented by lines from one node to another. In reality, these transitions are directed, but this is not illustrated in the figures. They are divided into three figures for clarity.

Figure 5.9 shows the transitions with a probability of 1.0. For instance, all processes which entered state 39 during this specific day also entered state 57 through state 48.

In contrast, figure 5.10 shows all transitions for which the probability is low. In this case, the upper limit for considering a probability for low is 0.00001. Evidently, many transitions fall into this category.

For completeness, figure 5.11 shows all transitions not already illustrated in the previous figures, i.e. transitions for which the probability is between 0.00001 and 1. Their relative probabilities are indicated by the thickness and darkness of the lines, where darker and thicker line means larger probability.

We also present similar illustrations of the Markov chain trained on the web logs. However, considering their limitation in additional value, they are placed in section D.1. The upper limit for the transition probabilities to consider low is lower in the case of the transaction log, than of the web log. Obviously, this results in fewer transitions falling into this category, and illustrates that this limit can be varied to adjust the threshold for which transitions should be considered abnormal.

When defining the states as function names in the transaction logs, rather than the requested resource from the web logs, it resulted in a more lucid figure. However, the two models show the same result. A simple Markov chain, without memory, or other extra information, does not discover normal paths taken by the users. However, transitions made by very few of the users are easily uncovered. The limit for which the transition probabilities are considered low may vary, thus providing the possibility to alter it to evaluate different configurations.

Research questions related to Markov chains

This section describes the answers to the research questions provided by the Markov chain.

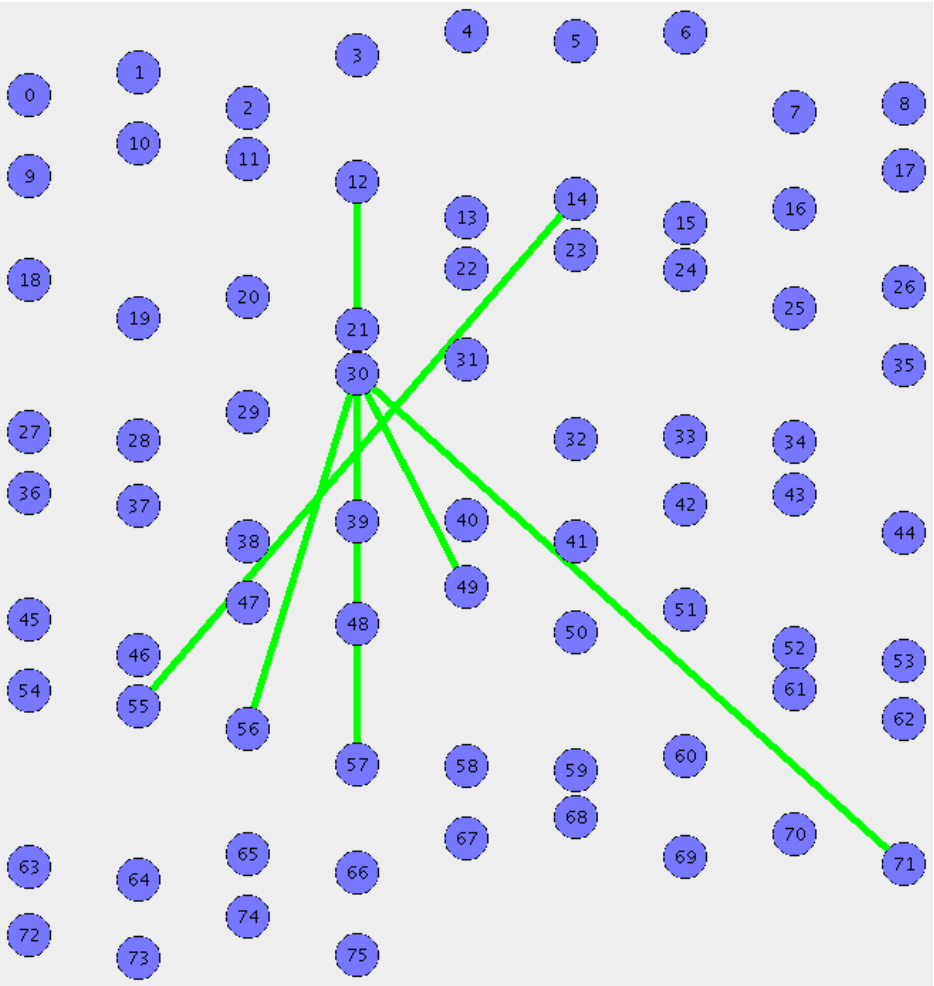

Figure 5.9: Markov chain with high probability transitions trained on transaction logs

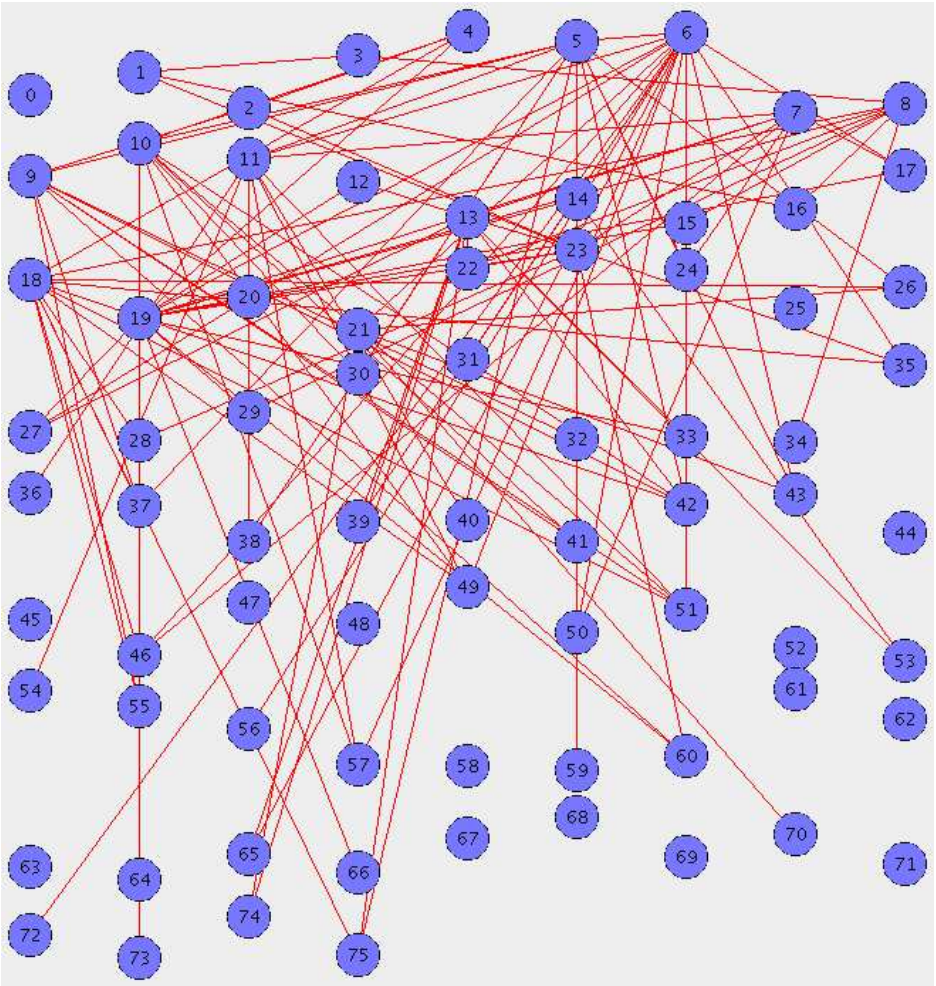

Figure 5.10: Markov chain with low probability transitions trained on transaction logs

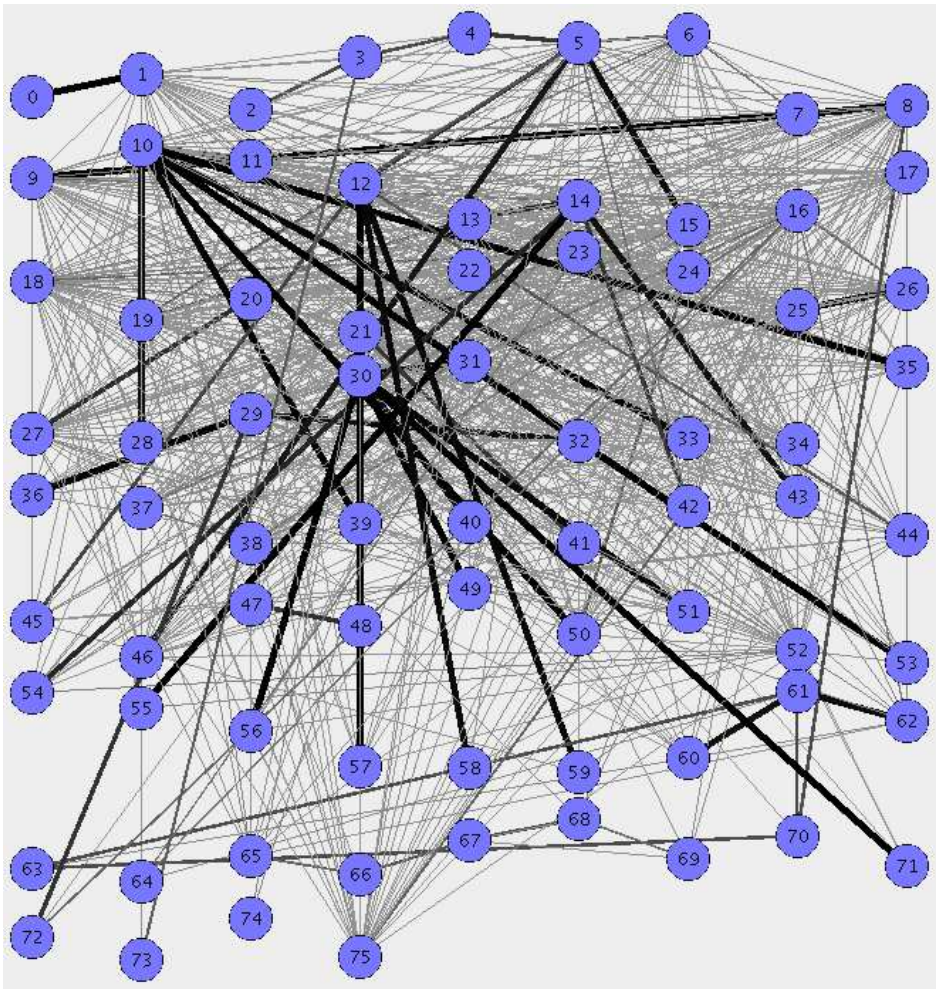

Figure 5.11: Markov chain with medium probability transitions trained on transaction logs

RQ 1.4: *Is it possible to separate normal and abnormal behavior in audit logs based on Markov chains?* Obviously, it is possible to divide sequences of states based on their relative frequency. The Markov chain used in this project only considers sequences of two states. Even so, many transitions could be classified as abnormal.

RQ 2.1.3: *Can research question 2.1 be answered by the use of Markov chains?* Figures 5.9, 5.10, and 5.11 show that Markov chains are capable of separating transitions with low probability from the rest. When gathering data from the transaction logs, rather than the web logs, we would assume that users' requests are interpreted and refined to some degree. This means that innocent transitions taken from the web logs, and considered abnormal, in many cases will generate a transition in the transaction log, which a Markov chain will consider to be normal. This is confirmed by the fact that the Markov chain encountered a lower number of states in the transaction logs, than in the web logs. However, users who, for instance, are testing for flaws in the application, would generate exceptions and function invocations not caused by the average user. Therefore, while a Markov chain cannot detect malicious transitions in the transaction logs, we expect an abnormal transition in the transaction logs to have a higher probability of being truly abnormal, than in the case of the web logs.

RQ 2.2.3: *Can research question 2.2 be answered by the user of Markov chains?* See answer for research question 2.2 in section 5.4.2.

RQ 2.3.3: *Can research question 2.3 be answered by the use of Markov chains?* As evident in figures D.1, D.2, and D.3, Markov chains can be used to separate abnormal and normal requests for resources. However, any user is of course free to enter any address into his browser's address bar. This includes fictional resources located at the server banking applications server. By doing this any user can issue a request for a unique resource at the server, i.e. a resource never requested before, and thereby generate a low-probability transition. Further, there is no doubt that fraudulent requests may be classified as normal by a Markov chain. Consider, for instance, the masquerader scenario in section 1.2.3. It should be obvious that a masquerader may issue a request for a payment by performing the same state transitions as a legal user, thereby giving a Markov chain no reason to classify the session as abnormal. However, malicious users conducting, for instance, guessing for predictable resource locations or page traversals, in order to document and test the system, would most likely generate some requests

classified as abnormal by the Markov chain. Therefore, Markov chains cannot detect fraud directly, but many fraudulent transitions would be detected by Markov chains.

RQ 2.4.1: *Can research question 2.4 be answered by the use of Markov chains?* See answer for research question 2.4 in section 5.4.2.

RQ 3.3: *Will research question 3 be true for Markov chains?* As discussed for RQ 2.3.3, the answer is no. In the case of a Markov chain, with no more information about the processes than that of the current and next states, many fraudulent transitions would pass as normal.

5.2.4 Analysis of validity for machine learning methods

With regards to the threats to validity discussed in section 3.3, we analyze the threats relevant to the results provided by our models to see if the validity of these results is threatened. As mentioned in section 3.3, there are two aspects to consider when evaluating the validity of results. First is the validity of the results within the population it is evaluated by, and second is the validity of generalizing the results to be valid within a larger population. In the following text, we search to find the validity of the results within both these populations based on an analysis of conclusion validity, internal validity, construct validity, and external validity.

The conclusion validity is concerned with the relationship between the detection models and the classification of instances. It consists of three relevant threats to the results of our models:

Low statistical power: We are aware of the potential threats of validity based on the high rate of false positives. The more important threat is the rate of false negatives. This would give the result a low statistical power. The important factor here is the difference between the classification of instances by our models, and the actual classes of instances. However, when the actual classes of instances are not available, this factor is impossible to review. It is also important to notice the difference between abnormal and malicious instances. There will always be a strong possibility for normal instances being classified as anomalies. Hence the rate of false positives is not a relevant measure.

Violated assumptions of statistical tests: SVMs and ANNs have some concrete demands on the properties of the input data in order to provide correct results. These requirements have been followed strictly in order to minimize the risk of lowering the validity of the result.

Fishing: One of the main advantages with machine learning methods, such as the ones employed in this project, is that they leave all the evaluation and processing of data to an objective machine. Hence, the opportunity for the participants of the project to influence the results is small. However, the participants can influence the results indirectly by model configuration, but this can be described more as optimizing the model than fishing for results.

Internal validity is concerned with the quality and properties of the data, and it is hence of little relevance to analyze within the scope of machine learning methods. The internal validity is considered in the analysis of the audit data in section 5.3. Construct validity treats the relation between theory and observation, and is discussed next.

Mono-method bias: The mono-method bias is prevented by two things. Firstly the relevant models use all parts of the available data. This leads to a high dimensional detection space which would inhibit any single attribute from corrupting the model. Second is the fact that the available data is provided over a large quantity of time, which would inhibit any non-consistent data bias from becoming apparent.

Experimenter expectancies: The expectancies of the people conducting the tests have little influence on the results. The only way to influence the results would be to actively alter attributes of the data sets or the classification models. We rely on the seriousness of the conduction of the tests, and hence, assume that only unintentional actions could be conducted, and as explained, there is risk of an unintentional action threatening the validity of the results is small.

The threats concerning external validity are concerned with the generalization of the results. We find that it is not of relevance to analyze these threats within the scope of the machine learning methods.

As this analysis shows, just a small subset of the threats to validity concerned in section 3.3 are relevant for the results of the machine learning models. Further analysis of the relevant threats shows that the safety measures suggested in advanced have contributed to prevent any of the relevant threats to compromise the validity of the results.

5.3 Audit data

The task of analyzing the audit data is hard and diffuse. This is because we have a two part responsibility between the machine learning methods used

and the available audit data. In this analysis we have emphasized how well the available data mirrors the actual behavior of the user's interaction with the system, the consistency of the audit data, and how well we were able to extract data from the audit data by log parsing. The rest of this section is organized as follows. First, we look at the available audit data and its strengths and shortcomings. Then, we analyze the log parsing conducted by this project, to see its effect on the general performance. Finally, we look at how the research questions concerning audit data are answered and how the threats of validity are handled.

5.3.1 Available audit data

We were provided with a large quantity of three different logs available for this project. One of these logs, the application log, was used in the project carried out the semester before this project [KK07]. The evaluation of this log, conducted in the preliminary project, led to the conclusion that its quality was too poor. The consistency of this log would lead to great problems, especially when wanting to use it in combination with an anomaly detection system. This led us to focus on the web and transaction logs as information sources for this project.

The transaction log lists all actions performed on behalf of user requests. It thereby contains the actions performed concerning moving an amount of money from one account to another. Most of the attributes in these entries originates from user input, of which receiver account number, amount, and message provide examples. As discussed earlier, attributes containing strings impose some problems for machine learning methods, which mostly can handle only numerical and categorical values. However, values containing information which is directly input by the user is of such interests that they need to be included.

We described a simple transformation scheme that transformed important aspects and properties of the string's value into numerical values. This was a necessity in order to keep some of the information in the string attributes. However, this transformation scheme imports some issues since not all of the information from the strings can be transformed over to numerical values. This naturally raises the question of what information should be transformed. In this case, we chose a simple solution for bringing only the most important properties over to a numerical form. As proposed in section 4.1, there are also other types of information that can be extracted from the attributes.

Besides the transformation of strings, the transaction entries from the log seem very consistent and suited for machine learning, but we also see limitations in the information available, which could come in handy in a

detection scheme, such as personal user information, account information, and receiver information.

The web log is the primary source of information for the user's direct interaction with the system interface through a web browser. However, there are limitations in what information is recorded. For example, only the GET parameters are recorded not the POST parameters[1]. This information would be of interest in any IDS, since any part of user interaction could contain valuable information.

There also exist several types of information regarding the user that could be collected, which might be used to increase the value of the logs when considering fraud detection. Examples of these are geolocation of IP addresses[2] and robot detection [KK07]. However, despite the mentioned weaknesses, the log contains valuable information which seems to be enough to create profiles of normal behavior. A great strength of the web log is that it follows a consistent format which leads to a high consistency in the data.

5.3.2 Log parsing

Log parsing is the action of transforming a general log over to a defined format. This is very important since machine learning methods look at each instance in a log as equals. Hence, the need for a consistent format on the log data is vital in order to utilize the data at all.

The parser was, for each log type, able to parse most of the instances over to a consistent format. However, some atrophy was experienced. The largest problems were with the application log, which seemed to have different data providers leading to a high rate of inconsistency. With a general parsing success of 47% of the instances, it is clear that this log is too inconsistent to provide high quality data for a machine learning method. The unparsed data from the application log seemed to be various stack traces and snippets of HTML code.

The transaction log provided a much more consistent format, and we were able to successfully parse all the instances involving transactions. However, 28% of the log's content was considered not to contain any additional information, and was discarded. The web log had an even better success rate of close to 100%. This is due to the strict format the web log is founded

[1]POST and GET are different methods that define the way attributes are carried within requests. In the GET method, the attributes set are appended to the Uniform Resource Locator (URL). In the POST method, the attributes are included in the body of the request.

[2]Geolocation refers to techniques used to identify the physical location of a computer connected to the Internet.

on. Based on this we can conclude that we were able to parse close to all of the information from the transaction log and the web log, but, we suffered a large loss of information from the application log, making it unsuited for parsing and machine learning. Based on this, we can say that our previous assumption of primarily using the transaction log and the web log was correct.

Another problem we experienced in the log parsing surfaced when we started to analyze the relations between instances with the goal of relating single instances to user sessions. This would mean that each instance in every log should be grouped by a session identifier provided. The way to identify the session of an instance varied between several different identifiers, such as session id, thread id, IP address, user id, and social security number (SSN). Clearly, this inconsistency lead to a large complexity in the parser that attempted to extract and group instances related to each session. This problem became even clearer when we were unable to find a session for several of the instances. Here, we can see that it is not only important to have consistency in form between instances, but also consistency in relations between instances.

5.3.3 Research questions related to Audit data

Following is answers to the research questions that are related to the audit logs.

RQ1.1: *Does the audit logs collect enough information about the user's interaction with the system to enable detection of fraud?* This is a very hard, if not impossible, question to answer. Since we deal with novelty behavior there is no real way of assuring that enough information is collected. A further question would be if all kinds of fraud will be manifested in some kind of abnormal behavior. This research question seems impossible to answer on a general basis for all kinds of fraud and malicious behavior. However, when we look at subsets of fraud and malicious behavior, it suddenly seems much easier to answer. Clearly, some kinds of fraud, malicious behavior or attempts on such will be manifested in attributes and values provided by the available audit logs.

5.3.4 Analysis of validity for audit data

Only the threats to validity concerning the data at hand are relevant. These originate from the internal and external validity threats. Internal validity is

considered first.

History: The history threat is concerned with the influence time could have on the data and its performance. It is obviously a threat that the available data is collected within a too narrow time interval to provide the proper variance in the data. It is also a fact that some parts of the data used is no longer valid due to system upgrades and alterations. Hence, we must conclude with that the history threat is valid for the models generated on outdated data. However, this threat is limited since the classification models can be rebuild on new valid data with ease.

Maturation: This threat is concerned with how users change their behavior to the same stimulus over time. This is a valid threat for the data at hand. However, we are aware of this property and have implemented our models so that they can be built on new data as behavior changes. This is more a question about updates of models, than properties within the data. We find it obvious that the user's interaction with the system will mature over time.

Instrumentation: looks at what and how data is collected. How data is collected is outside the scope of this analysis, but what data is collected is very relevant. The details and levels of data collected is a hard selection to perform. However, we find that the data collected provides proper information for this purpose. The transaction log provides enough information about each transaction. However, one can always look at additional sources for information that could lead to a better understanding of the user's behavior.

Selection: is the threat treating whether the result could be different if another set of instances where selected. This would be a major threat if we had limited data available. However, the data material provided to us is of such quantity that it would be a sufficient representation of the behavior. This point is also supported by the analysis of machine learning methods, where one could see that performance of the models did not increase as the training set became larger. Hence, we can conclude with that the selection threat is absent.

Mortality: is the fact that users can terminate their session with the system at any time, leaving sessions incomplete. However, by the manner the profiles are built now, this would not be a problem. A termination of a session will not lead to an incomplete data instance and hence will not

affect the profiles concerned with instances. The profiles concerned with sessions are concerned with state transitions, not complete sessions, so we disregard the mortality threat.

External validity is concerned with the generalization of the results to a broader population than the population used in this project. It is considered next.

Interaction of selection and treatment: During the review of threats we found that this threat was absent in the data set provided to us. After analyzing the data set more thoroughly we are even more secure that the population described in the data set is representative for the rest of the systems users.

Interaction of history and treatment: The data available to us is collected during several days, and hence, the possibility that the data is influenced by special times or dates should be eliminated.

5.4 Summary of analysis

This chapter provides a vast amount of information about parts of this project. During this chapter we have tried to analyze, not only the specific items, but also relevant research questions and threats to validity. In this section we try to bring these reoccurring themes to a conclusion by summarizing our analysis of the research questions and threats to validity.

5.4.1 Summary of result validity

In this section we summarize the findings from the analysis of the threats to validity of this project's results. We do this by looking at the four general points of validity we wish to protect and how well they are protected in this project.

Conclusion validity: The threats considered in conclusion validity originates from the possibility that the results found does not have a satisfying statistical significance. These threats have been analyzed for all the different machine learning methods used, and the results seem rather consistent. The threat of false negatives seems to be the largest threat to the validity of the results.

Internal validity: The internal validity is threatened by false relationship between instances and classifications due to unknown factors in the

data. We were early aware of this type of threats and, as described in section 3.3, we took precautions to limit this possible threat to the validity of our results. During the analysis of internal validity we found that, due to the nature of machine learning, these threats were limited and could easily be eliminated by updating the data set as changes occur.

Construct validity: Construct validity looks at the threats concerning relations between theory and observation. The analysis shows that these threats are limited in this project, but they can however not be disregarded. We find that mono-method bias is generally eliminated due to the high dimensionality in the data used by the machine learning methods. However, we cannot guarantee that the experimenters' expectancies have not been a factor in influencing the results. This is especially a concern with the work done here in this chapter, since analysis allows some subjective influence.

External validity: The external validity is concerned with whether the results were relevant for the population wishing to generalize to. During our analysis we found no indication that this threat should have caused any effect on this project's results. Due to the large quantity, variations, and quality of the data sources used, we have eliminated this threat.

5.4.2 Summary of research questions

In this section we summarize the answers to the five main research questions proposed in section 3.1. These answers are vital for the results of the final test to the hypothesis. During this analysis we have tried to answer to underlying research questions as best in order to be prepared to answer these main questions. The main questions' answers are directly affected by the answers provided in the previous analysis of the related subquestions.

RQ1: *Does the information contained within audit logs contain enough information about normal behavior to create a profile defining all types of normal behavior?* The analysis of the different machine learning methods shows that it is possible to create profiles defining normal activity of users' interaction with the system. We can with certainty say that this shows that the data contains enough regularity to define normality patterns. However, we also see that legal actions are classified as abnormal. This raises the question of the difference between normal actions and legal actions. The goal of this project was to learn normal

system behavior without considering what was legal. The basis was that illegal behavior would be manifested as abnormal behavior. So we can clearly say that the audit logs contain enough information to be able to define normal behavior.

RQ2: *Is it possible to detect malicious behavior and fraud based on audit records from different levels in the system?* In order to answer this question we have to look at the different logs that we have analyzed and used in this project. Obviously, different kinds of fraud can be detected by different kinds of information. The question is if the information from the different logs provides this kind of information.

RQ2.1: *Is it possible to detect malicious behavior and fraud based on transaction logs?* The transaction log would seem as a proper data source to detect fraudulent actions. However, as we analyzed the log we saw that there were some issues. Fraudulent actions can have many characteristics and manifest themselves in the data sources in different ways. The important thing is that the fraudulent actions are manifested in the data recorded by the log.

For the transaction log we have some variables that are determined by applications and some that are provided by the user. Both these types of variables are of interest for different reasons. This is because malicious behavior and fraud would manifest itself in different ways by the two different types of variables. Application provided variables would provide more structure and less variance and it would hence be easy to find abnormal deviations. User provided variables need a more thorough analysis, and contextual information is preferable in order to have some idea of what to look for.

The conclusion is that the transaction logs contain a lot of valid and interesting information, which in some cases can provide values of variables that suggest fraud or malicious behavior. However, a lot of this information is provided by the user which means that the variables have much less structure and higher variance than application provided variables.

RQ2.2: *Is it possible to detect malicious behavior and fraud based on application logs?* After the research questions were defined and we started to work with the available audit logs we found that not all the logs were suited for usage by this project. As seen in this chapter, all research questions concerning the application logs are left

unanswered. This is due to the fact that we found the application logs unsuited for further work with machine learning methods, and hence no results were generated by this log. All subquestions to this research question have thus been left unanswered.

RQ2.3: *Is it possible to detect malicious behavior and fraud based on web logs?* The most outstanding property of the web log compared to the transaction log is that it shows a much clearer structure for the values of each attributes. This is due to the fact that most values are set by the web server applications, and are not provided by the user directly. This fact has an advantage and a drawback. The advantage is that is easier to detect anomalous values in such well structured attributes. This would lead to a higher precision in the detection since there is less room for the user to interfere with the values. The drawback is that since the data in the log is mostly provided by an application, the possibility of it being able to detect fraud and misuse is reduced. However, we see that some behaviors indicating fraud and misuse can be manifested in such a log, i.e. path traversal, application probing, and attempts to reach predictable resources.

RQ2.4: *Is it possible to detect malicious behavior and fraud based on information from a combination of logs?* As explained in 2.1.2, combining information related to specific sessions from the various logs proved to be a challenging and time consuming task. Therefore, we chose not to investigate this question.

RQ3: *Will malicious activity and fraud always be classified as abnormal data?* This is clearly an ambitious question to answer, and the research we have conducted has not a clear answer to this. Based on a theoretical level, we can say that as the state of the art is today, there clearly exist types of fraud and misuse that would not be detected within abnormal data. An example of this is the Trojan scenario defined in section 1.2.3. These types of threats are hard to cope with since the behavior of a Trojan would exhibit limited deviations, compared to that of the legitimate user. The need for further research on creation of profiles of user behavior is apparent and highly valuable for future anomaly based fraud and misuse detection.

RQ4: *Can malicious behavior be detected by the use of profiles?* Since we are unable to test the performance of our profiles' fraud and misuse detection, we are unable to answer this question based on our experiences. However, based on the work done in the study of the state of the art,

we clearly can state that malicious behavior can be detected with the use of profiles. Each profile gathers a set of properties that are found in the available data. These properties can be learned by looking at what values they take on a general basis. Malicious behavior deviates from the ordinary since it tries to achieve some unforeseen or unintended reaction from the server, and hence would be of an anomalous nature.

This summarizes the answers to our research questions. We notice that many of the questions have been answered more on a theoretical level, than based on actual experiences and results. This is mostly due to problems related to testing and evaluating the implementations of the profiles. However, the answers are still valid and of value for this project.

Chapter 6

Discussion

Summary

This chapter sums up the project and extracts the contribution. First, we look at our findings that obtained by creating and evaluating a set of profiles, utilizing machine learning methods, and analyzing the results. Then, we look at our hypothesis and try to verify or falsify it based on our findings. In order to see the contributions of this project, we compare our findings to the background material of this research field. After this we formulate a platform for further research which would lead any further work in the right direction.

6.1 Findings

This section will describe our findings in this project. These findings are a direct result of the analysis of the results we have generated by testing machine learning methods on audit data for the purpose of fraud and misuse detection. This section is organized as follows. First, we look at the findings concerned with profile creation, selection, and usage. Then, we look at our discoveries related to implementing these profiles by different machine learning methods. After this, we discuss the results generated by our implementations and findings concerning these. Finally, we use our findings to test our hypothesis, so that a formal contribution in terms of a verification or falsification can be provided.

6.1.1 Profiles

The design of profiles for detection of fraud and misuse by anomalous behavior is hard, primarily due to the fact that we are dealing with the unexpected and unknown. The basis of anomaly detection is to detect behavior that is new and undiscovered, hence no assumptions about the behavior can be made. Creating a profile for fraud detection without any assumptions means having to take everything into account. Since nothing is certain, it is impossible to focus or limit expected behavior or attribute values. Due to this we cannot make any assumptions on the anomalies, rather we have to focus on behavior that we can make assumption of. This behavior is that of the normal users in the system. The only assumption we can make about these is that nearly all usage of the system is legal and can, hence, be used to learn the legal usage of the system.

A limitation for profile design is the available data material documenting the user's interaction with the system. In order to learn a total picture of the usage of the system, all information that the user inputs and generates on the system have to be stored. This is not a practical nor efficient solution. One of the keys to success in an anomaly detection system is that the audit logs contain a sufficient amount of relevant information. During this project we experienced that some logs contained a great deal of information that was of limited significance for anomaly detection.

One of the key findings of this project is that it seems more relevant to focus on higher level profiles. By higher level we mean profiles that take into account several log instances, and perhaps several different logs. We found that profiles at a lower level were unable to detect deviations in user behavior due to the limitations in scope. This does not mean that lower level profiles are not of interest, on the contrary, we find that low level profiles can pro-

vide highly relevant detection. However, it would detect types of behavior that would deviate on an attribute or instance level, not on a general behavior level. We also remark that low level profiles would have a lower impact on security, since detection on attribute level is a much more mature field than that on the behavior level. By this we mean that regular security measures usually include input validation and attribute analysis, while behavior detection is still a field in need of research [fBS04].

6.1.2 Audit data

After working with authentic audit logs we discovered that the log data is not always as consistent and of the quality desired. When working with machine learning methods we have to provide data that are on a consistent format. Permutations, deviations, and regular instances will all be evaluated on a common basis, making inconsistency a crucial threat to the validity of any result for a machine learning method. In this project we parsed the logs to a common format in order to utilize the data. This approach has its limitations, since we were unable to parse all logs and instances. Preventing loss of data in the log parser is an area which should be worked further with, since any loss of data is a potential loss of performance.

Another problem with the audit logs was how to track information across different logs. There was no unique global identifier for different users or sessions, making the collection of these hard. The inconsistency of identifiers across logs is something that can be improved by two methods. Firstly, by reconstructing the audit mechanisms so that it produces logs with a global consistency. However, this approach is unlikely due to impact on existing system functionality. The second is to build a better log parser that is able to collect more information about each user and session. This is an area which needs further research, and if conducted successfully, could lead to an improvement of profiles that can detect deviations in session and user behavior.

6.1.3 Machine learning

Three different machine learning methods have been employed to implement the two selected profiles. Based on the properties of the different methods, ANN and SVM were employed on the low-level user request profile, while Markov chains was employed in the more general high-level session structure profile. A general observation for the machine learning methods is that they all provide the wanted functionality and were able to detect anomalies in the data.

One key finding concerning machine learning methods is that they are fastidious regarding the input data. A general property is the restriction of only accepting numerical and categorical values. This leads to a reduction in the information available to build profiles on, and could further lead to a reduction in performance. String values are the attribute type that suffers the most due to this. Even though we created a transformation of the most important properties of a string, this is an area that needs further research. This limitation is not inherent in Markov chains in the same manner, since all they require are an enumeration of some values, be it textual or numerical to define as states, and sequences of them to interpret as transitions. However, the state space can not be infinite, and so the Markov chain can be considered to require categorical values as states as well.

The main problem with the machine learning methods was the absence of measures that could be used to evaluate their performance in detection of fraud and misuse. We search to find behaviors that are of a novel type and, naturally, we do not possess such examples. This leads to the question of how to evaluate the methods' and profiles' performance. One way would be to make a synthetic test set. However this would make assumptions on the properties of fraud and misuse behavior of which we have no basis. Another way would be to use historical data of fraud and misuse to make a test set, but this would only provide a measure on the detection of historical events. Hence it is unsuited to provide a measure on future detection of novel fraudulent and malicious events.

6.1.4 Results

After the profiles were implemented in the different machine learning methods, we were able to run them on sets of logs in order to see how normal behavior was learned and anomalies detected. All machine learning methods were able to learn the behavior and range of attribute values that were normal. They were also able to detect instances that deviated from the norm and classify them as anomalies.

The key observation within the results is that every data set has anomalies, but an anomaly is not equal to a fraudulent or malicious act. By looking at the instances classified as anomalies in more detail, we could look for properties within the instances that would be of an anomalous nature. A general observation was that user provided data could contain strange and illogical values. Most of the detected anomalies were due to the user providing unexpected values to different attributes. Application provided values had a much clearer structure and provided a better material for division between normal and abnormal values. Based on this, the key finding is that user provided

data needs more analysis and processing in order to provide a foundation for determining the normal ranges of the attributes' values.

An interesting problem we experienced during analysis of the results was how the fraudulent and malicious instances could be found within the set of anomalies. Clearly, not all anomalies are of a fraudulent or malicious origin. The majority will most likely be classified as anomalies due to strange behavior or rare values. Even with a detection rate such that 0.1% anomalies are detected in a data set, one can end up with thousands of anomalous instances, due to the size of the set. This leads to two problems that need further research. First, whether all instances that are of a fraudulent and malicious nature are within the set of anomalies. Second, how could the fraudulent and malicious instances be found within the set of anomalies. This project has only been able to detect these two problems, not provide a clear solution for how to answer them.

The system investigated in this project is not mature enough to be used as a security measure in the targeted domain. Initially, we see such a system as a tool for providers of banking applications in a process in which the available audit data is searched for fraudulent activity. As long as fraudulent behavior deviates from what is normal, it can aid in filtering out the normal data, to reduce the amount of information subject to other analysis.

6.1.5 Result of hypothesis test

In section 1.3 we described a problem definition we wished to answer in this project. The proceeding work led to the creation of a hypothesis which should be tested in order to provide an answer to the problem definition. Next, we will look at how the findings in the previous sections have lead to a verification or falsification of the hypothesis. The hypothesis was formulated as follows:

> *If it is possible to detect novel fraud and misuse by using anomaly detection on audit data from Internet banking systems, then it is possible to do so with the use of machine learning methods on audit data.*

We can break the hypothesis into three subsets that can each be verified or falsified. The first consist of the question whether machine learning methods can be used to perform any kind of detection that some other method is able to do. By this we mean that if somebody or something is able to detect an event or instance based on a set of premises, then machine learning methods can also perform such a detection. The answer is that as long at the detection

takes place based on a set of definable premises, then a machine learning method can learn these premises and carry out the detection. This part is hence verified.

The second part is a generalization of the subjects we want to detect. It is concerned with whether we are able to detect novel fraud and misuse at all by anomaly detection. Clearly, we can think of novel fraud and misuse that would have similar properties to existing fraud and misuse. Many of these occur when providing input values that would lead to an unintended response in the system, and hence, would be detected as an anomaly. So also this part can be verified.

The third part concerns whether the hypothesis can be answered for the special case of detection of the majority of novel fraud and misuse. This means that we want our hypothesis to be valid on a general basis, not just for a limited number of imaginable novel incidents. Here, we encounter a problem, which has been mentioned earlier in the findings. This is whether machine learning methods can be prepared to detect the unexpected and unknown. On the basis of the previous sub-verifications of the hypothesis, it is tempting to say that also this part is valid. However, there is no way to ensure that our hypothesis is valid for this case. This is because there is no way of knowing what to expect from novel incidents. Based on this we have to falsify our hypothesis, since there is no way of knowing how future novel fraud and misuse attempts will behave.

6.2 Comparison to background

In order to attach our contribution to the related research area we look at our findings in the light of the research and information reviewed in sections 1.1 and 2.2. In the following section we compare our findings to the related background material.

In section 1.1 we identified a general absence of a complete security measure for Internet banking system. The problem is that security measures do not return the investment cost in terms of detecting and preventing fraud and misuse. It seems instead as security by obscurity is just as important as a functional security measure. Based on this we identified the need for a cost efficient security mechanism that could be added and customized without interfering with the existing system.

The solution proposed by this report provided the identified properties needed of the security measure. By using the audit data as our data source we can collect information about the system, the users, and the interaction between them without interfering with the current system. By utilizing ma-

chine learning the solution is automatically customized to be used on the audit data available. Finally, when looking for anomalies, we find only the fraud and misuse instances that have not already been identified by existing security measures.

As seen in section 2.2 there has been conducted a lot of research on the area of fraud and anomaly detection by machine learning methods. This report tries to take this research further, by implementing state of the art machine learning methods and fraud detection schemes in a special case on a well known system, based on structured and well defined audit logs. By doing so, we wish to investigate the performance of state of the art methods by specializing them within a concrete domain.

Since we were unable to evaluate the performance of the different machine learning methods we find it more relevant to look at the basis of our work which is to define profiles for a concrete domain and use the knowledge and understanding of the domain, the system and the available data to improve these profiles. Our proposal was that by looking at a concrete domain we could use additional information to improve the performance of machine learning methods.

Several researches have considered a concrete area when trying to achieve anomaly detection with the use of machine learning. For instance, [AFR97] and [GSS99] looks at credit card transactions and system calls, respectively. The use of profiles that analyze data at different levels have also been researched in [MP99]. However, our work deviates compared to both of these areas in key points. Firstly, we look at a domain in which public research is very limited. We were unable to find any kind of publication that employed profile based detection on real data from an Internet banking system. Further, most of the research conducted on concrete domains have been conducted on very specific and structured data sets, such as credit card transactions, which has a very consistent and simple form for each instance. We have chosen not to limit the data used to basic data sets, rather we have tried to use more of the available information provided by the audit logs.

The second area, profile at different levels, has also been proposed in earlier research. However, we can not find any research that suggests such a total approach as ours. Although we had to select a subset of our designed profiles, due to limitations in working hours, our suggestion is designed to detect fraud and misuse at every level of data abstraction. The research we found that included profiles based on abstraction levels of the available information include [MP99] and [CUK02]. These researches are based on information sources from standardized systems following international standards. They also focus on profiles from a special area of data abstraction, such as *user level* or at different time frames. In our case we look at a highly

customized system that follows a mix of international and custom standards. Our profiles are especially tailored towards fitting the available information from the system in a proper way. Further, we have not chosen to look at a specific level of abstraction, but rather tried to gain a complete set of profiles that could provide picture of all aspects of the system.

6.3 Further research

The information described here is probably the contribution we find the most important in our work. As this subject clearly needs further research, it is very important to use the work and results provided by this project to create a foundation for further research. The use of anomaly detection and machine learning to detect fraud and malicious behavior has proven complex and hard. Therefore, it is even more important to lay an early foundation that can lead the research in the right direction, toward the ultimate goal of a functional solution that can be used in a production environment with satisfactory accuracy. Based on this, we have identified some key research areas which we would like to emphasize. These areas are those we found vital to answer in order to get satisfactory results, and are described here.

String analysis: Machine learning methods have a general limitation in utilizing string values. In order to deal with this limitation, previous research [KVR05], and this report, have proposed transformations of textual properties to numerical values. However, there is still a lot of information that we are unable to transform. Strings can contain valuable information and are of interest due to the fact that it often is directly provided by the user. Hence, there is a need to further research how to extract as much information from a string as possible. Possible further transformations could include extraction of grammars and understanding of sentences.

Profile creation: This project chose to create profiles to form the foundation of the detection of abnormal instances. The profiles suggested by this project were designed based on what information was available. The implemented profiles were limited by the available amount of work available to this project. Due to this, interesting profiles could not be selected for further work. We also acknowledge that there are other profiles that should be evaluated and researched further. Other types of information can be taken into account and data can be extracted differently from the logs. Hence we think it is a good idea to work further with creation and evaluation of different profiles.

User profiles: One profile with a significant potential is the user profile suggested by this report. Due to the limitations discussed, we could not consider it further in this project. However, we feel that this profile has the most potential among those not implemented here. Because of the limitations found in the evaluation we propose that this profile can be generalized to some extent, but not to such a degree that it becomes a replica of the session structure profile. It is plausible to believe that users can be grouped into similar clusters based on behavior, and hence, the problem with low statistical relevance of each cluster would be avoided. A profile based on user groupings, clustered by behavior, is the preferred choice if one should work further with this project.

Session profiles: The session structure profile implemented in this project was rather simple. We were able to define some properties in the available data source as states, and calculate the probability for one of them occurring immediately after another. We stated the option of extending the memory of such a model, such that sequences of more than two states are considered, in addition to including information about the duration of the states. Such a model quickly becomes complex, and so it would be interesting to investigate the most beneficial properties to include into the model, while still keeping the complexity to a manageable level. We believe that such a model would be a valuable addition, if research further.

Evaluation of machine learning: A key problem we encountered in this project was the limitation in evaluating the different detection models created by the machine learning methods. Since we wanted to evaluate the profiles' ability to detect novel fraud and misuse, there existed no available test set or evaluation criteria. Hence, we needed to develop an evaluation scheme that was able to test profiles' performance on a common basis. Two suggestions were proposed by this report, however none of them were implemented. Both have clear strengths and weaknesses, but the actual value of such an evaluation is still unknown. This is because there is no way of knowing what to expect from future fraud and misuse acts, and hence, no assumptions can be made to found an evaluation. The area of evaluating the novelty detection of machine learning models needs further research.

6.4 Conclusion

The goal of this project was to investigate the possibility to utilize machine learning methods within anomaly based detection. The aim was to detect fraudulent activities within an Internet banking system based on audit records from the system. The analysis of the current state of security in this domain confirms such a need. By designing a set of profiles, we define models that capture the normal behavior of users on several levels of abstraction.

The results show that the models are capable of learning a baseline for what to consider normal, thereby providing a method to discover deviations from this. However, the models' performance, with regards to discovering malicious activity, could not be measured. We identify several immature areas that needs to be researched further in order for the value of such a system to become satisfactory. The first opportunity we see for such a system is in providing valuable assistance for Internet banking systems in determining a subset of all the available information to investigate further in search of traces of malicious activity. However, as this field matures and more public research is conducted, such a system may eventually become a cost-efficient security measure for the targeted domain.

Appendix A

Formats

This appendix describes standard formats encountered in this report.

A.1 Common log format

The common log format (CLF) is a standard format for recording requests processed by web servers. It can be produced by many different web servers, and read by many log analysis programs. The format is as follows [18]:

remotehost rfc931 authuser [date] "request" status bytes

The meaning of the different attributes are explained in the following list [18], [19]:

remotehost This is the IP address of the client which made the request to the server. If a *HostnameLookups*-directive is set to *on*, the server tries to resolve and log the hostname instead. Also, if a proxy server exists between the client and the server, this would be the address of the proxy, rather than the originating machine.

rfc931 The RFC 1413 identity of the client [Joh93]. This information is highly unreliable, and some servers, such as the Apache HTTP Server, do not log this information unless explicitly told to do so.

authuser The user identification of the person requesting the resource, as determined by HTTP authentication. If the status code of the request is 401, then this value should not be trusted, as the client is not yet authenticated.

date The date and time the server finished processing the request. An example of the default format for the Apache HTTP Server is: `[05/Jan/2008:08:09:03 +0100]`

request The request line exactly as it came from the client. First, the method used by the client is logged. Next, the resource requested is logged, including any GET-parameters. Finally, the protocol used is logged. An example is: `"GET / HTTP/1.0"`

status The HTTP status code that is returned to the client, as defined by [FGM+99].

bytes The size of the object returned to the client, not including the response headers.

A hyphen, `-`, in any of the fields indicates that the information was not available.

A.2 Combined log format

The NCSA Combined Log Format is an extension of the Common Log Format. The Combined format contains the same fields as the Common format, but three additional optional fields are present: *referrer*, *user_ agent*, and *cookie*. The meanings of these additional attributes are [20]:

referrer The URL which linked the client to the server

user_agent The software and platform used on the client to access the server.

cookie Any cookies sent to the server as part of the request. The format is `KEY=VALUE`, and multiple key-value pairs are delineated by semicolons.

A.3 The HTTP protocol

The Hypertext Transfer Protocol (HTTP) [FGM+99] is the transfer protocol used throughout the World Wide Web. It specifies the format allowed for messages passed between entities in a distributed environment. A HTTP messages is either a *request* sent from a client to a server, or a *response* sent from a server to a client. It consists of a *start line* followed by, optionally, one or more *header fields* and/or a *body*. The content of the messages are presented briefly in the next sections. The following character rules are encountered:

```
CR      = <US-ASCII CR, carriage return (13)>
LF      = <US-ASCII LF, linefeed (10)>
SP      = <US-ASCII SP, space (32)>
HT      = <US-ASCII HT, horizontal-tab (9)>
DIGIT   = <any US-ASCII digit "0".."9">
separators = "(" | ")" | "<" | ">" | "@"
           | "," | ";" | ":" | "\" | <">
           | "/" | "[" | "]" | "?" | "="
           | "{" | "}" | SP | HT
```

A.3.1 Requests messages

A request message consists of a *Request-Line* as its start line. Allowed header fields are *general-header*, *request-header* and *entity-header* fields, followed by the optional body.

```
Request = Request-Line
          *(( general-header
           | request-header
           | entity-header ) CRLF)
          CRLF
          [ message-body ]
```

The Request-Line is of special interest, since it corresponds to the *request*-attribute in the common log format. Its format is specified as follows:

```
Request-Line = Method SP Request-URI SP HTTP-Version CRLF
```

where *Method* indicates the method to be performed, *Request-URI* is the resource on which the method is to be performed, and *HTTP-Version* indicates the highest HTTP version for which the application satisfies all the MUST level requirements[1] in the HTTP specification.

The method is one of *OPTIONS*, *GET*, *HEAD*, *POST*, *PUT*, *DELETE*, *TRACE*, *CONNECT*, or any single US-ASCII character except US-ASCII control characters and separators. All methods are optional to support, except for GET and HEAD.

The Request-URI is either an asterix, (*), *absoluteURI*, *abs_path*, or *authority*.

```
Request-URI    = "*" | absoluteURI | abs_path | authority
```

The four options are dependent on the nature of the request. The asterix means that the request does not apply to a particular resource, and the *authority* form is only used by the *CONNECT* method. Further, as far as HTTP is concerned, URIs are formatted strings which, via some characteristics, identify a resource. The syntax and semantics of URIs are defined in [BLFM98].

The HTTP-Version consists of the string *HTTP/*, followed by one or more digits, followed by ., and ends with one or more digits.

```
HTTP-Version   = "HTTP" "/" 1*DIGIT "." 1*DIGIT
```

[1] The interpretation of the key word "MUST" is described in [Bra97].

Appendix B

Instruments

In this appendix we describe the instruments used in the construction and operation of this project. It consists of machine learning software, development tools and other types of instruments such as illustration and communication tools.

B.1 Machine learning software

Machine learning software is freely available to download and use. They are primarily in the form of libraries which can be embedded in larger applications. Commonly, they also include a graphical user interface, making them a stand-alone application for evaluating machine learning methods on given problems.

B.1.1 WEKA

WEKA (Waikato Environment for Knowledge Analysis) is a collection of machine learning algorithms for solving real-world data mining problems [21]. It is written in Java and runs on almost any platform. The algorithms can either be applied directly to a dataset or called from your own Java code. The WEKA workbench contains a collection of visualization tools and algorithms for data analysis and predictive modeling, together with graphical user interfaces for easy access to this functionality. WEKA supports several standard data mining tasks, more specifically, data preprocessing, clustering, classification, regression, visualization, and feature selection. It also provides an interface for additional machine learning libraries.

B.1.2 LIBSVM

In our research we used the LIBSVM (version 2.0) as an external library to Weka [22]. This is an integrated tool for support vector classification and regression which can handle one-class SVM using the Sholkopf algorithms [SSWB00]. We used the standard parameters of the algorithm. For this 'one-class SVM', one has to choose the original frequency representation, (e.g. the number of features), the appropriate kernel, and for each kernel the appropriate parameters.

B.2 Development tools

The development tools are those tools that are used in the development and implementation of the actual system.

B.2.1 Java

Java is a programming language, which has two important characteristics, object-orientation and platform independence [23]. The first characteristic, object-orientation, refers to a method of programming and language design. The basic idea is to gather various types of data with their relevant operations into entities called objects. An object can be thought of as self-contained, consisting of code (behavior) and data (state). Platform independence means that Java programs will run similarly on different hardware. This means one should be able to write a Java program on any computer, and run it on any other with the same result. This is achieved by translating the Java program code to an intermediate format called byte code. Java has also the possibility to import packages to expand the interface with even more specialized features.

B.2.2 Eclipse

Eclipse is an extensible open-source Integrated Development Environment (IDE) written primarily in Java [24]. In its default form, it is meant for Java developers, consisting of the Java Development Tools (JDT). The Eclipse platform, when combined with the JDT, offers many of the features one would expect from a commercial-quality IDE such as a syntax-highlighting editor, incremental code compilation, a thread-aware source-level debugger, a class navigator, a file/project manager, and interfaces to standard source control systems. Users can also extend its capabilities by installing plug-ins written for the Eclipse software framework, such as development toolkits for other programming languages.

B.2.3 IntelliJ IDEA

IntelliJ IDEA is another IDE for Java [25]. It supports a wide range of modern programming languages and technologies. Among the features of IntelliJ IDEA is a deep and intelligent editor and code analyzer, aiding the programmer in writing clean code efficiently.

B.3 Other instruments

Other instruments contain tools which does not fit the previous classes.

B.3.1 Dia

Dia is free open source general-purpose diagramming software [26]. Dia can be used to draw many different kinds of diagrams. It currently has special objects to help draw entity-relationship models, Unified Modeling Language (UML) diagrams, flowcharts, network diagrams, and simple electrical circuits. It is also possible to add support for new shapes by writing simple Extensible Markup Language (XML) files, using a subset of Scalable Vector Graphics (SVG) to draw the shape.

B.3.2 SSH/Putty

Secure Shell (SSH) is a network protocol that allows data to be exchanged using a secure channel between two computers [27]. Encryption provides confidentiality and integrity of data over an insecure network, such as the Internet. SSH uses public-key cryptography to authenticate the remote computer and allow the remote computer to authenticate the user, if necessary. PuTTY is a terminal emulator application which can act as a client for the SSH, Telnet, rlogin, and raw Transmission Control Protocol (TCP) computing protocols [28]. It provides all the needed features needed to communicate securely with a remote system over an insecure channel.

Appendix C

Implementation

Although we have utilized packages providing some of the methodologies we wanted to explore in this project, we had to implement some functionality ourselves. Off course, since some of the logs were custom logs from the particular system available, no packages were available to parse these. In addition, no suitable implementations of the LVQ and Markov chain were encountered in an appropriate programming language. Since their functionality is rather simple, we decided to write the code ourselves.

This appendix lists key snippet of code from the implemented functionality. All implementations are made in the Java programming language.

C.1 Log parsing

Logic for parsing the available logs is defined in the abstract class `LogReader`. This class must be extended in order to parse the specific formats of the various logs. A `DataCollector` can be attached to a `LogReader` in order to be notified when any `LogEntry` is read and parsed.

C.1.1 Log reader

Listing C.1 shows the most important methods of the abstract class `LogReader`. This class is extended by readers for specific logs, for instance by `TransactionLogReader` and `WebLogReader`, which then must implement the logic for parsing one or more log entries into a `LogEntry` in the method `readEntry()`. Reading
of a log file is then initiated by calling the method `setInputFilename(String)` with the filename of the log file to read, before calling `readFile()`. This method creates an appropriate `BufferedReader`, and while this reader has more lines to read, the `readEntry()`-method of the subclass is called, to read and parse one or more log lines into a `LogEntry` object. If a `LogEntry` could be successfully read and parsed, `fireEntryRead(LogEntry)` is called to notify any listening `DataCollectors` of this.

Listing C.1: class LogReader

```
1  public abstract class LogReader {
2
3    protected BufferedReader inputFileReader;
4    private String inputFilename;
5    private List<DataCollector> collectors;
6
7    ...
8
9    public abstract LogEntry readEntry();
10
11   public void setInputFilename(String inputFilename){
12     this.inputFilename = inputFilename;
13   }
14
15   protected void fireEntryRead(LogEntry entry){
16     for (DataCollector collector : collectors){
17       collector.entryRead(entry);
18     }
19   }
20
21   protected int readFile() throws IOException {
22     int count = 0;
23     Reader reader;
24     if (inputFilename.endsWith(".gz")){
25      reader = new InputStreamReader(
26                 new GZIPInputStream(
27                  new FileInputStream(inputFilename)));
28     }else {
29       reader = new FileReader(inputFilename);
30     }
31     inputFileReader = new BufferedReader(reader);
32     while (inputFileReader.ready()){
33       readEntry();
34       count++;
35       if (entry != null){
36         fireEntryRead(entry);
37       }
38     }
39     return count;
40   }
41 }
```

C.1.2 Data collectors

The interface `DataCollector` can be implemented by classes wanting to be notified when a `LogEntry` has been read. Thus, collection and interpretation of data is independent of the mechanism used to create the `LogEntry`-objects. Listing C.2 shows the method defined by this interface. Classes wanting to receive `LogEntry`-objects must implement the method `entryRead(LogEntry)`, and are then free to examine and interpret these in any way desired.

Listing C.2: interface DataCollector

```
1 public interface DataCollector {
2
3   /**
4    * Called when a LogReader, for which this object
5    * is registered as a datacollector, has read
6    * an entry.
7    * @param logEntry the entry recently read from the
8    *                 data source
9    */
10  public void entryRead(LogEntry logEntry);
11
12 }
```

C.2 Machine learning

We have implemented both a LVQ-I and a Markov chain. The following sections show the most important elements of the classes providing these functionalities.

C.2.1 Learning vector quantizer

The core functionality of the class implementing the LVQ-I is shown in listing C.3. The network is given an input vector in form of an array of doubles. Then, for each weight vector, it calculates the squared distance from the input vector to the weight vector. Then, the square root of the BMU's distance from the input vector is calculated to yield the true Euclidean distance.

If the network is in training mode, the learning rate and weights are updated by calling the methods `updateLearningRate()` and `updateWeights(double[], int)`, respectively. In addition, if the number of input vectors exceeds the number of training cycles defines, the network turns of training mode.

Listing C.3: class LearningVectorQuantizerI

```
1
2 public final class LearningVectorQuantizerI
3                      extends UnsupervisedNetwork {
4
5   ...
6
7   @Override
8   public int feed(double[] z) {
9     if (z.length != nofInputUnits) {
10      throw new IllegalArgumentException(
11              "Input vector contains " + z.length +
12              " elements " + "- expected " +
13              nofInputUnits);
14    }
15
16    double dist;
17    double minDist = Double.MAX_VALUE;
18    int bmu = -1;
19    inputCount++;
20
21    // Compute the distance, dist, between input
22    // vector z and each weight vector, weights[i]
23    for (int i = 0; i < nofOutputUnits; i++) {
24      dist = distanceSquared(z, weights[i]);
25      if (dist < minDist) {
26        minDist = dist;
27        bmu = i;
28      }
29    }
30
31    lastInputVector = z;
32    lastBMU = bmu;
33    lastMinimumDistance = Math.sqrt(minDist);
34
35    if (training) {
36      updateLearningRate();
37      updateWeights(z, bmu);
38      if (inputCount >= nofCycles){
39        setTraining(false);
40      }
41      fireUnsupervisedNetworkChanged();
42    } else {
```

```
43        fireUnsupervisedNetworkProbed();
44      }
45
46      return bmu;
47    }
48
49    private void updateWeights(double[] z, int bmuIdx){
50      for (int i = 0; i < weights[bmuIdx].length; i++){
51        weights[bmuIdx][i] += learningRate *
52                              (z[i] - weights[bmuIdx][i]);
53      }
54    }
55
56    private void updateLearningRate(){
57      learningRate = initialLearningRate *
58      Math.exp(-inputCount / expDenominator);
59    }
60  }
```

The abstract superclass `UnsupervisedNetwork` defines core functionality common for unsupervised networks, as shown in listing C.4. The method `feed(double[])` must be implemented by a subclass in order to define the logic for that specific unsupervised network. It should return the BMU for that input vector. Methods `randomizeWeights(int)` and `distanceSquared(double[], double[])` provides functionality to randomize the weight matrix and calculate the squared distance between two vectors, respectively.

Listing C.4: class UnsupervisedNetwork

```
1 public abstract class UnsupervisedNetwork
2                         extends Observable {
3
4   protected int nofInputUnits;
5   protected int nofOutputUnits;
6   protected boolean training;
7   protected double[][] weights;
8   protected double learningRate;
9   protected int nofCycles;
10
11  ...
12
13  public abstract int feed(double[] v);
14
15  public void randomizeWeights(int scale) {
16    Random rand = new Random(System.nanoTime());
17    for (int i = 0; i < weights.length; i++) {
18      for (int j = 0; j < weights[0].length; j++) {
19        weights[i][j] = rand.nextDouble() * scale;
20      }
21    }
22  }
23
24  protected static double distanceSquared(double[] z,
25                            double[] u) {
26    double dist = 0;
27    for (int i = 0; i < z.length; i++) {
28      dist += Math.pow(z[i] - u[i], 2);
29    }
30    return dist;
31  }
```

C.2.2 Markov Chain

The core elements of the class implementing a Markov chain is shown in listing C.5. It does not expect to know the number of states in advance, so a method, `addState(String)` is provided to add states dynamically. This method must then create a new and bigger transaction matrix, to hold the new state, while keeping the existing probabilities. Further, the private method `addTransition(int, int)` defines the functionality for incorporating information about a new transition, i.e. updating the appropriate probability

in the transition matrix. The method `addTransition(String, String)` is the publicly available method, accepting the names of the relevant states, and looking up the associated indices.

Listing C.5: class MarkovChain

```
1  public final class MarkovChain {
2
3    /** The transition counts */
4    private int[][] transCounts;
5    /** The transition probabilities */
6    private double[][] transMatrix;
7    /** The states in this Markov Chain */
8    private List<String> stateList;
9    /** Whether this Markov chain is in
10   training mode */
11   private boolean training = true;
12
13   ...
14
15   public void addState(String state){
16     if (stateList.contains(state)) {
17       return;
18     }
19     stateList.add(state);
20
21     int[][] transCountsOld = transCounts;
22     double[][] transMatrixOld = transMatrix;
23
24     transCounts = new int[transCountsOld.length + 1]
25                          [transCountsOld.length + 1];
26     transMatrix = new double[transCounts.length]
27                             [transCounts.length];
28
29     for (int i = 0; i < transCountsOld.length; i++) {
30       System.arraycopy(transCountsOld[i], 0,
31                        transCounts[i], 0,
32                        transCountsOld[i].length);
33       System.arraycopy(transMatrixOld[i], 0,
34                        transMatrix[i], 0,
35                        transMatrixOld[i].length);
36     }
37   }
38
```

```
39   public double addTransition(String prev,
40                   String next)
41                   throws IllegalArgumentException {
42
43     double retVal = getTransitionProbability(prev,
44                       next);
45     addState(prev);
46     addState(next);
47     inputCount++;
48     if (training) {
49       addTransition(stateList.indexOf(prev),
50                     stateList.indexOf(next));
51       if (inputCount >= nofCycles) {
52         training = false;
53       }
54     }
55     return retVal;
56   }
57
58   private void addTransition(int idxPrev,
59                                          int idxNext)
60                 throws IllegalArgumentException {
61
62     transCounts[idxPrev][idxNext]++;
63
64     //update transMatrix
65     int sum = DiplomToolkit.sumArrayInt(
66                                transCounts[idxPrev]);
67     for (int i = 0; i < transMatrix.length; i++) {
68       transMatrix[idxPrev][i] =
69         transCounts[idxPrev][i]/(double)sum;
70     }
71
72     fireMarkovChainChanged();
73   }
74 }
```

Appendix D

Additional figures

This appendix contains additional illustrations we found not important to include in the main sections of this report. They do not carry important information individually, but they still provide some value in the matter of completeness.

D.1 Markov chain figures

Figures D.1, D.2, and D.3 illustrate the Markov chain, trained on data from the web log. Similarly to the transaction log, they are divided into three figures for clarity.

Figure D.1 shows the transitions for which the probability is 1.0. Figure D.2 shows all transitions for which the probability is below 0.001. In figure D.3 all transitions not already illustrated in the previous figures are shown.

As can be seen, they are more complex than the figures for the transaction log. This confirms that they are influenced by the unpredictable behavior of users to a greater degree than is the case for the transaction log.

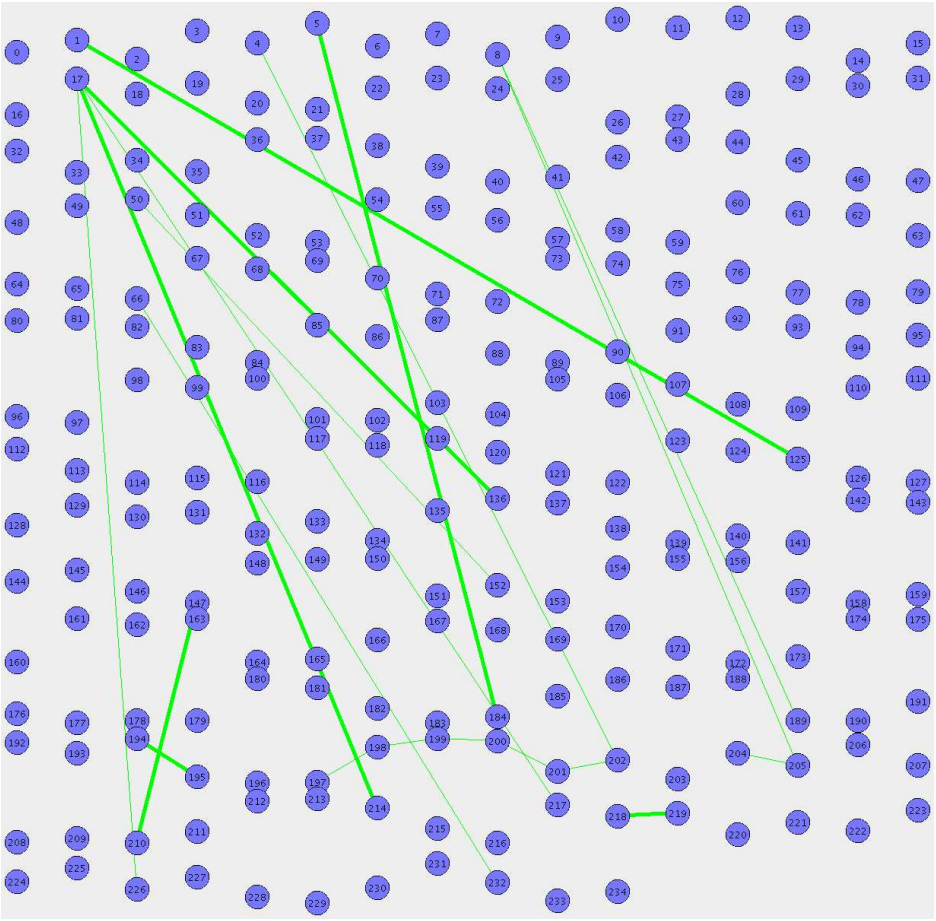

Figure D.1: Markov chain with high probability transitions trained on web logs

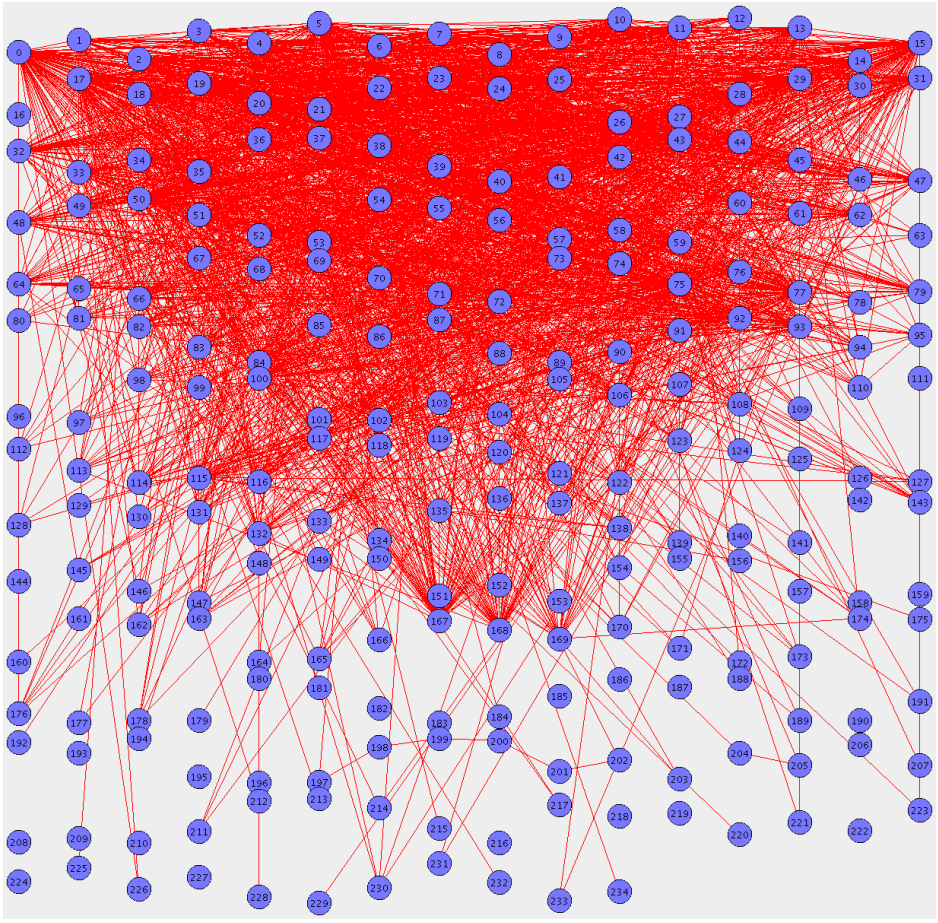

Figure D.2: Markov chain with low probability transitions trained on web logs

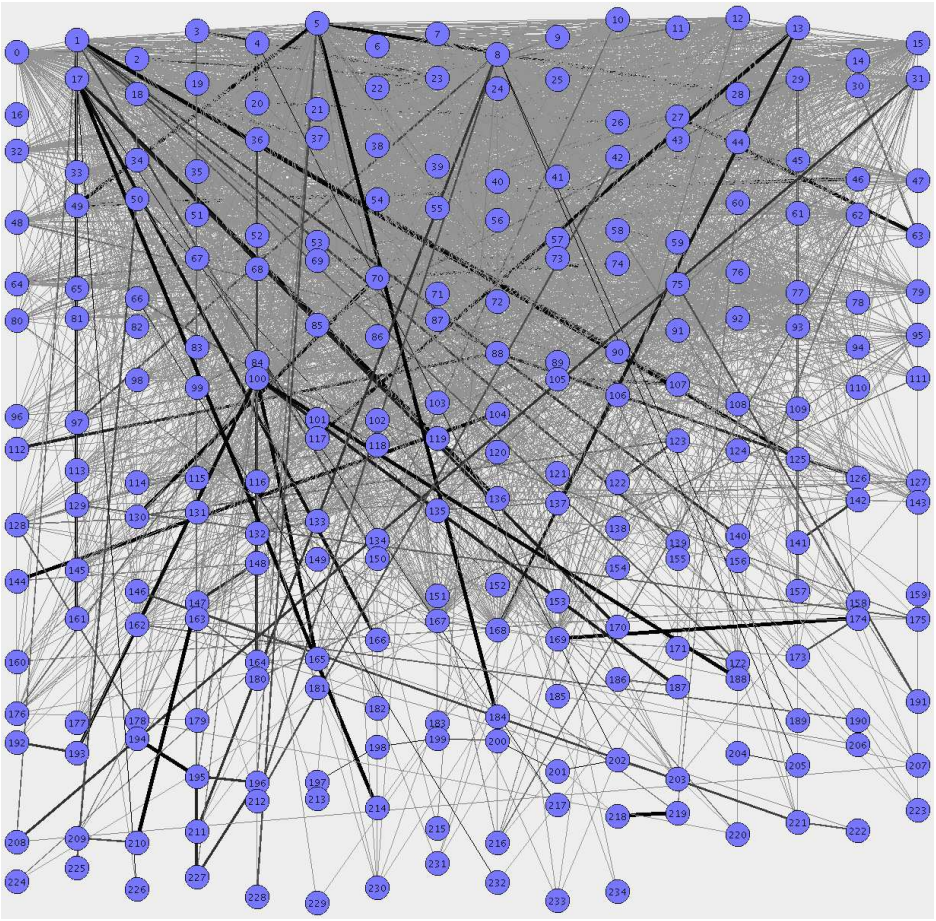

Figure D.3: Markov chain with medium probability transitions trained on web logs

References

[AFR97] E Aleskerov, B Freisleben, and B Rao. Cardwatch: A neural network-based database mining system for credit card fraud detection. In *Proceedings of the IEEE/IAFE on Computational Intelligence for Financial Engineering*, pages 220–226, 1997.

[All01] Douglas Allchin. Error types. *Perspectives on Science 9:38-59*, 2001.

[And80] J. P. Anderson. Computer security threat monitoring and surveillance, 1980.

[Bac00] Rebecca Gurley Bace. *Intrusion detection*. Macmillan Publishing Co., Inc., Indianapolis, IN, USA, 2000.

[BGV92] Bernhard E. Boser, Isabelle M. Guyon, and Vladimir N. Vapnik. A training algorithm for optimal margin classifiers. In *COLT '92: Proceedings of the fifth annual workshop on Computational learning theory*, pages 144–152, New York, NY, USA, 1992. ACM.

[BLFM98] T. Berners-Lee, R. Fielding, and L. Masinter. RFC2396: Uniform resource identifiers (uri): Generic syntax. `ftp://ftp.rfc-editor.org/in-notes/rfc2396.txt`, August 1998. Accessed 2008.01.22 - 10:00.

[Bra97] S. Bradner. RFC2119: Key words for use in RFCs to indicate requirement levels. `ftp://ftp.rfc-editor.org/in-notes/rfc2119.txt`, March 1997. Accessed 2008.02.22 - 10:00.

[BST01] Peter Burge and John Shawe-Taylor. An unsupervised neural network approach to profiling the behavior of mobile phone users for use in fraud detection. *J. Parallel Distrib. Comput.*, 61(7):915–925, 2001.

[Bur98] Christopher J. C. Burges. A tutorial on support vector machines for pattern recognition. *Data Min. Knowl. Discov.*, 2(2):121–167, 1998.

[CCCY05] Rong-Chang Chen, Tung-Shou Chen, Yuer Chien, and Yuru Yang. Novel questionnaire-responded transaction approach with svm for credit card fraud detection. In *ISNN (2)*, pages 916–921, 2005.

[CLM01] B. D. Cabrera, Lundy Lewis, and Raman K. Mehra. Detection and classification of intrusions and faults using sequences of system calls. *SIGMOD Rec.*, 30(4):25–34, 2001.

[CM02] A. Chakrabarti and B. Manimaran. Internet infrastructure security: A taxonomy. *IEEE Network*, 16(6):13–21, Nov-Dec 2002.

[Con] Web Application Security Consortium. Web application security consortium: Threat classification v.1.00. `http://www.webappsec.org`.

[Coo79] T. D. Cook. *Quasi-experimentation: Design and Analysis Issues for Field Settings*. Houghton Mifflin, Boston, MA, 1979.

[CUK02] Ramkumar Chinchani, Shambhu Upadhyaya, and Kevin Kwiat. Towards the scalable implementation of a user level anomaly detection system, 2002.

[CWHL] Chih Chung Chang Chih Wei Hsu and Chih Jen Lin. A practical guide to support vector classification. `citeseer.ist.psu.edu/689242.html`.

[DC] Jose R. Dorronsoro and Carlos Santa Cruz. Discrimination of overlapping data and credit card fraud detection. citeseer.ist.psu.edu/26951.html.

[Den87] Dorothy E. Denning. An intrusion-detection model. *IEEE Trans. Software Eng.*, 13(2):222–232, 1987.

[ELBJ03] Haakan Kvarnstrom Emilie Lundin Barse and Erland Jonsson. Synthesizing test data for fraud detection systems. In *ACSAC '03: Proceedings of the 19th Annual Computer Security Applications Conference*, page 384, Washington, DC, USA, 2003. IEEE Computer Society.

[Eng07] Andries P. Engelbrecht. *Computational Intelligence - An introduction.* John Wiley & Sons, 2 edition, 2007.

[fBS04] European Committee for Banking Standards. Security guidelines for e-banking. `http://www.ecbs.org/Download/tr411v2.pdf`, August 2004. Accessed 2008.01.29 - 10:00.

[FGM+99] R. Fielding, J. Gettys, J. Mogul, H. Frystyk, L. Masinter, P. Leach, and T. Berners-Lee. RFC2616: Hypertext transfer protocol – http/1.1. `ftp://ftp.rfc-editor.org/in-notes/rfc2616.txt`, June 1999. Accessed 2008.01.22 - 10:00.

[FHSL96] Stephanie Forrest, Steven A. Hofmeyr, Anil Somayaji, and Thomas A. Longstaff. A sense of self for Unix processes. In *Proceedinges of the 1996 IEEE Symposium on Research in Security and Privacy*, pages 120–128. IEEE Computer Society Press, 1996.

[Fin03] For Combating Financial. Design of an artificial immune system as a novel anomaly detector, 2003.

[GD03] A. Gretton and F. Desobry. online one-class -support vector machines. an application to signal segmentation. *IEEE ICASSP, Hong-Kong, April 2003.*, 2003.

[GE03] Isabelle Guyon and André Elisseeff. An introduction to variable and feature selection. *J. Mach. Learn. Res.*, 3:1157–1182, 2003.

[GSC06] Rupinder Gill, Jason Smith, and Andrew Clark. Experiences in passively detecting session hijacking attacks in ieee 802.11 networks. In *ACSW Frontiers '06: Proceedings of the 2006 Australasian workshops on Grid computing and e-research*, pages 221–230, Darlinghurst, Australia, Australia, 2006. Australian Computer Society, Inc.

[GSS99] Anup K. Ghosh, Aaron Schwartzbard, and Michael Schatz. Learning program behavior profiles for intrusion detection. In *Proceedings of the Workshop on Intrusion Detection and Network Monitoring*, pages 51–62, Berkeley, CA, USA, 1999. USENIX Association.

[Gun98] Steve R. Gunn. Support vector machines for classification and regression. *Technical Report ISIS-1-98, Department of Electronics and Computer Science, University of Southampton, 1998.*, 1998.

[HLV98] Wenjie Hu, Yihua Liao, and V. Rao Vemuri. Robust anomaly detection using support vector machines, 1998.

[HMT06] Kjell J. Hole, Vebjorn Moen, and Thomas Tjostheim. Case study: Online banking security. *IEEE Security and Privacy*, 4(2):14–20, 2006.

[Hor89] A. S. Hornby. *Oxford Advanced Learner's Dictionary of Current English*. Oxford University Press, Oxford, 4 edition, 1989.

[How98] John Douglas Howard. *An analysis of security incidents on the Internet 1989-1995*. Carnegie Mellon University, Pittsburgh, PA, USA, 1998.

[HSKS] Katherine A. Heller, Krysta M. Svore, Angelos D. Keromytis, and Salvatore J. Stolfo. One class support vector machines for detecting anomalous windows registry accesses. `citeseer.ist.psu.edu/634736.html`.

[Jap99] N. Japkowicz. *ConceptLearning in the Absence of Counter-Examples: An Autoassociation-Based Approach to Classification*. PhD thesis, Rutgers University, 1999.

[Joh93] M. St. Johns. RFC1413: Identification protocol. `ftp://ftp.rfc-editor.org/in-notes/rfc1413.txt`, February 1993. Accessed 2008.02.21 - 08:00.

[JSDJ01] H. JPT, T. SG, A. DG, and D. JJ. Statistical heterogeneity in systematic reviews of clinical trials: a critical appraisal of guidelines and practice, 2001.

[JW05] Hai Jiang and Hankang Wang. Markov chain based anomaly detection for wireless ad hoc distribution power communication

networks. *Power Engineering Conference, 2005. IPEC 2005. The 7th International*, pages 1–249, Nov. 29 2005-Dec. 2 2005.

[JY06] Songlun Zhao JingTao Yao. An enhanced support vector machine model for intrusion detection. *Rough Sets and Knowledge Technology*, 2006.

[KHK+96] Teuvo Kohonen, Jussi Hynninen, Jari Kangas, Jorma Laaksonen, and Kari Torkkola. LVQ_PAK: The learning vector quantization program package. Technical Report A30, Helsinki University of Technology, Laboratory of Computer and Information Science, FIN-02150 Espoo, Finland, 1996.

[KK07] Kåre Karlsen and Tarje Killingberg. Data collection in internet banking application for the purpose of intrusion detection. Project report, December 2007.

[Kok] A. I. Kokkinaki. On atypical database transactions: Identification of probable frauds using machine learning for user profiling. citeseer.ist.psu.edu/461086.html.

[KPJ+03] Hyun-Chul Kim, Shaoning Pang, Hong-Mo Je, Daijin Kim, and Sung Yang Bang. Constructing support vector machine ensemble. *Pattern Recognition*, 36(12):2757–2767, 2003.

[KVR05] Christopher Kruegel, Giovanni Vigna, and William Robertson. A multi-model approach to the detection of web-based attacks. *Computer Networks: The International Journal of Computer and Telecommunications Networking*, 48(5):717–738, August 2005.

[Lee99] Wenke Lee. *A data mining framework for constructing features and models for intrusion detection systems (computer security, network security).* PhD thesis, Columbia University, New York, NY, USA, 1999. Adviser-Salvatore J. Stolfo.

[Lou01] Daniel Lowry Lough. *A taxonomy of computer attacks with applications to wireless networks.* Virginia Polytechnic Institute and State University, 2001. Chairman-Nathaniel J. Davis, IV.

[LS00] Wenke Lee and Salvatore J. Stolfo. A framework for constructing features and models for intrusion detection systems. *Information and System Security*, 3(4):227–261, 2000.

[LT06] Kunlun Li and Guifa Teng. Unsupervised svm based on p-kernels for anomaly detection. In *ICICIC '06: Proceedings of the First International Conference on Innovative Computing, Information and Control*, pages 59–62, Washington, DC, USA, 2006. IEEE Computer Society.

[MDW+95] James Miller, John Daly, Murray Wood, Marc Roper, and Andrew Brooks. Statistical power and its subcomponents missing and misunderstood concepts in software engineering empirical research. Technical Report RR/95/192, Livingstone Tower, Richmond Street, Glasgow G1 1XH, Scotland, 1995.

[MH96] J. Kent Martin and Daniel S. Hirschberg. Small sample statistics for classification error rates II: Confidence intervals and significance tests. Technical Report ICS-TR-96-22, University of California, Irvine, 1996.

[MP99] Uzi Murad and Gadi Pinkas. Unsupervised profiling for identifying superimposed fraud. In *PKDD '99: Proceedings of the Third European Conference on Principles of Data Mining and Knowledge Discovery*, pages 251–261, London, UK, 1999. Springer-Verlag.

[MT93] S.P. Meyn and R.L. Tweedie. *Markov Chains and Stochastic Stability*. Springer-Verlag, 1993.

[MTV00] S. Maes, K. Tuyls, and B. Vanschoenwinkel. Machine learning techniques for fraud detection, 2000.

[MY00] Larry M. Manevitz and Malik Yousef. Document classification on neural networks using only positive examples (poster session). In *SIGIR '00: Proceedings of the 23rd annual international ACM SIGIR conference on Research and development in information retrieval*, pages 304–306, New York, NY, USA, 2000. ACM.

[MY01] Larry M. Manevitz and Malik Yousef. One-class SVMs for document classification. *Journal of Machine Learning Research*, 2:139–154, 2001.

[MZI08] Federico Maggi, Stefano Zanero, and Vincenzo Iozzo. Seeing the invisible: forensic uses of anomaly detection and machine learning. *SIGOPS Oper. Syst. Rev.*, 42(3):51–58, 2008.

[NAH05] Maria Nilsson, Anne Adams, and Simon Herd. Building security and trust in online banking. In *CHI '05: CHI '05 extended abstracts on Human factors in computing systems*, pages 1701–1704, New York, NY, USA, 2005. ACM.

[NC74] Vapnik V. N. and A. Ya. Chervonenkis. Teoriya raspoznavaniya obrazov: Statisticheskie problemy obucheniya. (theory of pattern recognition: Statistical problems of learning). *Moscow: Nauka*, 1974.

[NTGG+05] Anh Nguyen-Tuong, Salvatore Guarnieri, Doug Greene, Jeff Shirley, and David Evans. Automatically hardening web applications using precise tainting. In *SEC*, pages 295–308, 2005.

[OMSH03] Dirk Ourston, Sara Matzner, William Stump, and Bryan Hopkins. Applications of hidden markov models to detecting multistage network attacks. In *HICSS '03: Proceedings of the 36th Annual Hawaii International Conference on System Sciences (HICSS'03) - Track 9*, page 334.2, Washington, DC, USA, 2003. IEEE Computer Society.

[PB05] Tadeusz Pietraszek and Chris Vanden Berghe. Defending against injection attacks through context-sensitive string evaluation. In *Recent Advances in Intrusion Detection 2005 (RAID)*, 2005.

[PPLC06] KyoungSoo Park, Vivek S. Pai, Kang-Won Lee, and Seraphin Calo. Securing web service by automatic robot detection. In *ATEC '06: Proceedings of the annual conference on USENIX '06 Annual Technical Conference*, pages 23–23, Berkeley, CA, USA, 2006. USENIX Association.

[PS00] Joost M.E. Pennings and Ale Smidts. Assessing the construct validity of risk attitude. *Manage. Sci.*, 46(10):1337–1348, 2000.

[RLM98] Jake Ryan, Meng J. Lin, and Risto Miikkulainen. Intrusion detection with neural networks. *Advances in Neural Information Processing Systems*, 10:943–949, 1998.

[RP96] S. Roberts and W. Penny. A maximum certainty approach to feedforward neural networks. *In Press: Electronics Letters October 1996*, 1996.

[SEF08] Deborah Ashby Sandra Eldridge and Gene Feder. Internal and external validity of cluster randomised trials: systematic review of recent trials. *BMJ*, 2008.

[SPST+01] Bernhard Schölkopf, John C. Platt, John C. Shawe-Taylor, Alex J. Smola, and Robert C. Williamson. Estimating the support of a high-dimensional distribution. *Neural Comput.*, 13(7):1443–1471, 2001.

[SSWB00] Bernhard Schölkopf, Alex J. Smola, Robert C. Williamson, and Peter L. Bartlett. New support vector algorithms. *Neural Comput.*, 12(5):1207–1245, 2000.

[TS98] M. Theus and M. Schonlau. Intrusion detection based on structural zeroes. *Statistical Computing and Graphics Newsletter, Vol. 9, No 1, 12-17*, 1998.

[WBB08] Willem Waegeman, Bernard De Baets, and Luc Boullart. Roc analysis in ordinal regression learning. *Pattern Recogn. Lett.*, 29(1):1–9, 2008.

[WFP99] Christina Warrender, Stephanie Forrest, and Barak A. Pearlmutter. Detecting intrusions using system calls: Alternative data models. In *IEEE Symposium on Security and Privacy*, pages 133–145, 1999.

[WRH+00] Claes Wohlin, Per Runeson, Martin Höst, Magnus C. Ohlsson, Björn Regnell, and Anders Wesslén. *Experimentation in software engineering: an introduction*. Kluwer Academic Publishers, Norwell, MA, USA, 2000.

[WS02] David Wagner and Paolo Soto. Mimicry attacks on host-based intrusion detection systems. In *CCS '02: Proceedings of the 9th ACM conference on Computer and communications security*, pages 255–264, New York, NY, USA, 2002. ACM.

[YC01] N. Ye and Q. Chen. An anomaly detection technique based on a chi-square statistic for detecting intrusions into information systems. *Quality and Reliability Engineering International, 17:105–112, 2001.*, 2001.

[YD02] D. Yeung and Y. Ding. Host-based intrusion detection using dynamic and static behavioral models. *Pattern Recognition, vol. 36, pp. 229-243, 2002.*, 2002.

[YTWM04] Kenji Yamanishi, Jun-Ichi Takeuchi, Graham Williams, and Peter Milne. On-line unsupervised outlier detection using finite mixtures with discounting learning algorithms. *Data Min. Knowl. Discov.*, 8(3):275–300, 2004.

[Zan06] Stefano Zanero. *Unsupervised Learning Algorithms for Intrusion Detection*. PhD thesis, DEI Politecnico di Milano, 2006.

[ZS04] Stefano Zanero and Sergio M. Savaresi. Unsupervised learning techniques for an intrusion detection system. In *SAC '04: Proceedings of the 2004 ACM symposium on Applied computing*, pages 412–419, New York, NY, USA, 2004. ACM.

[ZY02] Pan Y. Zhang Y.Q. Parallel granular neural networks for fast credit card fraud detection. In *Proc. 2002 Internat. Conf., vol. 1*, pages 572–577, 2002.

Floating references

[1] Arne Hyttnes and Arne Skauge. Trygt aa bruke nettbank. http://www.aftenposten.no/meninger/debatt/article2113075.ece, November 2007. Accessed 2008.02.09 - 10:00.

[2] DN.no. Slakter nettbank-sikkerhet. http://www.dn.no/forsiden/article316366.ece, September 2004. Accessed 2008.02.05 - 10:00.

[3] Henrik V. Ebne. Kritiserer sikkerheten i nettbankene. http://forbrukerportalen.no/Artikler/2006/1150387578.78, July 2006. Accessed 2008.02.15 - 10:00.

[4] Morten Lyse. Nedslaaende sikkerhet i nettbankene. http://www.idg.no/computerworld/article54881.ece, May 2007. Accessed 2008.02.05 - 12:00.

[5] Assessment of Policing and Community Safety APACS. Fraud the facts 2007. http://www.apacs.org.uk/resources_publications/documents/FraudtheFacts2007.pdf, February 2008. Accessed 2008.01.25 - 10:00.

[6] Andreas Bakke Foss and Tone Tveoy Stom-Gundersen. Sikkerhet lonner seg ikke for nettbanker. http://e24.no/it/article2233891.ece, February 2008. Accessed 2008.02.14 - 10:00.

[7] CyberSource. Online fraud rapport. http://www.cybersource.com/resources/collateral/Resource_Center/whitepapers_and_

reports/CYBS_2008_FraudReport.pdf, January 2008. Accessed 2008.01.15 - 10:00.

[8] TriCipher. Tricipher consumer online banking study. http://www.antiphishing.org/sponsors_technical_papers/TriCipher_Consumer_Online_Banking_Study_4-5-07.pdf, March 2007. Accessed 2008.02.05 - 11:00.

[9] Morten Lyse. Slakter sikkerhet i nettbanker. http://e24.no/it/article1790444.ece, May 2007. Accessed 2008.01.15 - 13:00.

[10] Federal Trade Commission. Consumer fraud and identity theft complaint data 2007. http://www.consumer.gov/sentinel/pubs/top10fraud2007.pdf, February 2008. Accessed 2008.03.15 - 10:00.

[11] Joseph Yam. Internet banking: Some useful tips for users. http://www.info.gov.hk/hkma/eng/viewpt/20040715e.htm, July 2004. Accessed 2008.02.11 - 11:00.

[12] Einar Ryvarden. Slik blir norske nettbanker ranet. http://www.digi.no/php/art.php?id=363235, January 2007. Accessed 2008.01.25 - 13:00.

[13] Mat Buckland. Kohonen's self organizing feature maps. http://www.ai-junkie.com/ann/som/som1.html. Accessed 2008.05.09 - 12:00.

[14] Wikipedia. Markov chain. http://en.wikipedia.org/wiki/Markov_chain. Accessed 2008.01.21 - 15:00.

[15] Sanjay Kumar Bose. An introduction to queueing systems - lecture slides. http://www.ntu.edu.sg/home/eskbose/qbook/chapter_2.html, 2002. Accessed 2008.02.05 - 13:00.

[16] Web center of social research methods. Threats to construct validity. http://www.socialresearchmethods.net/kb/consthre.php, October 2006. Accessed 2008.03.21 - 16:00.

[17] Pai-Hsuen Chen, Chih-Jen Lin, and Bernhard Scholkopf. A tutorial on support vector machines. citeseer.ist.psu.edu/605359.html. Accessed 2008.03.02 - 09:00.

[18] Henrik Frystyk Nielsen. Logging control in W3C httpd. http://www.w3.org/Daemon/User/Config/Logging.html, July 1995. Accessed 2008.02.15 - 14:00.

[19] The Apache Software Foundation. Apache HTTP server version 2.2 - log files. http://httpd.apache.org/docs/2.2/logs.html. Accessed 2008.02.15 - 14:00.

[20] Log file formats. http://publib.boulder.ibm.com/tividd/td/ITWSA/_info45/en_US/HTML/guide/c-logs.html. Accessed 2008.05.10 - 11:40.

[21] The University of Waikato. Weka 3 - data mining with open source machine learning software in java. http://www.cs.waikato.ac.nz/ml/weka/. Accessed 2008.01.15 - 13:00.

[22] Chih-Chung Chang and Chih-Jen Lin. Libsvm – a library for support vector machines. http://www.csie.ntu.edu.tw/~cjlin/libsvm/. Accessed 2008.01.15 - 12:00.

[23] Sun Microsystems. Developer resource for java technology. http://java.sun.com/. Accessed 2008.01.15 - 10:00.

[24] Inc Eclipse Foundation. Eclipse.org home. http://www.eclipse.org/. Accessed 2008.01.15 - 12:00.

[25] JetBrains. Intellij idea: The most intelligent java ide. http://www.jetbrains.com/idea/. Accessed 2008.06.06 - 12:00.

[26] Gnome. Dia - gnome live! http://live.gnome.org/Dia. Accessed 2008.01.15 - 10:00.

[27] Wikipedia. Secure shell. http://en.wikipedia.org/wiki/Secure_Shell. Accessed 2008.01.15 - 10:00.

[28] Simon Tatham. Putty: A free telnet/ssh client. http://www.chiark.greenend.org.uk/~sgtatham/putty/. Accessed 2008.01.15 - 10:00.