Institute of Parallel and Distributed Systems
University of Stuttgart
Universitätsstraße 38
D–70569 Stuttgart

Diploma Thesis Nr. 3492

A Method for Security Breach Detection through File Access Monitoring and Pattern Recognition

Zeynep Öztürk

Course of Study:	Computer Science
Examiner:	Prof. Dr.-Ing. habil. Bernhard Mitschang
Supervisor:	Dipl. Inf. Tim Waizenegger
Commenced:	May 16, 2013
Completed:	November 15, 2013
CR-Classification:	C.2.0, C.2.3, C.2.4, D.4.6, K.6.5

Abstract

In the enterprise context a common requirement is to protect confidential information, such as sensitive customer data, internal corporate data, or research findings, not only against external, but also internal unauthorized access. The rapidly changing technology environment has seriously affected the computer security of organizations and governments around the world. According to the 2013 Data Breach Investigations Report from Verizon, more than 47,000 reported security incidents, 621 confirmed data breaches, and 44 million compromised records have been analyzed in 2012. Security breaches cause enormous damage and cost organizations billions of dollars annually.

Intrusion Detection Systems (IDS) or Intrusion Prevention Systems (IPS) are typical protection mechanisms that monitor security breaches. Nevertheless, these systems suffer from several major drawbacks, such as increased configuration complexity, high cost, high maintenance, high number of false alarms, and requiring a security administrator that can react with countermeasures to a security breach. In this context, the goal of this thesis is to develop a novel Breach Detection System (BDS) able to overcome the disadvantages of current IDSs or IPSs.

In order to detect and analyze security breaches at the operating system level, with a special focus on file operations, a BDS will be conceptualized and implemented. The aim of this BDS is to enable Security Breach Detection through File Access Monitoring and Pattern Recognition. For this purpose, a sensor is used to gather information about the system behavior while the system is running in a controlled state. Additional, a pattern recognition engine derives patterns from file access events. These patterns are used to monitor a process that accesses a certain file, and to determine the legitimacy of the file operations. It can also be used to suggest a possible access permission to an administrator.

Contents

List of Figures

List of Tables

List of Listings

List of Algorithms

1 Introduction

The time when important information used to be written on paper and stored in filing cabinets is as good as over. Document types have changed in recent years, from classic paper documents to electronic ones. Meanwhile, almost all information is created electronically by organizations, private persons, and governments around the world, provided by modern communication channels such as Internet or intranet.

Moreover, content management systems (CMS) provide a structure which aims to facilitate the maintenance, representation and usage of edited information like images, graphics, and texts[1]. For instance, IBM provides SmartCloud Content Management (SCCM), a cloud-based enterprise solution with the aim to maintain and archive critical information assets[2].

Naturally, this rapidly changing technology environment enables organizations like IBM new business opportunities, on the one hand, but on the other creates new threats, which can have serious effects on the security of information resources. A common requirement of organizations, private persons, and governments is that their confidential information or data has to be protected against external as well as internal unauthorized access. For organizations, confidential information is comprised of internal corporate data, research findings, scientific data, or sales figures; for private persons, their sensitive personal data like credit card numbers; and for governments, data about legal regulations or other specific data which might be interesting for third parties.

According to the 2013 Data Breach Investigations Report from Verizon [Ver13] more than 47,000 reported security incidents, 621 confirmed data breaches, and 44 million compromised records were analyzed in 2012. Security breaches cause enormous damage, and cost organizations, private persons, as well as governments billions of dollars annually, besides provoking loss of availability, confidentiality, or integrity which are basic security principles.

Typical protection mechanisms that have the aim of monitoring security breaches are Intrusion Detection Systems (IDS) or Intrusion Prevention Systems (IPS). These systems have the purpose of detecting attacks against networks and computer systems, ensuring compliance of the security policy, and initiating countermeasures if necessary. Nevertheless, these systems suffer from several major drawbacks, such as increased configuration complexity, high cost, high maintenance, high number of false alarms, and requiring a security administrator who can react with countermeasures to a security breach.

[1] https://www.bsi.bund.de/DE/Publikationen/Studien/CMS/Studie_CMS.html
[2] http://public.dhe.ibm.com/common/ssi/ecm/en/bud03045usen/BUD03045USEN.PDF

Goal of the thesis

The goal of this thesis is to conceptualize a novel method for Security Breach Detection through File Access Monitoring and Pattern Recognition, as well as to implement a Breach Detection System (BDS) framework, consisting of a communication protocol, a generic client component (Sensor), and a server component (Breach Detection Component), to detect and analyze security breaches at the operating system level, with a special focus on file operations. The purpose of implementing a BDS is to overcome the disadvantages of current Intrusion Detection Systems (IDS) or Intrusion Prevention Systems (IPS), and to give an insight into a novel approach that aims to find out if it is possible to derive patterns from file accesses in order to achieve a certain security level.

In this BDS, a sensor is used to gather information about the system behavior while the system is running in a controlled state. A pattern recognition engine is employed to derive patterns from file accesses events. These patterns are used to monitor a process that accesses a certain file and to determine the legitimacy of the operation. It can also be used to suggest a possible access permission to an administrator.

Structure of the thesis

This thesis is divided into six chapters. The *first part of the first chapter* will give a brief introduction into the topic as well as the goal and structure of this thesis. The *second part of the first chapter* is concerned with basic information about the underlying file concept, general aspects of computer security, and a brief introduction into the pattern recognition and pattern matching technologies that might help understand the further work. The *second chapter* presents the findings of literature research, and will give an insight into existing developments security implementation such as Intrusion Detection and Prevention Systems. The *third chapter* will introduce the novel concept of Security Breach Detection through File Access Monitoring and Pattern Recognition and will give the conceptual idea based of this thesis. Afterwards, a detailed insight into the prototypical implementation of the Breach Detection System (BDS) is described in *chapter four*. The *fifth chapter* will evaluate the concept and the implementation concerning open conceptual problems, performance, limitations and implementation hurdles. In conclusion, the *final chapter* summarizes the results of this thesis and gives a brief overview of the improvements and expansion possibilities for the novel approach and proposed Breach Detection System.

1.1 File System

Unix is a class of operating systems. This class includes Unix derivatives such as Linux, AIX and Solaris. A computer operating system offers a base which allows the user and the software to use a computer. It provides an execution environment for programs, file system management, networking stack, access to hardware devices, and administrative tasks. One of the most significant properties of Unix is the ability of allowing multiusers and multitasking. The term *multiuser* conveys that multiple users can work simultaneously on one system without disturbing each other. Each user has a separate area for their own data, which cannot be changed or read by other users. The term *multitasking* indicates that multiple programs can run simultaneously on one system [Kri98].

One can say that *"on a Unix system, everything is a file; if something is not a file, it is a process"* [Gar08]. This statement constitutes one of the meaningful characteristics of the Unix operating systems. The term *file* is widely spread in the Unix userland. A file is a kind of container that contains information about the data. Depending on their content or usage, a file can be classified into various types. This way, *regular files* are ones that cover text files, programs, and executable files. *Directories* are files that contain information about other files that are located in this directory. *Special device files* are file types like disk drives, webcam, printer, or monitor and are accessed through files. Reading or writing to special device files cause input or output on the corresponding devices.

In order to organize, manage, and store huge amount of files, these are grouped together into a file system. That means, a file system is a structure that is able to archive and administer data [Kri98]. According to the definition of *data* from the Internet Security Glossary (RFC 2828), *describes data information in a specific physical representation, usually a sequence of symbols that have meaning; especially a representation of information that can be processed or produced by a computer*.

The file system in Unix-like operating systems has a tree-like hierarchical directory structure, shown in Figure 1.1[3]. Each file is located in a directory, which itself is a subdirectory of a directory, except the root directory. The root directory ('/') is the root of the directory structure and contains standard directories. Some of the standard directories are listed in Table 1.1.

For the sake of completeness of the file system section, the implementation of a file system and the steps of a file access will be described in the next section in consonance with [Kri98].

In the Unix userland, files are realized with a file system, as described, above and with *inodes*. Data file directories contain only the name of the file and an assigned number. This assigned number is a reference to the inodes list. Figure 1.2 illustrates a data file directory with its corresponding inode list. A data file directory could be /home/freak/File. A number is assigned for each file name of the data file directory and this number is used as a index for

[3]http://www.imb-jena.de/ gmueller/kurse/linux/linuxfibel/dirstruct.htm

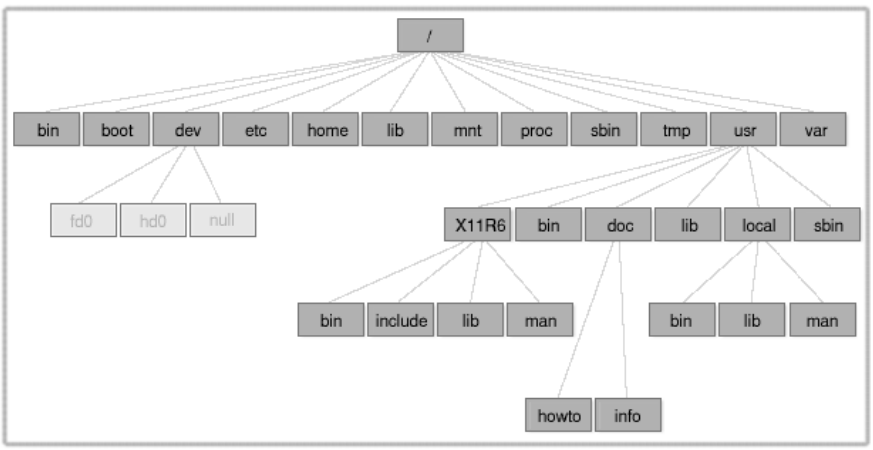

Figure 1.1: Tree-like hierarchical Directory Structure of a Unix File System

Standard directories	Description
/bin	executable files
/dev	special device files
/home	user directories
/lib	program libraries
/usr	Unix System Resources, system and application programs
/proc	special files for accessing kernel features
/etc	local configuration files
/sbin	system programs
/tmp	temporary files
.....	....

Table 1.1: Standard Directories with respective Descriptions

the list of inodes. The inode contains all information about the file, except the file name: file type, the number of the file names, access rights of the file, the user number and the user group number, the size of the file, address fields, the date and the time of the last modification, the date and time of the last file access.

This example in Figure 1.2 illustrates a file access of a user to the file */home/freak/File*. UNIX searches in the root directory ('/') for the data file directory of *home* and loads the assigned inode in the main memory. From this inode, UNIX searches the assigned data block which contains the directory entry *freak* and loads this in the main memory. An inode number is assigned to this directory entry. Afterwards, UNIX loads the assigned inode number of *freak* in the main memory and searches for the data block which contains the directory of *freak*. The data block is loaded and examined for *File1*. This entry is associated with an inode

number. The inode is loaded and contains the number of the first data block of the file. This inode can be made available now for an application which searched after this file.

To increase the efficiency of a file access, a *cache* is used which keeps a set of often accessed data blocks in the main memory. This way, not all blocks have to be loaded from the hard disk and as a result, access speed increases.

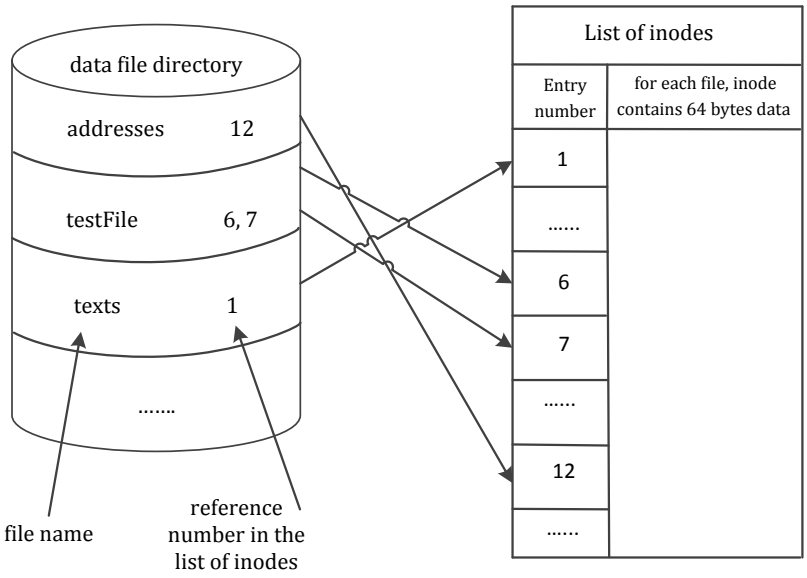

Figure 1.2: Data File Directory with its corresponding Inode List

1.2 LSOF - List Open Files

The Unix userland provides a tool called lsof. Lsof stands for *list open files*, and lists all information about files that are currently opened by a user or a running process.

In order to access a file, a process needs a file handle from the operating system. Therefore, a request like "Can you give me file handle for file 'file x'?" is sent to the underlying operating system, and as response it receives the file handle of the file. A file handle can be seen as an adapter for file accesses between a process and the underlying operating system.

Through the use of lsof, it is possible to retrieve information about this file handle from the operating system. It is possible to see which files are accessed by which processes. Information about network communication, system users, or devices can be displayed by choosing particular parameters. If lsof is executed with root rights, all information is displayed for any opened process that is opened by any user. A sample output executed with *root rights* and with the command *lsof* is shown in Table 1.2 [A.W06].

COMMAND	PID	USER	FD	TYPE	DEVICE	SIZE/OFF	NODE	NAME
sched	0	root	cwd	VDIR	136,8	1024	2	/
init	1	root	cwd	VDIR	136,8	1024	2	/
init	1	root	txt	VREG	136,8	49016	1655	/sbin/lsof
init	1	root	txt	VREG	136,8	51084	3185	/lib/libuutil.so.1
vi	2013	root	3u	VREG	136,8	0	8501	/var/tmp/ExXDaO7d
...	...	...	...	...	...	...	...	...

Table 1.2: Sample Lsof Output

Each line of the output represents one open file. *Command* stands for the name of the process, *PID* for the process identification number, *User* for the name of the system user, under which the process is running, *Type* for the format of the file, *Device* for the name of the device, *SIZE/OFF* for the size of the file, *Node* for file's identification on the disk and *Name* for the actual name of the file.

The column *FD* stands for the file descriptor which describes the file from the application point of view. The entry *cwd* in the FD column describes the current working directory which is the starting point of the application. The entry *txt* describes a program code and data like the application binary or a shared library. At the last line of this sample output, a user edits the file */var/tmp/ExXDaO7d* with *vi* and with the file descriptor *3u*. The *u* stands for the read/write mode. Possible further modes could be *r* for read-only and *w* for write-only. An application is opened with three file descriptors, 0 through 2, for the standard input, output, and error streams [A.W06].

As mentioned above, it is possible to list information about network communication, system users, or devices by choosing particular parameters. The application examples of lsof which are shown in Table 1.3 through 1.5 are taken from the homepage of Johannes Franken[4].

The first example in Table 1.3 will illustrate who accesses and uses the file *vim*. Command: *lsof /usr/bin/vim*

COMMAND	PID	USER	FD	TYPE	DEVICE	SIZE/OFF	NODE	NAME
vim	495	jfranken	txt	REG	3,3	1102088	175460	/usr/bin/vim
vim	1919	jfranken	txt	REG	3,3	1102088	175460	/usr/bin/vim

Table 1.3: Example Lsof Output for a File

The second example in Table 1.4 will illustrate the network communication of a given host and port. All processes that communicate on port 80 are listed. Command: *lsof -i :80*

[4]http://www.jfranken.de/homepages/johannes/vortraege/lsof_inhalt.de.html

COMMAND	PID	USER	FD	TYPE	DEVICE	SIZE/OFF	NODE	NAME
thttpd	569	root	ou	IPv4	2886			TCP *:www (LISTEN)
opera	3834	jfranken	20u	IPv4	86644			TCP localhost:1055-> localhost:www (CLOSE_WAIT)

Table 1.4: Example Lsof Output for a Network Communication

The third example in Table 1.5 will illustrate all processes that are opened with a given PID. Command: *lsof +p 3050*

COMMAND	PID	USER	FD	TYPE	DEVICE	SIZE/OFF	NODE	NAME
bash	3050	jfranken	cwd	DIR	3,64	2048	53248	/cdrom
bash	3050	jfranken	rtd	DIR	3,3	4096	2	/
bash	3050	jfranken	txt	REG	3,3	511400	191483	/bin/bash
bash	3050	jfranken	mem	REG	3,3	90210	159620	/lib/ld-2.2.5.so
bash	3050	jfranken	mem	REG	3,3	248132	160128	/lib/libncurses.so.5.2
bash	3050	jfranken	ou	CHR	136,3		5	/dev/pts/3
bash	3050	jfranken	1u	CHR	136,3		5	/dev/pts/3
bash	3050	jfranken	2u	CHR	136,3		5	/dev/pts/3
bash	3050	jfranken	255u	CHR	136,3		5	/dev/pts/3

Table 1.5: Example Lsof Output for a Process Identification Number

1.3 Computer Security

Information system resources such as processors, data storages, file systems or devices, as well as confidential information assets such as credit card numbers, sensitive customer or internal corporate data are prone to threats and attacks of third parties. The aim is to detect vulnerabilites of the system in order to violate the computer security and potential misuse of confidential information and information system resources.

According to the survey of the 15th Annual CSI Computer Crime and Security Survey of 2010/2011 [CSI11], some types of attacks that occured (by percent of 149 respondents) are as follows: malware infection 67%, password sniffing 12%, denial of service 17%, unauthorized access 13%, insider abuse of internet access or email 25% and system penetration by outsider 11%.

Another survey of the CyberSecurity Watch Survey of 2011 from the Computer Emergency Response Team (CERT) [CSO11] stated that whereas outsider attacks are more reported (58%) than insider attacks (21%), insider attacks cause more damage to organizations than outsider attacks. Insider attacks are attacks caused by employees or other third parties of the company with authorized access and outsider attacks are attacks caused by intruders without authorized access to the information system resources and confidential information.

Not only organizations, but also private persons are affected by threats from malicious attackers which intrude in the personal area over the Internet to spy on their private data like credit card numbers, online banking accesses or any online shopping accounts.

Considering the results of the surveys, a major task of organizations and private persons should be the protection of their computer security. A definition of the term *computer security* is given from National Institute of Standards and Technology (NIST)[5] in Computer Security, Principles and Practice [SB08] as follows:

The protection afforded to an automated information system in order to attain the applicable objectives of preserving the integrity, availability, and confidentiality of information system resources (includes hardware, software, firmware, information/data, and telecommunications).

A further definition of security concerning to information technology (IT) is defined in [Kap07] as follows:

The term IT security is used to protect confidential information and information systems against unauthorized access and manipulation. Furthermore, to ensure the availability of systems which provide services for legitimate users, including all measures for prevention, detection or logging of threats.

The common intention of both definitions is to ensure the confidentiality, availability and integrity of information assets, as well as of information system resources. Figure 1.2 shows the confidentiality-integrity-availability (CIA) triad which represents the fundamental concept of information security. These terms are defined and used in practice as *protective goals* or *security goals* in order to provide a base for securing information assets and information system resources.

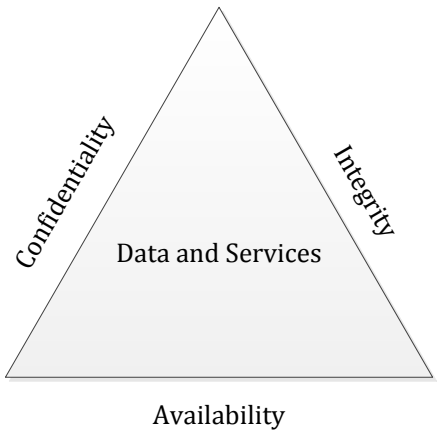

Figure 1.3: Confidentiality-Integrity-Availability (CIA) triad

According to [SB08] and [Kap07], the following section will define the terms confidentiality, availability and integrity. In addition to those protective goals, a further term is defined in practice to ensure the security of information assets and information system resources.

[5] http://www.nist.gov/

Confidentiality describes the protection of sensitive information against unauthorized accesses of third parties. Data confidentiality and privacy are two approaches in relation to confidentiality. Data confidentiality ensures the protection of sensitive data in order not to permit unauthorized access and privacy ensures the protection of information that might be collected or stored without permission.

Integrity describes the protection of sensitive information against unauthorized modifications of third parties. The integrity can be considered as data integrity and system integrity depending on the area of use. Data integrity ensures the modification of information and programs only with authorized access and system integrity ensures the modification of system functions

Availability describes the protection of information system resources and services in order to provide available access to legitimate user. An unavailable service or access of a server over a long period could be existence-threatening, for example, for an online retailer.

Authenticity describes the unique identification of an object or subject in order to ensure the reliability and the validity. An object in relation to information technology is a term which is used for defining files, data base entries and processes, whereas a subject defines the user of the objects for instance server, procedure or the user of the system [Eck12].

The aim of malicious attackers or intruders is to find weaknesses or vulnerabilities in computer networks or in IT infrastructure of organizations, as well as of private persons in order to gain access and breach confidential information assets or information system resources. A weakness or vulnerability of an information system is a flaw in which the system security could be manipulated, breached or spoofed by unauthorized third parties [Eck12]. These weaknesses and vulnerabilities can depend on different aspects and classified according to [Kap07] in following fault categories:

- *Requirement fault*: the requirements in relation to security are faulty and insufficient
- *Design fault*: the specification of a hardware or software component does not satisfy the requirements and contains weaknesses and vulnerabilities which can be exploited through malicious attacker
- *Implementation fault*: faulty implementation of the design specification which in turn can cause vulnerabilities
- *Installation and Administration fault*: during the installation or administration a system security function might be disabled which can cause vulnerabilities

1.4 Pattern Recognition and Pattern Matching

People have the ability to recognize coherences from a sequence of an information stream and build patterns out of it. An experimental test set-up from Jason Zweig, described in [Pet08], gives test participants a sequence of red and green symbols like *GGGG**R**GGGGG**RR**GGGGGG**RR**?*. The task was to guess which symbol will come after R

and if they can guess the right symbol, they will rewarded with 5 euro. In order to guess a sign, most of the test participants try intuitively to recognize a pattern from the given sequence.

Assuming, a sequence of several signs is given like the above mentioned example. There are two methods to determine whether a given string matches into the sequence. The sequence of several signs follows a pattern.

A pattern is described according to Norbert Wiener in [ST13]: *one of the most interesting aspects of the world is that it can be considered to be made up of patterns. A pattern is essentially an arrangement. It is characterized by the order of the elements of which it is made rather than by the intrinsic nature of these elements.* That means, a pattern classifies different objects depending on the application, for instance measured values as a signal, pixels of an image or a sequence of strings from a text.

The first method is *Pattern Recognition*. Pattern Recognition is a method which attempts to recognize a structure of a given sequence of objects in order to build a pattern. Based on these patterns, predictions can be made about the appearance of further objects. Pattern Recognition is used in different field of applications. Some of the applications are image processing, character recognition, document analysis, speech recognition or computational face recognition. In order to determine whether a given string matches into a pattern which is build of a sequence of several signs, a markov chain can be generated from the given string. Markov chain is an approach which can recognize simple patterns through the use of a state machine. With this state machine, it is possible to examine whether the state machine would be able to recognize the given string. If the state machine can recognize the given string, it would then follow, that the given string matches into the pattern, otherwise not.

The second method is *Pattern Matching*. Pattern Matching is a method which in turn predefines a pattern in some way and examines whether an object matches into the pattern or not without using a prediction. That means, a string which consists of a text or binary data, is searched for a given character sequence. The simplest form of Pattern Matching is for example the use of a search engine where can be searched after a given criteria[6].

[6]http://perldoc.perl.org/perlre.html

2 Related Work

This chapter will provide a detailed view into works that deal with the problem of security in computer systems, and give an insight into current developments. First of all, the meaning of the term intrusion is explained, so that the fundamental approaches, Intrusion Detection System (IDS) and Intrusion Prevention System (IPS), can be presented. Afterwards, Honeypots and ACCEPT are discussed as alternative approaches to IPS and IDS. All approaches are examined on basic principles, design approaches and implementation examples in practice and each approach is reviewed for advantages and disadvantages.

2.1 Intrusion

Before presenting protective and detective mechanisms against intrusions, violations and security breaches, the term *intrusion* should be defined. According to Heberling, Levitt and Mukherjee, an intrusion is a set of actions that aim to compromise the confidentiality, integrity, and availability. This means that an intrusion is a violation of the security measures of an information system in order to gain access to confidential information assets [BH02]. Confidential information includes, for instance, credit card numbers, sensitive customer or internal corporate data, research findings, or software codes.

Confidentiality, integrity, and availability are defined in chapter 1.3 and are known as protective goals. The aim of protective goals is ensuring the security of confidential information assets as well as of the information system resources. Confidentiality describes the protection of sensitive information against unauthorized access of third parties. Integrity describes the protection of information against unauthorized modification of third parties. Availability describes the disposability of resources and services for legitimate users.

2.2 Intrusion Prevention System (IPS)

Today, mostly preventive security mechanisms are used in practice. Prevention mechanisms are supposed to protect information systems resources and confidential information assets against malicious attacks, security breaches, and intrusions. These prevention mechanisms are known as *Intrusion Prevention Systems (IPS)*.

IPS focuses on the transport layer of the ISO/OSI-7-layer model and provides not only a protection on network level by monitoring the whole data traffic against unauthorized activity; but, they are also used to block attacks before an intrusion occurs. The ISO/OSI-7-layer

reference model is a design base for communication protocols and computer networks. OSI stands for Open System Interconnection and was standardized by International Organization for Standardization (ISO) in order to provide communication standards[1].

In some literature, an IPS is referred to as a bouncer who stands in front of a door and pays attention so that unauthorized or uninvited persons do not enter. In general, IPS can be distinguished in *Host-based Intrusion Prevention System (HIPS)* and in *Network-based Intrusion Prevention System (NIPS)*[2]. Modification of system resources like trojan horses and backdoors, buffer-overflow exploits and access to e-mail contact lists are some of the threats that are aimed to be addressed by HIPSs. HIPSs are installed on individual hosts in order to protect the computer networks against these malicious behaviors. NIPSs are located on the network level of an information system and monitor the incoming data packets for malicious behavior in order to react with countermeasures, if a malicious behavior or suspected data packets are detected [SB08]. Possible countermeasures could be to send a notification mail, block the whole data traffic or delete malicious packets.

Next, some examples of currently used IPS are shown:

Firewall is the most common and the most used preventive mechanism of organizations in practice. A firewall, located at the connecting point of the Internet and intranet, as shown in Figure 2.1 [Kap07], is a preventive protection mechanism that, through the application of certain rules, has the goal of analyzing and filtering the data traffic on the network. Rules are defined by the user during the configuration of the firewall, which will then decide whether a data packet is legitimate or not [BH02]. Potential threats that could be harmful for the data and components of a network are detected and the forwarding of these packets is not allowed.

Types of firewalls are packet filtering firewalls and application filtering firewalls. Packet filtering firewalls are the simplest form of firewalls based on the concept of a router at the IP level [BH02]. According to [SB08] a packet filtering firewall uses a set of rules in order to allow or forbid incoming and outgoing IP packets. A packet filtering example is shown in the following Table 2.1:

action	ourhost	port	theirhost	port	comment
block	*	*	spigot	*	we do not trust these people
allow	our-gw	25	*	*	connection to our SMTP port

Table 2.1: Example Rule of a Packet Filtering Firewall

This example of a packet filtering rule depicts that an incoming mail is allowed only to a gateway host and SPIGOT, an external host, is blocked. The '*' depicts, that there is no restriction.

[1] www.elektronik-kompendium.de/sites/kom/0301201.htm
[2] www.securitymanager.de/magazin/die_tuersteher_intrusion_prevention_systeme.html

Application filtering firewalls, also referred to as application-level gateways, are firewalls located at the application level of the ISO/OSI reference model. The task of an application filtering firewall is to monitor, analyze and filter the communication between client and host at the application level [Kap07].

"You can't fight the fire from behind the firewall" [Thr12]. This statement from ThreatMetrix, a company providing cybercrime protection, reflects the limitations of a firewall. A firewall is only able to protect the system from being attacked by the outside attacker, while an inside attacker can easily bypass these security measures [SB10]. The detection of complex DoS attacks, for example an attack on port 80 (HTTP server) or port 25 (Mail server), is not possible. Therefore, only using traditional firewalls is not sufficient to ensure the security of an information system [MPB+12].

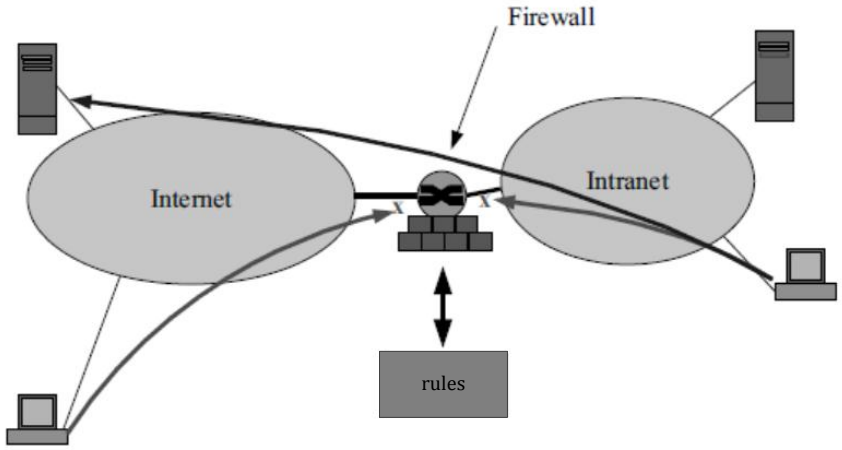

Figure 2.1: Example of a Firewall

Fail2ban is a preventive mechanism that analyzes logfiles for unauthorized access attempts. Logfiles are generated automatically from the web server and the underlying operating system. If there is a malicious indication[4], for instance a repeated wrong password entry or attempted access, the IP address is temporarily blocked for a specified period. Existing firewall rules are updated, or new rules are generated in order to block further login attempts.

Access control models are a different way to secure the information system against unauthorized access. In contrast with known prevention mechanisms like firewalls or fail2ban,

[4]www.fail2ban.org/wiki/index.php/MainPage

access control models monitor and control the access of certain resources in order to ensure the integrity, availability and confidentiality of an information system.

A known access control model for the operating system Linux is the approach of Security-Enhanced Linux (SELinux). SELinux is a security concept based on mandatory access control (MAC). MAC is a control policy [Upa11] that acts as an additional access control. The approach is based on a central instance and a security label. The central instance is referred to as security policy, and the security label as security context. The system assigns objects and subjects of an information system to a security label. An access matrix is generated from the security server, which is located in the kernel, with the values of the policy and the label. According to this access matrix the access can be allowed or denied. In other words, SELinux decides whether a resource is allowed to access a certain target or not[5].

TrustBox is a security architecture to prevent data breaches. A safe environment with virtual machine monitors (VMM) and a framework that restricts certain accesses or actions is established. A differentiation is made among sensitive and insensitive data. Access to sensitive data is only allowed by the trusted virtual machine. This is a platform-independent virtual machine that contains security policies and has applications like email client and Microsoft Office Suite. Applications such as web browsers or instant messages run on the untrusted virtual machine, as they have direct Internet access, and could thus cause potential safety hazards. Sensitive data is stored in Storage Area Network (SAN), and the access is secured through an encrypted and authorized communication like IPSpec network tunnel [SFSF11].

Figure 2.1 [SFSF11] shows how complicated and impractical an environment can be in which the main goal is to prevent data breaches and securing the whole system.

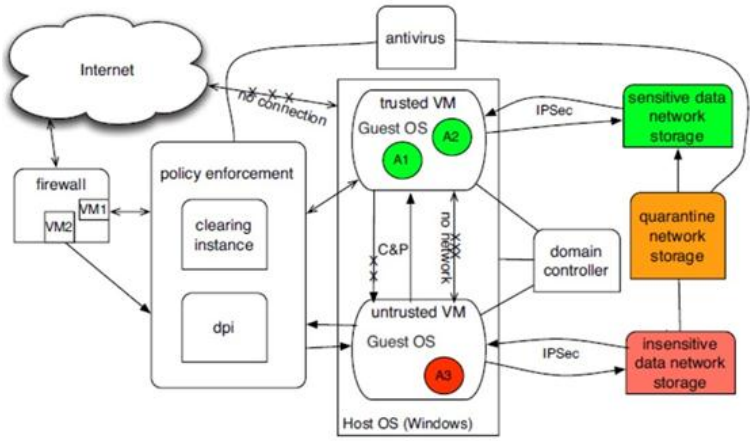

Figure 2.2: Architectural View of TrustBox

[5]www.heise.de/ix/artikel/Gut-bewacht-506652.html

2.3 Intrusion Detection System (IDS)

The increasing number of security incidents shows that the sole use of Intrusion Prevention Systems (IPS) is not sufficient to ensure the security of an information system [Mei07]. The configuration and the maintenance of firewalls, as well as access control models, are too complex. These are main drawbacks of prevention mechanisms. Therefore, preventive mechanisms need to be complemented by detective mechanisms.

These significant drawbacks of IPS require further considerations so as to solve this security problem. Besides preventive mechanisms, there are detective ones whose aim is to constrain and remove damage caused by security breaches and identify the intruder.

An approach that handles this situation are *Intrusion-Detection Systems (IDS)*. According to [SB08], an *Intrusion Detection* is a security service that monitors and analyzes the information system resources in order to detect and provide real-time or near real-time warning of attempts of unathorized third parties. An intrusion, as described in the previous section, generally means that an unauthorized malicious user takes advantage of a vulnerability in the information system and violates the confidentiality, availability, or integrity of the resources of an information system.

An intrusion detection system operates like an alarm system. If an invasion is detected by the IDS, an administrator will be informed via email, SMS, or pager, who then decides how the problem will be handled [Hil01]. The aim of such a detection system is to monitor the underlying information system resources and the computer network against anomalous behavior, minimize security breaches, identify malicious intruders, and detect existing vulnerabilities.

Although IDSes became well known by the end of the 1990s through the usage of vendors like Cisco, Symantec, and ISS, their origin lie in the 1980s. The former concept of intrusion was first presented by James Anderson. His proposal was to protocol audit trails in order to trace abuses and gain information about the intruder and their behavior [Dra04].

Following that idea, Dorothy Denning reported her approach in the paper *An Intrusion Detection Model*. This was the first approach for an intrusion detection model, and based on this, various IDSes are used in today's practice. Denning's theory is based on the assumption that security breaches can be detected by monitoring audit records, and figuring out if there is a deviation in the usual or expected usage pattern. Profiles are used to describe the behavior of subjects concerning objects as metrics, and statistical models. The model does not depend on a certain system, application environment, system vulnerabilities, or kind of security breaches. Therefore, it is referred to as an IDES (Intrusion Detection Expert System). Denning's security model contains basic elements like *Subjects*, *Objects*, *Audit records*, *Profiles*, *Anomaly records*, and *Activity rules* [Den87].

Subjects describe operations that are initiated by a user of the monitored system. A process, or also the system, can be considered as a subject that operates instead of a user or a user group. Operations are triggered by subjects through user commands.

Objects are resources like programs, files, records, commands, and devices that are generated by users or programs.

Audit records are reactions from the monitored system caused by subjects that invoke operations, such as user login, command execution, and file access. Every record consists of six elements: <Subject, Action, Object, Exception-Condition, Resource-Usage, Time-stamp>

Profiles use metrics and statistical models to define the behavior of subjects relating to objects. Every activity profile describes normal behavior of subjects or a group of subjects regarding objects or a group of objects. Each profile characterizes some specific behavioral aspects of the considered subjects. An activity profile consists of ten elements: <Variable-Name, Action-Pattern, Exception-Pattern, Resource-Usage-Pattern, Period, Variable-Type, Threshold, Subject-Pattern, Object-Pattern, Value>

Anomaly records describe an abnormal behavior of a user. If an audit record is created or a period ends, IDES refreshes activity profiles and searches with activity rules for potential abnormal behavior. Once an abnormal behavior is noticed, an anomaly record is created that describes the abnormal behavior. An anomaly record consists of three elements: <Event, Time-stamp, Profile>

Activity rules are operations that should be performed when an audit record or anomaly record is created or a period terminates. Activity rules can be sorted in audit-record rule, periodic-activity-update rule, anomaly-record rules, and periodic-anomaly-analysis rule.

Denning's concept can be interpreted as a rule based pattern matching system. Audit records are compared with existing profiles, and profiles decide which rules should be used to refresh profiles, monitor abnormal behavior, and protocol identified deviations. As mentioned before, security breaches can be detected by monitoring the target system against deviation from normal usage. Relations between attack types and deviation of normal usage are classified in *Attempted break-in*, *Successful break-in*, *Penetration by legitimate user*, *Leakage by legitimate user*, *Inference by legitimate user*, *Trojan horse*, *Virus*, or *Denial-of-Service* [Den87].

This type of intrusion detection is defined in practice as *anomaly detection*. In some literature, it is referred to as behavior-based detection. In order to ensure the security of an information system, another approach is introduced and referred to as *misuse detection*. In literature, it is sometimes called knowledge or signature-based detection. For the following chapters, we use the terms anomaly detection and misuse detection.

2.3.1 Anomaly Detection

The approach of anomaly detection is based on the hypothesis that anomalous behavior of the information system or the user itself indicates a security breach. Anomalous behavior can be defined as a deviation of normal or expected behavior [Deb02], which is measurable in different ways and stored in reference information. This information about normal or

expected behavior is employed to match against actual behavior. A deviation of normal or expected behavior is considered as a security breach and an alarm is triggered.

In *Intrusion Detection effektiv!* [Mei07], Michael Meier describes an approach on how reference information is created. In order to generate reference profiles, the system requires a phase to learn the normal behavior. Methods for this learning phase are, for example, neural networks, statistical methods, or decision trees. Behavior of the user may change with the time or the period, so this process should be repeated regularly. During the learning phase of the system, it is possible that a security breach may occur. In consequence of that, reference profiles could include possible security breaches that are not recognized and each deviation is interpreted as an attack; as a result, false positives are increased.

False positives are results specified incorrectly as positive. That means, a behavior was mistakenly identified as an attack. False alarm is triggered by an IDS. **False negatives** are results specified incorrectly as negative. That means, a malicious behavior was mistakenly not identified as an attack.

Although anomaly detection has already existed since the 1980s, this approach is not often used in practice, because there are several disadvantages that complicate the usage of anomaly detection:

- high false positive rate results in high costs;
- results of anomaly detection are not clear enough;
- detected anomalous behavior cannot be automatically seen as a security breach;
- before an action against the security breach can be done, the detected anomaly behavior must be evaluated by a security expert;
- definition of normal behavior is complex and to classify anomalous behavior as a general term is difficult.

Besides these disadvantages, the main advantage of this approach is the ability of detecting unknown and unexpected attacks. Other advantages include:

- information about the structure of the attack and the system is not required;
- can detect attackers that are logged in with a 'false'-account;
- maintenance of a signature database is no longer required;

2.3.2 Misuse Detection

The approach of misuse detection is based on the detection of attacks by using known signatures. For each attack a signature is created. A signature is a byte-sequence attack pattern that uniquely identifies an attack and is stored in a signature database. During the analysis, signatures in the database are compared with audit data. According to [Deb02]

audit data describes information provided by a system concerning its inner workings and behavior.

For example, *syslog*[6] is a service which provides audit data from the underlying operating system like UNIX and others. Syslog gleans produced messages and faults in form of a text string by background processes like services, server or daemon, adds a time stamp and the application name, and writes it to a log file. If a match is found, an alarm is triggered and countermeasures are executed by a information security officer or administrator.

Due to the simple implementation, misuse detection is widespread and often used in practice. One of the most popular signature-based intrusion detection systems is *Snort*. The reason for its popularity is the accuracy and the structure of the results. There are several advantages to this detection system:

- low false positive rate, if signature is precisely defined;
- widespread and simple to implement.

The disadvantage of this detection system is that only known attacks, whose signature is stored in the database, can be detected. Other disadvantages include:

- if signature is not precisely and correctly defined, it will raise false positives and false negatives;
- precisely defining a signature is not easy and requires experience;
- signatures are stored in a database and, if this is not updated well, the analysis is not effective;
- maintenance of a database is an additional effort and fraught with risk.

Depending on the source of the audit data, this detection system can be distinguished in *Network-based Intrusion Detection (NIDS)* and *Host-based Intrusion Detection (HIDS)* in order to provide segregation of duties. If the data source contains network traffic material, it is recommended to use network-based IDSs; if the data source is based on host material, it is recommended to use host-based IDSs.

Network-based Intrusion Detection System (NIDS)

Through the rapid development of the Internet, network attacks like DNS spoofing, TCP hijacking and port scanning are becoming more popular and an IDS which specializes on network is needed. The core idea of network intrusion detection is based on monitoring network traffic and packets.

[6] https://tools.ietf.org/html/rfc5424

Organizations use this kind of detection for monitoring their intranet communication. Sensors are used to intercept network traffic and compare it with existing signatures. If there occurs a discrepancy between network configuration or an unexpected event, this will be sent to an IDS server and stored there in a database. Analysis and possible countermeasures are taken by the IDS server.

An advantage of a network-based IDS is that it is possible to monitor the whole system with only one computer. Therefore, it is a central approach and demands less administrative and financial effort. This is an advantage compared to host-based IDSs, in which a host-based IDS must be installed and maintained on each host that has to be monitored.

Encryption is one of the biggest challenges that need to be managed by network-based IDSs. The basis of most internet or intranet connections is Secure Sockets Layer (SSL). SSL is a network protocol that enables secure data communication between networks. Network-based IDS cannot decrypt and analyze encrypted data. In this case, it is not possible to detect attacks and malicious intruders. A network-based IDS should be able to understand common network protocols like TCP/IP, HTTP, SMTP, or POP [Bie05].

Snort is an open source product and a well-known network-based IDS. Snort can be both network intrusion detection and network intrusion prevention. It can be configured in three modes, the simplest of which is the sniffer mode. Packets from the network are read and displayed as byte sequences in the console. Packet mode stores packets on the hard disk, and further analysis can be performed from there. Network intrusion detection is the most complicated mode. It monitors network traffic, and rules are defined to control packets that might contain possibly malicious content.

Snort is a rule-based approach. Rules have a simple structure shown in Figure 2.3 [SB08], consisting of two main components: header and option(s).

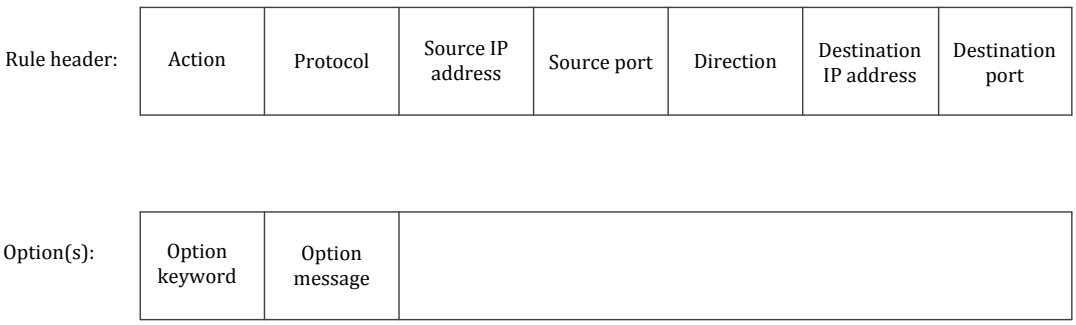

Figure 2.3: Rule Structure of Snort

The rule header contains a set of fixed elements: **action** stands for a reaction if a packet is found that suits the rule criteria, *protocol* stands for the type of the recognized protocols, currently TCP, UDP, ICMP and IP, *source IP address* stands for the source of the packet, *source*

port stands for the source port, *direction* stands for the direction of monitoring, direction options are unidirectional (->) and bidirectional (<->), *destination IP address* stands for the destination of the packet and *destination port* for the destination port.

The option(s) contains one or more rule options: *option keyword* stands for the option and *option message* stands for the specification of the option [SBo8].

A rule example of Snort is shown in Figure 2.4 [Eck12]:

```
alert TCP any any   ->  192.168.1.0/24 143 (content: "|90CB C0FF FFFF|/bin/sh";\
                                                msg:"IMAP buffer overflow!";)
```

Figure 2.4: Rule Example of Snort

First line defines the header. *Alert* is a keyword that triggers and logs an event if a packet matches with the specified values in the rule. Keyword *content* determines according to which pattern it should be looked after. One or multiple options define the warning message with *msg*. This rule scans all packets from any IP address with any source port, where the destination IP address is in the range between 192.168.1.1 and 192.168.1.255. Port is 143. Rules should be defined in one row, otherwise snort may return an error or interrupt the analysis.

Thus, Snort is a rule-based approach, and it is possible to define several rules and events to detect attempts of attacks in network traffic. Attacks can be buffer overflow, port scans, CGI, and OS fingerprinting. Still, Snort has improvement potential, because defining rules requires deep knowledge. Next to defining rules, configuring of Snort is also important because misconfiguration can lead to false positives or false negatives.

Host-based Intrusion Detection (HIDS)

Host-based intrusion detection is based on monitoring system and application data from an individual host. HIDSs receive audit data from the host, for instance log files, executed system calls, or file attributes. If anomaly detection is applied, changes of file attributes (owner, group, or permissions), file content, or path will be monitored. If misuse detection is applied, system or log files will be checked for predefined patterns.

The advantage of this kind of detection system is that it triggers less false positives than network-based IDS. Additionally, a host-based detection system can detect attacks over a terminal, encrypted connections, or malicious programs like Trojan horses. Disadvantages of this approach are additional costs caused by installation and administration of all monitored hosts and Denial of Service (DoS) cannot be recognized [Bie05].

Tripwire is a well-known open source host-based IDS for Unix. It verifies the integrity of important system files and directories by storing properties of files (size, modification date, or hash value) in a database. Checking for modifications of file systems at regular intervals can prevent possible attacks. If an attack is detected, an administrator will be notified and a reaction or countermeasure is initiated. The main drawback of this application is that integrity checking by md5 and sha checksum for each file takes too much computation time. Another drawback is that intruders who only read data and do not change it are not recognized. Furthermore, it is essential to keep the tripwire database updated. This is a great effort if the database is very large [Eck12].

2.4 Alternative Approaches

Besides IDS and IPS, there are alternative approaches with the aim of securing an information system. One of the alternative approaches are Honeypots and Honeynets which simulate a supposedly interesting system to lure attackers and a second approach is ACCEPT which uses a hypervisor to detect, analyze and handle security anomalies. Both approaches are discussed in the following chapter.

2.4.1 Honeypot and Honeynet

Honeypots are decoy systems [SBo8] that distract attackers from actual valuable data such as addresses, documents, and security-related firm-specific information. The core idea is to observe and analyze attacks, in order to gain information about the attackers and the attack types. Honeypots do not manage access and access rights; therefore, every access is a potential attack that should be analyzed. If an attack is suspected, the honeypot triggers an alarm. The attacker and the attack method are investigated in more detail. With additional information about the attacker and the used methods and techniques, security mechanisms can be improved and developed.

Advantages of honeypots as a security mechanisms are:

- long-term use can serve to improve the security mechanisms by analyzing identified attacks, and be used for protection against future attacks;
- supplement to Intrusion Detection or Prevention Systems, since valuable information about attackers and attack method can be found;
- suitable in demilitarized zone (DMZ) or in firewall, since the attacker would have no other attack possibility except the Honeypot.

Disadvantages of honeypots as a security mechanisms are:

- high-cost and increased complexity;
- lack of practical experience, usually used only for research;

A distinction is made between fields of application, such as production honeypots and research honeypots. Production honeypots process unauthorized access. Every access is considered as an attack and and therefore triggers an alarm. Production systems may thereby protect against attacks. Research honeypots are used for studies about the attacks and the technique of the attacker. Depending on the desired target, planned vulnerabilities will be left open to analyze attacks, for example by worms. Multiple honeypots are called Honeynets. They have the same functionality as honeypots [SP04].

2.4.2 ACCEPT

ACCEPT is a relatively novel prevention approach based on the core idea of preventing the security of an information system with the usage of a virtual machine monitor (hypervisor). The aim of the approach is to detect, analyze, and handle security anomalies in virtualized computing systems. Therefore, sensors monitor security anomalies (events) on different layers of a virtualized computing system. Complex Event Processing (CEP) is used to abstract, correlate, and aggregate events in order to detect attacks, perform actions, and reduce false positives. The main drawback of ACCEPT is that this is currently only a theoretical approach, without a significant implementation [LB12].

3 Motivation and Concept of a Breach Detection System (BDS)

3.1 Motivation

Through the change of the IT environment in recent years, a common requirement of organizations, private persons, or governments is the security of their confidential information assets. Every day, huge amounts of sensitive and confidential data is created and transmitted over the Internet or intranet without one being aware of what actually happens to these data. Credit card numbers, sensitive customer or internal corporate data, research findings, or software codes are valuable information that has to be protected against third parties.

The Related Works in chapter 2 gave an insight into existing protection mechanisms and approaches which attempt to secure the computer systems. The protection mechanisms can be roughly divided into two approaches, namely Intrusion Prevention System (IPS) and Intrusion Detection System (IDS). A typical IPS approach is a firewall. A firewall is a preventive protection mechanism that, through the usage of certain rules, has the goal of analyzing and filtering the data traffic on the network. Rules are defined by the user during the configuration of the firewall, which will decide whether a data packet is legitimate or not [BH02]. The IDS approach distinguishes between anomaly (behavior-based) detection and misuse (signature-based) detection. Anomaly detection generates profiles of the normal usage behavior, and a deviation of the normal usage is considered as security breach. Differently, misuse detection maintains a signature database with the signature of attacks, which are compared with audit data during the analysis.

Despite being common practice, IPS and IDS suffer from several significant drawbacks. A drawback of prevention mechanism is the definition of precise enough rules. The more imprecise the rules, the worse is the detection of malicious attacks. This means that defining rules that allow not too many, but also not too few data packets requires prior knowledge that many users might not have. A further drawback is that the profiles are always updated, considering that the more recent the profiles are, the more efficient the detection of malicious intruders is. Drawbacks of detection mechanisms are inflexible behavior of misuse detection because only stored signatures in a database are detected and high number of false alarms of anomaly detection.

The goal of this diploma thesis is to conceptualize a novel method for Security Breach Detection through File Access Monitoring and Pattern Recognition, design and implement a Breach Detection System (BDS) framework to overcome the drawbacks of current IDS and IPS.

3.2 Concept Overview

The core idea of this novel approach is a method for Security Breach Detection using File Access Monitoring and Pattern Recognition. A BDS shown in Figure 3.1 was developed and able to detect and analyze security breaches at the operating system level. A particular focus is on the file operations. The concept can be roughly described as follows:

A *Sensor* continuously monitors the system behavior in a controlled state where the system is newly installed and not connected with the Internet, and provides this information in form of an event to the Pattern Engine. An event is a set of parameters used for describing the properties of a process. The *Pattern Engine* receives all events and creates a pattern in the pattern creation phase of the learning phase. The *Event Processing Matrix* and the *Access Revocation* define actions considered to be the result type of the pattern in the pattern definition phase of the learning phase. Patterns are used to monitor a process that accesses a certain file and determines the legitimacy of the file operation.

While the system is running in a normal environment (not secured), new events are created from the current system behavior. These new events are compared in the recognition phase of the application phase with existing patterns in order to evaluate the system behavior.

In case of an illegal system behavior, an action is executed through the Access Revocation, which in turn triggers an action like sending a notification mail. In case of a valid system behavior, the new event is added to the corresponding pattern in the adjusting phase of the application phase.

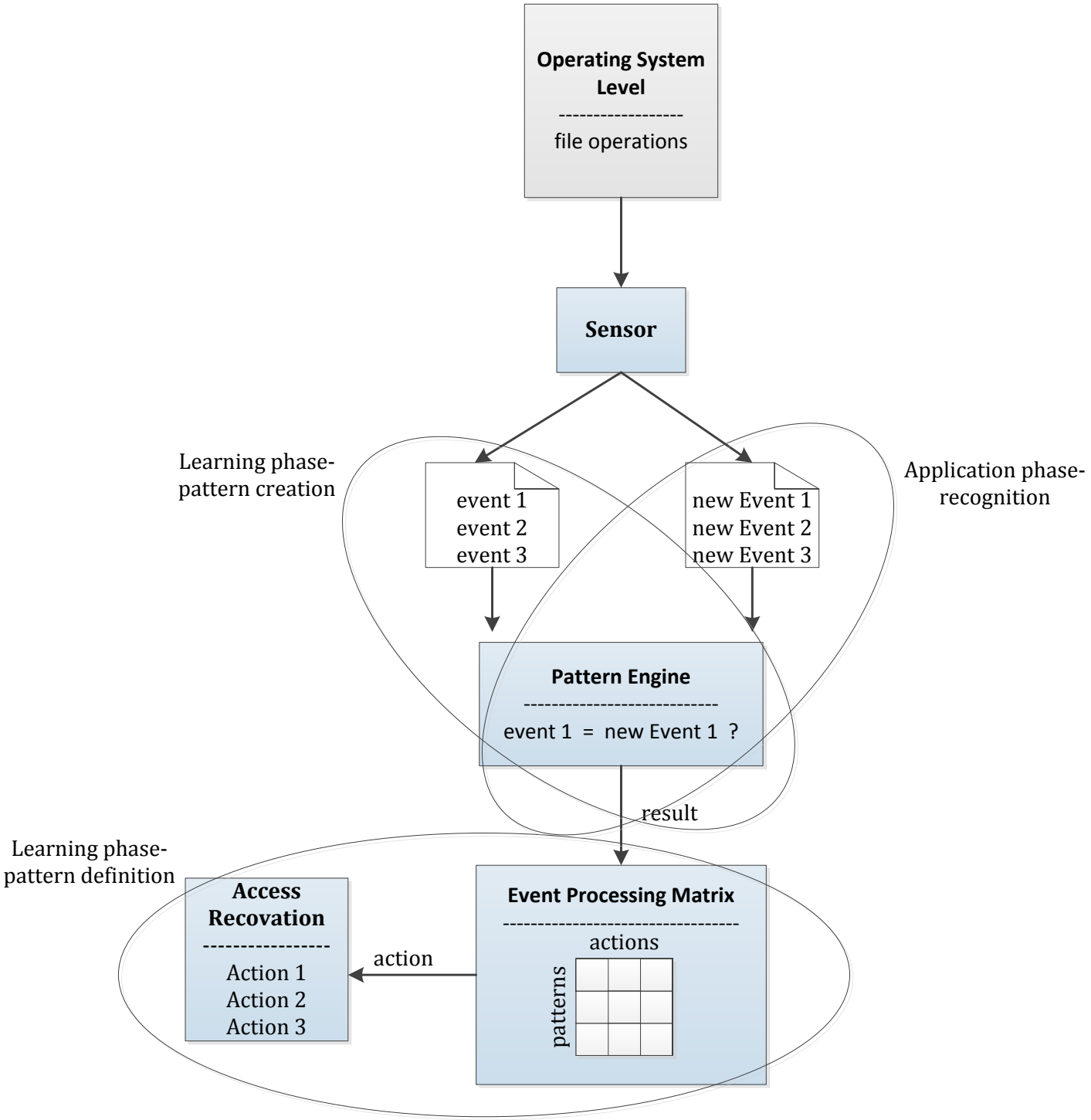

Figure 3.1: Overview of the Breach Detection System Architecture

3.3 Description of the Key Terms

Event

David Luckham defines an *event* as an object that is a record of an activity in a system. The event signifies the activity, and may be related to other events [Luco2]. We use the term event here to denote the properties of a process. A process specifies a running program. Every program that is started is a process. For instance, a user opens a website or checks the inbox. An event record consists of five elements: *processID*, *userID*, *operation*, *path*, *date*.

It is quite challenging to choose right parameters to uniquely describe a process or an action. The reason behind choosing these five parameters is that we assume them to be suitable for this purpose. *ProcessID* describes the number which uniquely identifies each process. *UserID* describes the uniquely assigned user number. *Operation* describes the name of the process. *Path* describes the name of the file. *Date* describes the current date when the event is established. For further researches, it is possible to define more parameters.

An example of an event is shown in Table 3.1.

	processID	userID	operation	path	date
event	10	root	echo	/user/bin/echo	27/09/2013

Table 3.1: Example of an Event

This event shows who has accessed the file echo. Echo is a UNIX command that writes strings to a standard output. A root user runs a process with a processID 10, and accesses the file echo.

System behavior

The term *system behavior* defines the behavior of a system, and depends on various parameters such as behavior of a user, underlying operating system, program execution, or configuration. A differentiation is made between normal or expected and anomalous system behavior. Normal or expected system behavior means that both hardware and software components fulfill the expectations that are defined by certain conditions. Anomalous system behavior can be defined as a deviation of the normal or expected system behavior, caused by anomalies resulting from changes in hardware and software or their configuration.

There are three types of anomalies. The first type of system anomaly is a behavior which is not intended or specified by a specification or configuration, for instance environmental factors such as over-voltage caused by lightning, transmission, or programming errors. The second type of system anomaly is a behavior in which components like libraries, software, hardware, or operating system modules are intentionally extended by additional, harmful functions; an example is Trojan horses. The third type of system anomalies are computer viruses and worms [GW].

File access pattern

Each event described as mentioned before represents properties of a file access. A *pattern* describes one system function, for example *a web server loads a homepage*, or *an e-mail server accesses the inbox of a user* and consists of multiple events which form a point cloud. That means, a pattern represents a set of file access events which is gathered while the system is running.

However, a system does not consist of only one system function; hence, multi-event patterns are needed for describing one or more system functions. File access patterns are used to monitor file accesses and decide whether an file operation was legitimate or not.

3.4 BDS - A Breach Detection System

Every action that is performed by a user or a program leaves a trace in some way. The forensic computer science tracks these digital traces to detect and clarify criminal acts in court [C.F]. Just as log files, for example from a web server where all requests are logged, provide information about the usage of an Internet offer[1].

Keeping this in mind, a novel approach is conceptualized which examines the file operations in order to decide the legitimacy of a file operation and detect illegal system behavior which could be harmful for the security of an information system. Therefore, the proposed BDS shall monitor the system behavior and map it on patterns of file access events. Every action that is executed through a user or a program affects a certain number of files. As described in the introduction chapter, a significant property of the class of Unix operating systems is the widespread term *file*. A file can be a regular file (text, programs and executable files), directories or special device files (disk, webcam or printer). Therefore, file access events are used to derive patterns, and which in turn monitor processes.

To construct a pattern, events are generated by the sensor component through monitoring the system behavior. In order to monitor various system functions, it is necessary to use multiple patterns. That means, by using only one pattern, only one system function can be monitored. A pattern engine component is used to detect deviations between patterns that consist of system behaviors recorded at a certain secure time, and patterns that are generated with current system behavior while the system is running.

The proposed architecture is divided in several phases. Each of the phases represent different parts and functionalities of the program sequence of the system. The next section will describe the several phases in detail.

[1] http://www.e-teaching.org/didaktik/qualitaet/logfile

BDS Phases

The proposed BDS is separated into two phases, which is shown in Figure 3.2. The first phase is called *learning phase*, which in turn is separated into *pattern creation phase* and *pattern definition phase*. The proposed system learns in the pattern creation phase the system behavior and creates patterns from this information and after the automatic part is done, the user defines corresponding actions in the pattern definition phase.

The second phase is called *application phase*, which in turn is separated into *recognition* and *adjusting*. The proposed system creates new events from the current system behavior and compares them with existing events in the recognition phase in order to determine the legitimacy of the file operation and add the new event to the pattern in the adjusting phase in order to improve the patterns.

The reason behind deciding to divide the BDS into two phases is the issue that, at the beginning of the system, there is available neither information about the system behavior, nor events to generate patterns. If it is so, the system cannot differentiate between a valid or invalid event. Therefore, in a first phase, information about the system behavior has to be gathered, and in a second phase the pattern can be further improved.

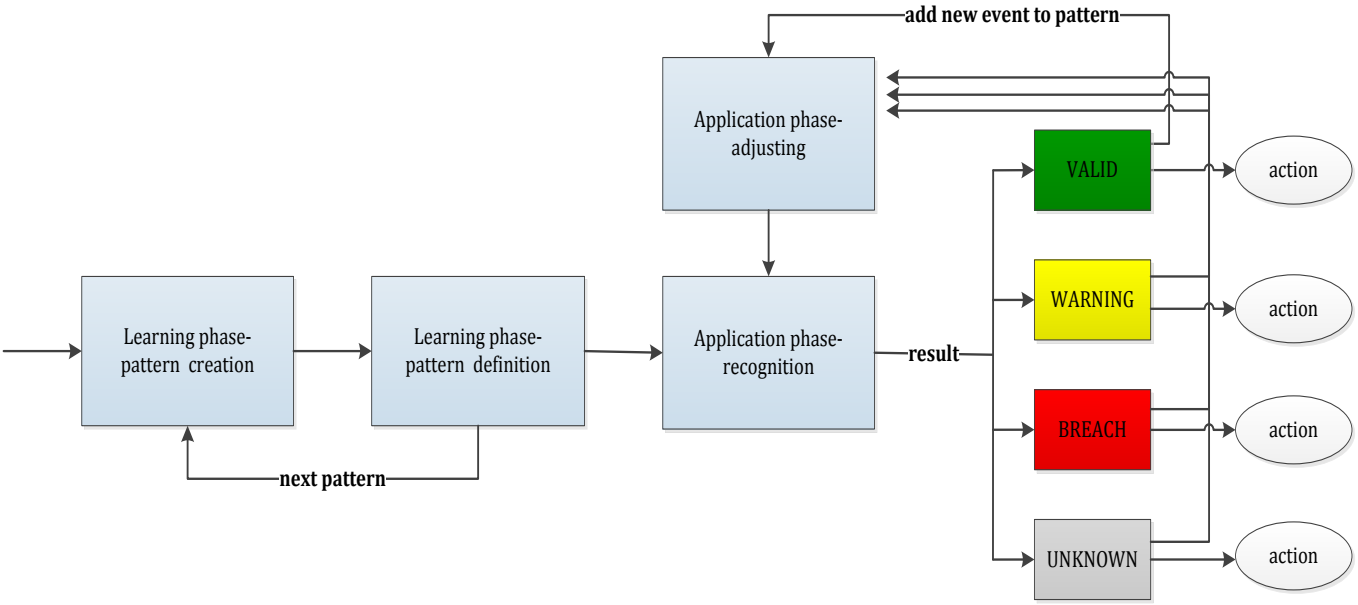

Figure 3.2: BDS Phases

Learning phase

The purpose of the learning phase is the creation of a set of patterns and the definition of corresponding actions. Therefore, the learning phase is separated into two parts, the pattern creation phase and the pattern definition phase. The following sections will describe in detail the functionality of both phases:

The first part of the learning phase is the pattern creation phase, which learns events from the system behavior while the system is running in a controlled state and generates patterns from this events by applying a density function. That means, a pattern is consisted of several events which represents a system function and the density function itself. A pattern is constituted by forming a point cloud of all events in a multidimensional space.

After the automatic part is completed, the second part of the learning phase, the pattern definition phase will be executed. In this phase, a user defines actions which should be executed, on basis of the generated patterns. In order to define an action, the result type needs to be considered. Thus, result types are defined as *VALID* for events which match into a pattern, *WARNING* for events which match only to a certain degree into a pattern, *BREACH* for events which do not satisfy the properties of the pattern and *UNKNOWN* for events where the result is not interpretable. These result types have to associate with actions in order to react against detected illegal system behavior or breached patterns.

The Breach Detection System Architecture shown in Figure 3.1 consists among others of an *Event Processing Matrix* and an *Access Revocation* component. The Event Processing Matrix defines an association-matrix that links actions to patterns. In order to know which action should link to which event, the action has to be defined depending on the result type of an event. Therefore, a general action in form of a notification-mail is defined which will be executed if an event applies. This general action can be adapted with a text documentation or a special action depending on the type of the result and the pattern. For some patterns it will be necessary to define special actions in order to be able to execute actions on a special level.

To be able to monitor several system functions multiple pattern support is required. After defining one pattern in the pattern definition phase, it is possible to repeat the pattern creation phase and define the next pattern.

Application phase

The general approach of the application phase is to compare new events with existing patterns in order to determine the legitimacy of a new event. An event consists of various parameters like *processID*, *userID*, *operation*, *path* and *date*. Considering that a process is accessing not only one but also several files, patterns are generated from various events. A pattern, as described in the learning phase above, is consisted of several events which represents a system function and the density function itself. A pattern is constituted by forming a point cloud of all events in a multidimensional space and each parameter value of the event represents a coordinate value. The application phase is divided in turn in two

phases, the recognition phase and the adjusting phase. Both phases will be discussed in detail in the next section:

After the completion of the learning phase, a set of patterns, each with definitions of actions, are available and the BDS will proceed with the application phase.

The first phase of the application phase is the recognition phase. In this phase, the pattern recognition algorithm compares each new event with existing patterns in order to examine whether the new event is valid or not. Therefore, the purpose is to determine the density of the current position of the new event by using a distance function. To do that, there are a few major computation steps for determining the density of each new event. The major computation steps are as follows:

1. generate the new event
2. determine the density of the new event
    a) calculate the distance for each pattern
    b) calculate the distance for each event
3. decide the legimitacy of the new event
4. add valid event to pattern

The first computation step is to generate the new event. Therefore, the current system behavior is recorded while the system runs in a normal environment which is not secured. After generating the new event, the second computation step is to determine the density of the new event. The density describes the relation between the new event to all events in the point cloud, that means the frequency of events around the new event. This yields the following statement:

the higher the density, the more likely it is that the new event matches with a pattern, and the smaller the density, the more likely it is that the new event does not match with a pattern

This statement indicates that the decision whether the new event is legitimate or not, depends on the frequency of events around the new event. In order to determine the density, the distances have to be calculated by calling the distance function. The distance function computes the distances between the new event to each event of each pattern. To do that, the distance function calls a method where each parameter of the new event is compared with each parameter of the actual compared event in the point cloud.

The distance between the new event and the compared event defines how two events differ from each other. This yields the following statement:

the higher the distance, the more likely it is that the compared events are similar and the smaller the distance, the more likely it is that the compared events are not similar

Considering this statement, the distance of two events is an essential value while determining the density. That means, if the distance value is high, as a consequence the density would be

high, too and if the distance value is small, as a consequence the density would be small, too.

The third computation step is to decide the legitimacy of the new event. Therefore, the computed distances are compared with a given radian. If the distances are greater than a predefined radian, the density is incremented. Depending on the density value, the result type is determined.

After the first phase is finished and the density value is determined, the second phase, the adjusting phase will be begin. If the density of the new event fulfills the characteristics of the pattern, the result type will be set to *VALID* and the new event will be added to the existing patterns in order to continue improving the patterns. Otherwise, the new event will be set to the corresponding result type and adjust to the existing patterns.

4 Implementation

In this chapter, the architecture of the BDS will be designed and implemented, which realizes the concept described in the previous chapter. The prototype is executed in Java and uses web service for client-server communication. The description encloses procedures, methods, and results as well as problems arising during realization.

The chapter will give technical details on the prototype and how the client-server communication and the web service are realized. Afterwards, an overview of the structure and components of the proposed BDS will be given. This includes descriptions of the components, functionalities of the BDS, and the development environment.

4.1 Client-Server Model

Architecture

This prototypical implementation of the BDS is realized with the web service technology. Web services are associated with the concept of Service Oriented Architecture (SOA). SOA is an architectural style that offers different services to a client, in form of methods and functions [Ley11].

A web service provides functionalities and services for several applications, which can be accessed over the network. There is no user involvement; therefore, a web service is represented as an application-to-application communication, as opposed to a website, with a human-to-application communication [Ley11]. Platform independence, scalability, reusability, integration with other systems, and the usage of common standards such as HTTP and XML are some benefits of web services over other architectural models. They are able to provide services to the outward through an interface description language, called Web Service Description Language (WSDL). A WSDL document describes the methods and functionalities, by which the service will be addressed, as well as the data format that a web service is offering.

Another way to realize a web service is the RESTful web services technology. REST stands for Representational State Transfer, and describes an architectural model rather than a protocol. There is no remote procedure call and argument passed in an XML document; methods such as POST, PUT and GET are used for request and response, which are transmitted over a stateless client-server protocol.

Communication

For the proposed BDS, we decided to realize a SOAP-based web service shown in Figure 4.1. SOAP is a standardized network protocol able to exchange data between systems. Initially, it stands for the acronym Simple Object Access Protocol. Since 2003, SOAP is used as SOAP without the acronym. SOAP uses HTTP to access the client through a network, Internet, or intranet, and invokes their objects and methods. The technology of Remote Procedure Call (RPC), in which methods are invoked on remote servers to pass arguments and return values, is used. The parameters and return values are described in the WSDL document [Ley11]. For implementing and deploying the web service, a java-based technology, Java API for XML Web Services (JAX-WS), is used. JAX-WS provides a platform that facilitates the development.

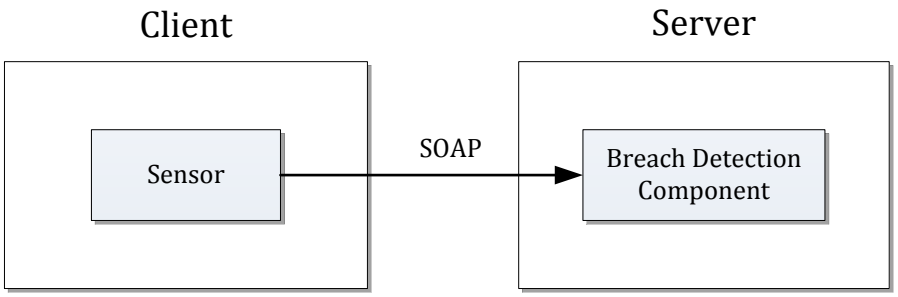

Figure 4.1: Client-Server Communication over a SOAP Protocol

Web service implementation

To provide a web service as a service on the network, the following steps are needed:

1. *Web Service Endpoint Interface*: First, an interface with the Java Annotation @WebMethod is implemented. The @SOAPBinding (style = Style.RPC) indicates that this class should be bound by a SOAP protocol, and the communication by a remote procedure call.

```
1 @WebService
2 @SOAPBinding(style = Style.RPC)
3 public interface BDCInterface {
4     @WebMethod
5     void notifyEvent(Event event);
6     @WebMethod
7     void notifyPoint();}
```

Listing 4.1: Web Service Endpoint Interface

2. *Web Service Endpoint Implementation*: Second, the implementation of the actual service is realized. Therefore, the Java Annotation @WebService is denoted with the implemented endpoint interface from step 1.

```
@WebService(endpointInterface = "com.breachDetection.ws.BDCInterface")
@SOAPBinding(style = Style.RPC)
public class BDCImplementation implements BDCInterface {
    public void notifyEvent(Event event) {
        patternengine.processEventLerningphase(event);
    }
    public void notifyPoint() {
        patternengine.processEventApplicationphase();
    }}
```

Listing 4.2: Web Service Endpoint Implementation Class

3. *The Endpoint Publisher*: With an endpoint publisher it is possible to publish the implemented service to the outward. A WSDL file is created automatically by publishing the endpoint publisher. For testing purposes, the service can be called up with the URL: http://localhost:8080/PublishService/BreachDetectionSystem?wsdl"

```
public class PublishService {
    public static void main(String[] args) throws InstantiationException,
        IllegalAccessException {
        Endpoint endpoint = Endpoint.publish
        ("http://localhost:8080/PublishService/BreachDetectionSystem",
                        new BDCImplementation());}}
```

Listing 4.3: Endpoint Publisher Class

4. *Web Service Client*: The web service client is the application that accesses the previously published web service. Therefore, the URL which is the URL from the WSDL file that was published in step 3 and QName which is the qualified name of the service, has to be initialized.

```
public Sensor() {
    URL url = new URL(
        "http://localhost:8080/PublishService/BreachDetectionSystem?wsdl");
    QName qname = new QName("http://ws.breachDetection.com/",
                    "BDCImplementationService");
    Service service = Service.create(url, qname);
    BDCInterface bdcInterface = service.getPort(BDCInterface.class);}
```

Listing 4.4: Web Service Client

4.2 Components of the proposed Architecture

The class diagram in Figure 4.2 gives an architectural overview of the classes that have been implemented, and represents the relationships between the classes. The BDS consists of five main classes and can be roughly described as follows:

The *Sensor* component can be seen as a container of information, which continuously delivers information to the Breach Detection Component. The *Breach Detection Component* receives the input from the Sensor and passes it to the Pattern Engine Component. The *Pattern Engine Component* is a significant component in this proposed architecture because it implements the pattern recognition algorithm. The *Event Processing Matrix* defines an association-matrix that links patterns to corresponding actions. The *Access Revocation Component* triggers different actions depending on the severity and nature of the received event.

The next section will discuss in detail the architectural structure and the underlying data structure of the implemented components.

Sensor

Sensor is the component which represents the client-side of the web service. Considering this, a web service is an application-to-application communication, and the term client represents a server. In order to avoid misunderstandings, we use the term client in this case. This component has the task of gathering information about the system behavior during file operations, and passing it by invoking a method that is implemented in the Breach Detection Component.

Breach Detection Component

Breach Detection Component is the component which represents the server-side of the implemented web service. This component provides the connection between the Sensor Component and the Pattern Engine Component. It executes the service that is accessed by the Sensor Component through the implemented methods.

Pattern Engine

Pattern Engine Component implements the pattern recognition algorithm. The task of the Pattern Engine Component is to compare two events and determine the legitimacy of the file operation. In order to detect a deviation of the system behavior or illegal system behavior, a result is defined. There are four result types: *VALID*, *BREACH*, *WARNING* and *UNKNOWN*.

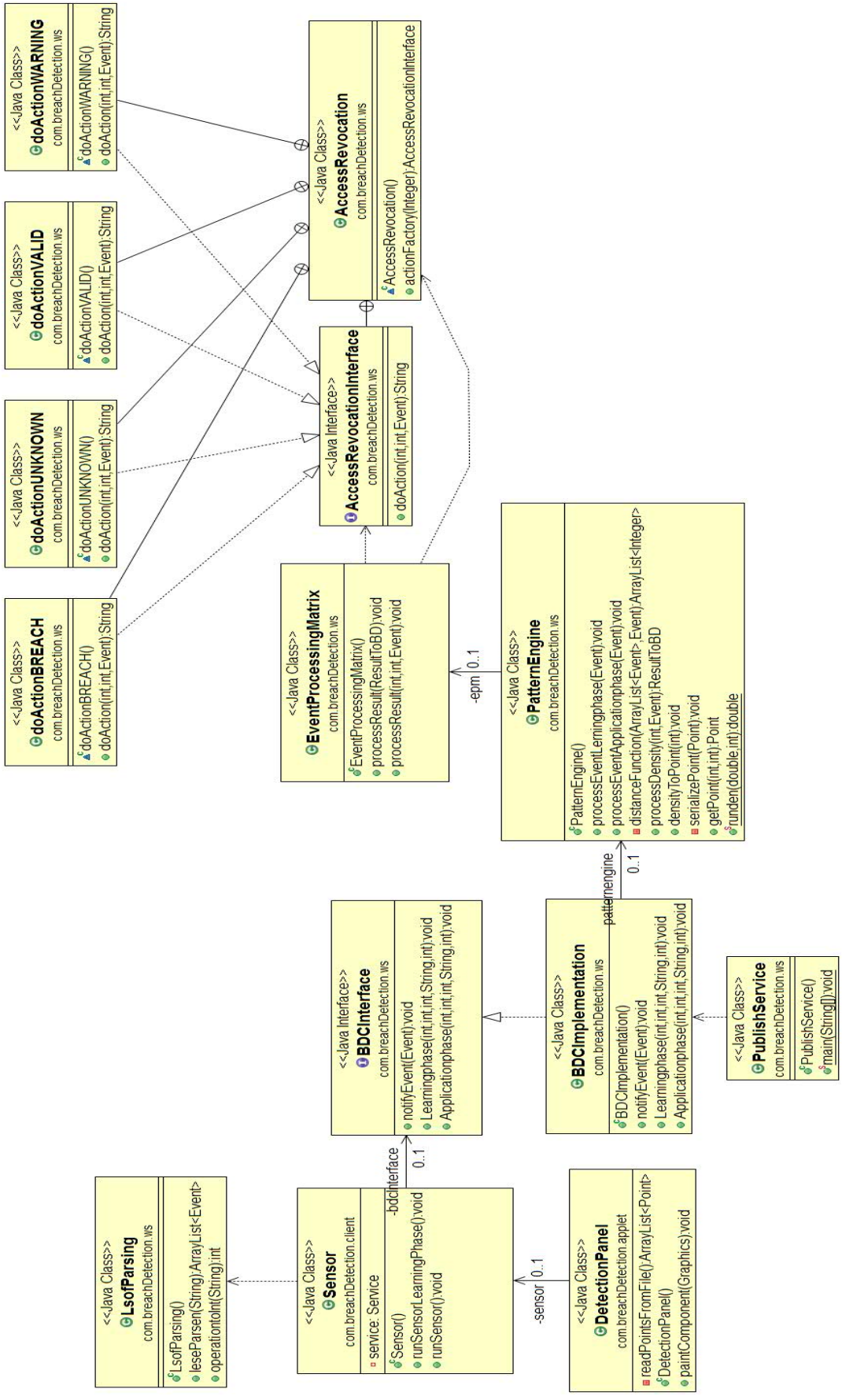

Figure 4.2: Class Diagram for the Implementation of BDS

Event Processing Matrix

Event Processing Matrix, shown in Table 4.1, defines an association-matrix that links patterns to corresponding actions in order to achieve a certain security level. Columns of the matrix represent the actions that are executed depending on the result of the Pattern Engine Component. For instance, actions could be to send a notification mail, or block a malicious user, or shutdown the server by serious security breaches. Rows of the matrix represent the patterns. For instance, a pattern could be that a web server loads a homepage, or a mail server accesses the wrong user directory.

patterns/actions	send notification mail	block user	shutdown server	.....
mail server accesses wrong user directory	x			
web server loads home page		x		
...				

Table 4.1: The association-matrix Table

The underlying data structure for this prototypical implementation of an association-matrix is a nested hashmap, shown in Figure 4.3. A hashmap is an associative storage which keeps a key with a value. It is ideal for storing a set of unsorted elements and making them available quickly over the keys. The internal hashing method is fast[1]. A nested hashmap is a hashmap linked to another hashmap or multiple hashmaps. The first hashmap stores the identification number of the patterns and the second, nested hashmap stores the result typ defined in the Pattern Engine Component.

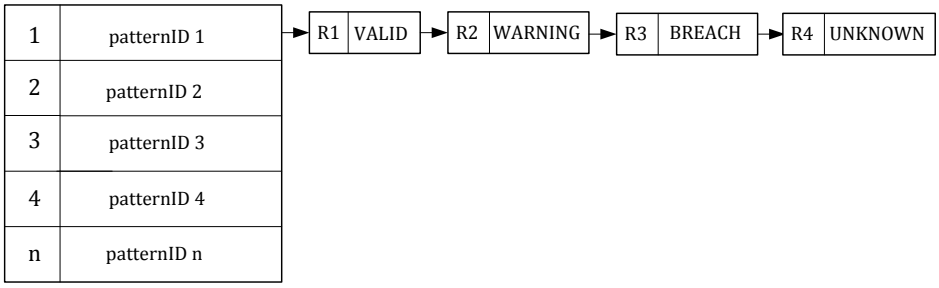

Figure 4.3: The nested Hashmap

Alternatively, another possibility can be used for further developments. Complex Event Processing (CEP) is a generic term for methods, techniques, and tools to process events systematically and automatically as they happen, i.e., continuously and promptly [EB09]. However, for the special use case of file operations, we decided to implement our own Event

[1]Galileo Computing : Java ist auch eine Insel

Processing Matrix instead of using CEP. But for future work, an implementation with CEP could be possible.

4.2.1 Access Revocation

The Access Revocation Component triggers different actions according to the severity and nature of the received event. Depending on the defined result type at the Pattern Engine Component, an action is executed. For *VALID*, a possible action could be writing a log entry; for *BREACH*, shutting down the server; and for *WARNING* and *UNKNOWN*, sending a notification-mail.

The underlying structure of the Access Revocation Component is a factory method pattern. This is a design pattern, and belongs to the creational patterns. It represents an interface for the creation of an object, and lets the subclasses decide which concrete class should be instantiated. In other words, the idea of the factory method pattern is to define a base class type, and then have any number of subclasses which implement the contract defined by the base class[2].

In our implementation the Access Revocation Class implements the factory class. It can return several action types, in which the criteria of an action matches the specified action. The Access Revocation Interface Class represents the abstract class that implements the action method.

Every action that is returned has to implement this class. The classes *doActionBREACH*, *doActionUNKNOWN*, *doActionVALID* and *doActionWARNING* are concrete action classes that implement the Access Revocation Interface class.

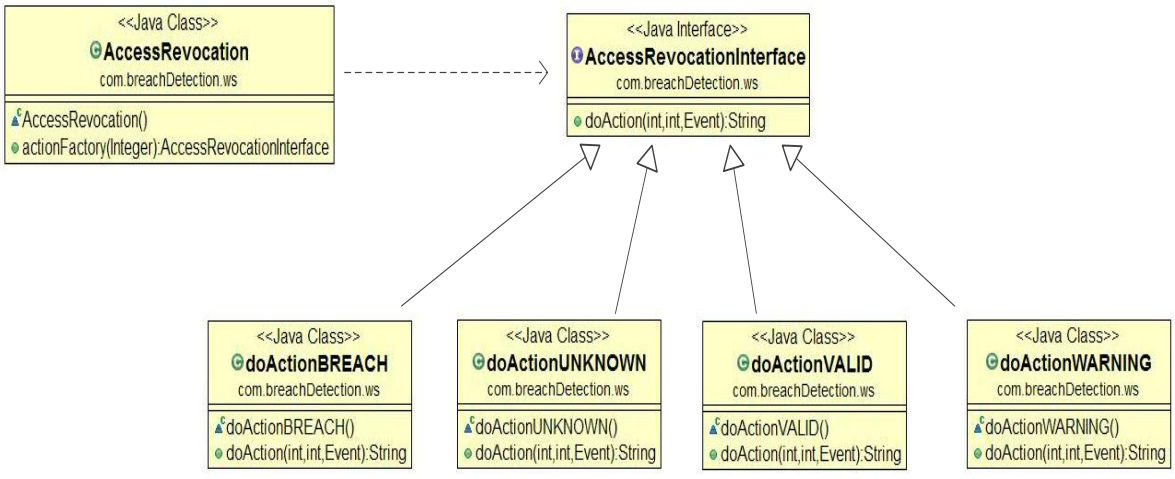

Figure 4.4: Class Diagram of the Factory Pattern Method

[2]alvinalexander.com/java/java-factory-pattern-example

4.3 Functionality

Considering that this prototype is the first version for the implementation of the concept described in chapter 3, some differences exist between the concept and the implementation. In addition to the learning phase and the application phase described in the concept, a visualization phase is implemented with the help of an applet in order to enable a representation of the calculated values.

The comparison of the new event with existing patterns and the computation of the density function is not as described in the concept implicit in the learning phase, but rather explicit in the application phase. That means, if the pattern recognition would be in the learning phase, then indeed patterns are generated by a polynomial function or a markov-chain state machine. The application phase allows instead of an implicit computation, an explicit *measure*. By doing so, the environment of the new event is examined and the density values are measured.

A further difference between the concept and the implementation is that the first version of the implementation supports only one pattern with predefined actions. That means, only one pattern is generated and examined against illegal system behavior and a general action is not used as the concept chapter described. As the Figure 4.4 shows, there are four actions predefined.

The functionality of the BDS is divided into a learning phase, an application, phase and a visualization phase, as shown in Figure 4.5. In the following section, the phases are explained in detail.

Figure 4.5: Functionality of the Breach Detection System

4.3.1 Learning phase

At the beginning of the learning phase, there are no events available. Therefore, in the first part of the learning phase, for the pattern definition, events have to be loaded and defined. An event, as introduced in the previous chapter, is a term to define properties of a process.

In order to get information about processes that perform file operations, and thus access files, the lsof tool is used. As introduced in chapter 1.2, list open files (lsof) consists of a tool that lists information about files opened by a user or a running process. For this purpose, we run the command with the following options: *-Fpuctn*, to receive a list of files that are currently opened by a user or a process; option *-F*, to produce output so that other programs can work more easily; option *p*, which stands for process ID; option *u* for process user ID;

option *c* for process command name; option *t* for file type; option *n* for file name[3]. A lsof output might look like:

p12
cbash
u20
tDIR
n/home/user
tREG
n/sbin/init
tCHR
n/dev/null
tFIFO
npipe
.....

Considering that the use case of this diploma thesis are file operations, only *REG*, which stands for regular files, and *DIR*, for directories are considered. This lsof output is stored in a text file, and read in through the Sensor Component during the first part of the learning phase, the pattern creation. An event with the example lsof output is shown in Table 4.2. This represented event depicts a process with the processID 12 that is executed by a user with the userID 20 in order to run the operation bash.

	processID	userID	operation	path	date
event	12	20	bash	/home/user	creation date

Table 4.2: Generated Event from Lsof Output

To promote a better understanding of the use case, it can be imagined that the whole events of a pattern form a point cloud in a multidimensional space which is shown in Figure 4.6. From the figure, it can be seen that, in this pattern, similar events have clustered at the bottom right.

[3]http://www.computerhope.com/unix/lsof.htm

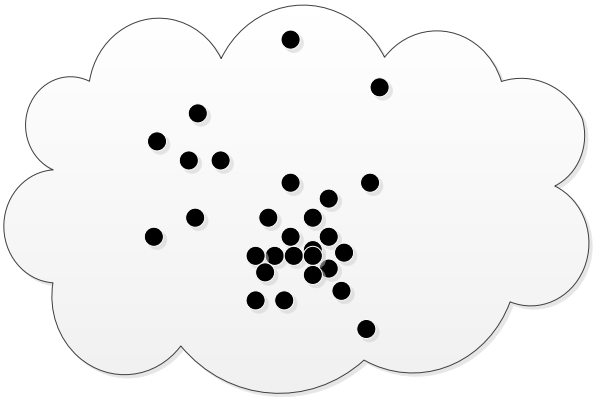

Figure 4.6: A sample Point Cloud

4.3.2 Application phase

The prototype verifies the file access events and determines the legitimacy of a file operation in the first part of the application phase, the recogition phase. To realize the pattern recognition, new events have to be compared with existing ones from which a pattern has been generated. New events arise from the current system behavior just as described in the learning phase.

The purpose of the pattern recognition algorithm is to examine, based on the calculated density, whether a file operation is valid or not. The density describes the relation between all previous events and the new event. In other words, it depicts the frequency of events around a new event in the point cloud. The higher the density, the more likely it is that the new event matches with a pattern, and the smaller the density, the more likely it is that the new event does not match with a pattern. The computation steps of the pattern recognition algorithm to determine the density are shown in the sequence diagram of Figure 4.7.

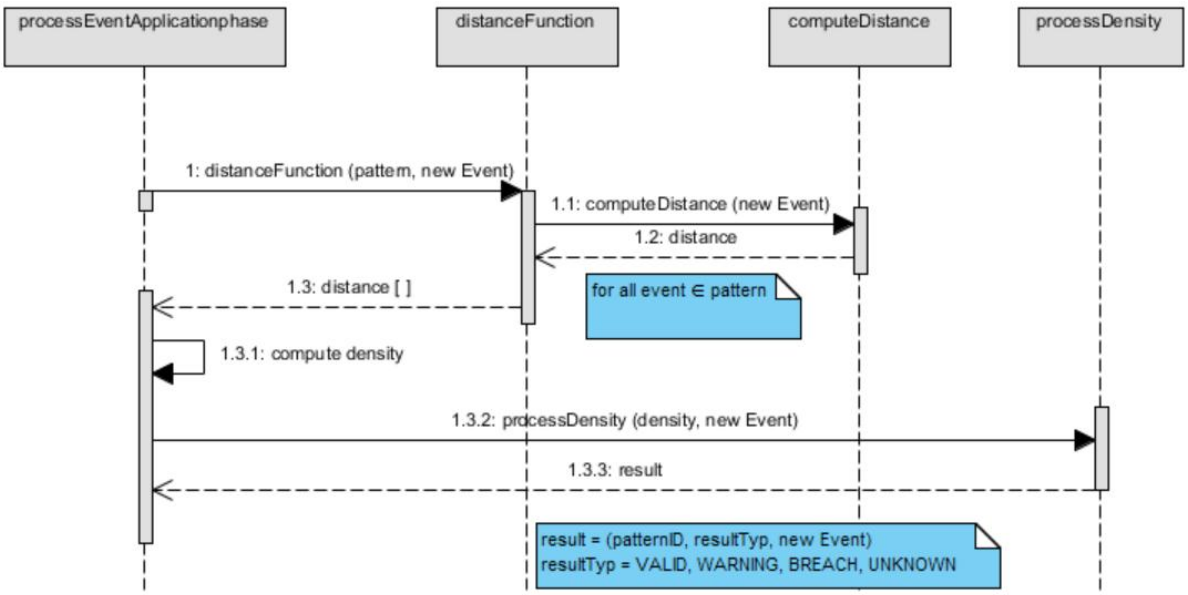

Figure 4.7: Sequence Diagram of the Computation Steps of the Pattern Recognition Algorithm

In order to decide whether a file operation is valid or not, a pattern recognition algorithm is used to calculate the density. To determine the density, first of all, the new event has to be initialized. After this has been done in a way described in the first part of the learning phase, the first computational step of the pattern recognition algorithm to determine the density is to invoke the distance function. The distance function builds the cartesian product of all events.

As described in Algorithm 4.1, the distance function is called with input parameters pattern and newEvent for each new event, which comes in and computes the distance to all events in the point cloud by invoking a sub algorithm, the distance computation function described in Algortihm 4.2. The distance function then returns the calculated distances of all events.

Algorithm 4.1: Distance Function builds the Cartesian Product of all Events

Algorithm distanceFunction (pattern, newEvent)
Input: pattern, newEvent
Output: distances []

```
distances[]
for all event ∈ pattern do
    distance.add(distancecomputationfunction(newEvent, event))
end for
return distances
```

The second computation step of the pattern recognition algorithm is to compare each parameter of the new event with each parameter of all events in the point cloud. This is done with a weighting function which is implemented in the Algorithm 4.2 and weighted the parameters considering the similarity and diversity. The distance computation function has as input values, an event which is included in the pattern, and a new event, which is recorded by monitoring the current system behavior.

To illustrate the computation with an example, the previous defined *event1(12, 20, bash, /home/user, creation date)* and a *new event2(10, 1, echo, /usr/bin/echo, creation date)* are used.

The parameters processID (12 and 10), userID (20 and 1) and operation (bash and echo) are simple to weight because they only can be equal or not equal. If they are equal, they are weighted with 2 and if not, with 0.

The value of the date parameters are the timestamp of the creation date of the both events. If the date value of the event is greater than the date value of the new event, than the date value of the new event is substracted from the date value of the event. Otherwise, if the data value of the new event is greater than the date value of the new event, than the data value of the event is substracted from the date value of the new event. But this case indicates an event in the past which should not be possible, but however if this case occurs, it can be interpreted as a breached event. If the result of this computation is without remainder, than they are weighted with 2. The timestamp is a time designation in seconds, therefore a time difference of at least 1 second is weighted with 0.

Problems occur when comparing the path parameters. For this version of the prototype, we decided to compare both paths without considering the semantic of the content. This means that */home/user1* and */home/user2* would be as similar as */user/bin/echo/x* and */user/bin/echo/y*. However, they are not equally similar, because */home/user1* and */home/user2* are not in the same directory. In Unix-like operating systems, every user has its own home directory. Differently, */user/bin/echo/x* and */user/bin/echo/y* share the same directory. A further sub algorithm compares each sign of the path parameters on equality and returns an integer value. This value is used through the weighting function in order to weight the path parameters. If the result is less than 2, than they are weighted with 2, if the result is less

Algorithm 4.2: Function for Computing the Distance between two Events

Algorithm distance computation function (event, newEvent)
Input: event, newEvent
Output: distance

$dateDifference \leftarrow 0$
$processDifference \leftarrow 0$
$pathDifference \leftarrow 0$
$path \leftarrow 0$
if $event.process == newEvent.process$ **then**
  $processDifference = 2$
else
  $processDifference = 0$
end if
// same for the user and the operation parameter
if $event.date > newEvent.date$ **then**
  $dateDifference = event.date - newEvent.date$
end if

if $dateDifference == 0$ **then**
  $dateDifference = 2$
else
  $dateDifference = 0$
end if
// sub algorithm compares the both path parameters and returns an integer value
if $path < 2$ **then**
  $pathDifference = 2$
else if $path < 6$ **then**
  $pathDifference = 1$
else
  $pathDifference = 0$
end if

$distance = (processDifference + userDifference + operationDifference + pathDifference + dateDifference)$

if $distance == 10$ **then**
  $distance = 1$
else if $distance > 1$ **then**
  $distance = 11 - distance$
else if $distance == 1$ **then**
  $distance = 9$
else
  $distance = 10$
end if
return $distance$

than 6, than they are weighted with 1, otherwise with 0. That means, we normalize the path parameters on a scale of 1 for totally equal and 10 for totally different.

All weighted values are calculated together to a distance value and in turn normalized again on a scale of 1 for nearby and 10 for wide apart. If the distance value is 10, that means, both events are nearby and the distance is weighted with 1. If the distance value is greater than 1, the distance is weighted with the result of 11-distance and if the distance value is equal 1, the distance is weighted with 9, otherwise with 10.

The distance computation function returns the calculated distance of both events to the distance function. These two steps are repeated till all events are compared with the new event in the point cloud. All computed distances are added and stored in a *list of distances*.

The density function Algorithm in 4.3 computes the density of one new event by comparing the distances in the list of distances with a predefined radian. If the distance of the new event is less than or equal the radian, the density, starting at 0, is incremented by 1. It is preferable to choose the radian great enough to ensure that there are enough events in the radian.

Algorithm 4.3: Function for Computing the Density

Algorithm density (list of distances [])
Input: list of distances []
Output: density

$density \leftarrow 0$
$radian \in Integer$

for all $density \in distance$ **do**
  if $density \leq radian$ **then**
    $density \leftarrow density + 1$
  end if
end for
$processResult(processDensity(density, newEvent))$
$densityToPoint(density)$

In order to determine the result type for the density and the new event, the *processDensity* method described in Algorithm 4.3 is invoked. This method implements a conditional structure on a scale from 1 to 10. If the density is less than 4, the result type is *BREACH* and if the density is greater or equal to 4 and less than 8, the result type is *WARNING*, otherwise it is *VALID*. The result type is not considered for now. This method returns a result with the *patternID*, the *result type* and the *newEvent* which in turn is processed to the Event Processing Matrix with the method *processResult*. The Event Processing Matrix is an association-matrix that links patterns to corresponding actions and is implemented as a nested hashmap. In this association-matrix, first after the patternID is searched, afterwards the result type which is stored behind the patternID in order to execute predefined action.

A point cloud with various event points is shown in Figure 4.8. In this example, the red point represents the new event, the black points represent the events from the pattern, the green points represent the density and grey circle represent the radius.

The density for this example is six, which means that the associated result type is *WARNING* according to above classification. An action for this result type might be sending a notification-mail. If the density is smaller than four, the associated result type is *BREACH* and an action might be shutting down the server or some other countermeasures. If the density is greater than eight, the associated result type is *VALID* and an action might be writing a log entry.

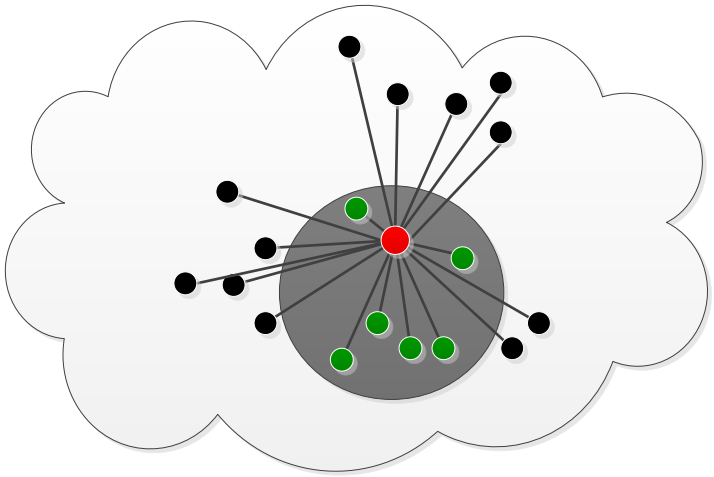

Figure 4.8: Computes the Distance of a new Event (red point) to the existing Events in the Point Cloud (black and green points)

4.3.3 Visualization phase

The purpose of this visualization phase is to provide a user interface in form of an applet in order to visualize the outputs and the behavior of the Breach Detection System and give a motivation for a better understanding. The general approach is to map the outputs from the learning and application phase on two-dimensional coordinates. At a first glance, it is not obvious which outputs or which values should be visualized. There are some possibilities, which are discussed at the next section:

One possibility would be to visualize the patterns. A pattern defined in chapter 3 consists of multiple events and represents a set of file access events. It describes one system function for example a *web server loads home page* or an *email server accesses to the inbox of a 'user'*. The implementation of a pattern is a list of events which have each five parameters. The parameters are *processID*, *userID*, *operation*, *path* and *date*. The idea is to visualize each pattern and mark the new event red, in order to see, to what extent the new event is remote of the patterns. The difficulty of this approach is the question of how five different parameters can

be visualized on a two-dimensional space and how the unit of the coordinate axes has to be chosen in order to map all five parameters properly.

Therefore, another possibility would be to visualize the density. The idea of this approach is to draw a point for each event on the coordinate system concerning the density. The density describes the relation between events and patterns. For each event there is only one value and in order to draw a point on the coordinate system, a further value is required. We decide to take the value of the density as the x-coordinate axis and the value of the patternID as the y-coordinate axis.

In order to draw the points which have only one value, the circle coordinate representation is chosen. With the circle coordinate representation the points can be drawn on the diagonal or on any angle, so that they do not accumulate on a point.

To know the position of the new event on the circumference of a circle, we have to calculate the point (x,y) first. The density is determined in Algorithm 4.3 and with the method *densitytoPoint*, a point can be calculated. For the parameter notation of the point, the radius (r), the angle (a), and the origin (x, y) are required. This yields the following formula:

$$x = cx + r * \cos(a)$$
$$y = cy + r * \sin(a)$$

For our approach, the radius (r) corresponds to the density, the angle (a) corresponds to the patternID and the origin (x, y) to the origin (0, 0). The unit of a has to be converted into radians with *Math.toRadians (patternID)*. For our approach, this yields the following formula:

$$x = density * \cos(patternID)$$
$$y = density * \sin(patternID)$$

The Cartesian coordinate system and the Graph2D class of Java is used to create two-dimensional graphs. Since the origin of the coordinate system in Java is by default in the upper left corner of the applet, the axes of the coordinate system have to be transformed.

Events with a lower density should be drawn further outside, and events with high density should be drawn more in the middle of the coordinate system. To do that, the value of the density has to be reversed to ensure a semantically correct drawing. The maximum density value is deducted from the actual density value in order to normalize the value and ensure that the point is in a range between the maximum and minimum density. The maximum value is the size of the eventlist. This yields the following formula:

$$x = (eventlist.size() - density) * \cos(patternID)$$
$$y = (eventlist.size() - density) * \sin(patternID)$$

When the new event is close to the existing events, then they are within the pattern; the further they move away, the more different they are from the corresponding pattern. Figure 4.9 shows points with low density (red) and points with high density (green). The angle is determined from the patternID, so that it is possible to see how close or how far a point from a pattern is.

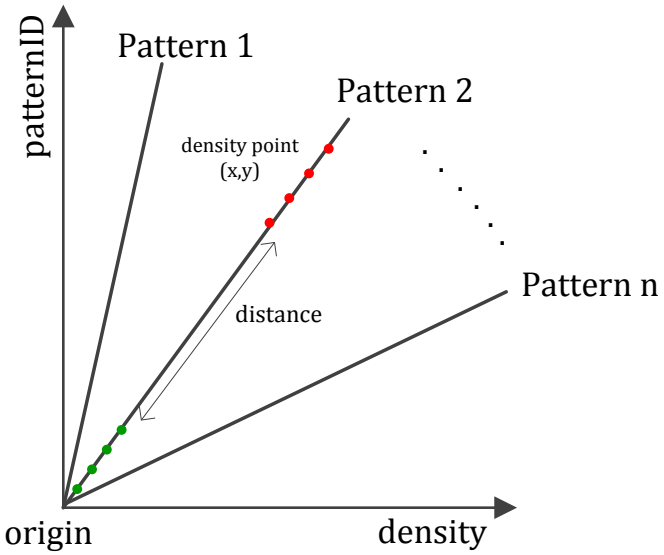

Figure 4.9: Graphical Visualization of the Density

5 Evaluation

In this chapter, an evaluation of the conceptualized novel approach for Security Breach Detection and implementation of a BDS prototype is within the scope of this diploma thesis will be given. Thus, the strengths and weaknesses of the concept, as well as implementation can be figured out because conceptualizing a novel approach and implementing an associated prototype, can lead to several challenges. Therefore, primarily the concept is evaluated in view of open conceptual problems and subsequently, the prototype is evaluated in view of test data, performance, limitations and implementation hurdle during the implementation.

5.1 Evaluation of the concept

Open conceputal problems

At the moment, it is possible to generate only one pattern and examine new events concerning this one pattern. As in previous chapters described, a pattern represents a system function and in order to examine various system functions, a multiple pattern support is required.

Further developments for handling multiple patterns can be added by improving the pattern recognition algorithm. The issue is, that the algorithm should be changed so, that multiple patterns can be recognized. Assuming, that we have a new event, that matches into several patterns, or only into one pattern, or into none of the patterns. The question is when would be an event valid.

A conceptual decision was to represent all events as points in a multidimensional space, also referred as point cloud in the previous chapters in order to constitute a pattern through defining an area in this point cloud. Patterns are objects in a multidimensional space and currently considered as a field which is an area where the events are located. With this approach that only cover an area, it is not possible or very difficult to recognize coherences between events. Assuming, that the sequence of file operations plays a significant role, this approach would not be sufficient anymore. Additional to this issue, it is not obvious to see for a user, when a new event comes in, where the pattern starts and ends.

Therefore, other mathematical constructs have to be used instead of fields, that consider possible probabilities and state transitions and give the opportunity to enable coherences in events. An approach would be the usage of vectors in a multidimensional space.

5.2 Evaluation of the prototype

Test data

In order to evaluate the performance and the proper functionality of the implemented BDS prototype, test data is required. For this testing purpose, a test scenario is evolved which should represent a known system function. A pretty known system function is the process that *a web server loads a web page*.

Therefore, in the learning phase of the prototype, a lsof output is generated manually by executing the command *-Fpuctn*. This lsof output is used to constitute patterns which consists of events and represent various system functions, as well as define corresponding actions that are executed whether an event is valid or not. All events of a pattern form a point cloud. In the application phase, another lsof output with the current system behavior is generated in the same way, as described above.

Performance and Limitations

Once the patterns are generated in the learning phase and a new event is constituted in the application phase, the pattern recognition algorithm will compute the density of the new event concerning the patterns. The density is a value which represents the relation between the new event to all events in the point cloud. Depending on this density, a statement can be made about the validity of the new event. To compute the density, a distance function is used which builds the cartesian product of all events. That means, each parameter of events in the patterns must be compared with each parameter of the new event.

Assuming, our test data consists of a set of 2,000 events or even more which is possible because lsof lists all files which is opened by any process, in the point cloud and each event consists of five parameters. That means, we have to compare 2,000*5 parameters with the five parameters of the new event. The approach to compare each parameter with each other is too time-consuming and not efficient which in turn has negative effects on the performance of the prototype. The comparison increases exponential growth of runtime.

A limitation of the first version of the prototype is, as described in the previous section, that there is currently no support for multiple patterns. At the moment, it is only possible to generate one pattern and examine new events concerning this one pattern. Further developments for handling multiple patterns can be added by improving the pattern recognition algorithm.

A further limitation is the visualization of the density in order to provide a meaningful representation of the calculated values. The current implementation of the prototype uses a predfined coordinate system which is implemented in a Java applet to draw all density values as points. A coordinate system with predefined x and y-axis is not scalable and flexible because assume that the density value is 50 and the patternID is 1. If the corresponding point is (49,99/0,87) and the size of the x and y-axis is only 40 units, the calculated point can

not be drawn. Therefore, the visualization requires a dynamic, scalable coordinate system in order to provide an appropriate visualization. As a first approach, we have taken the maximum x and y value of the calculated points from the density values, as the maximum x and y-axis unit. By doing so, we managed to draw all points in the coordinate system, but a further difficulty occured. All points are having nearly the same values or are having at least only minimal differences. That means, all points are accumulated together and this results in a non-usable graphic.

Implementation hurdle

As introduced and described in the implementation chapter, a pattern recognition algorithm is used to compute the density in order to decide whether an event satisfies the validity of the patterns or not. To do this, a distance function is used which builds the Cartesian product of all events. This distance function described in Algorithm 4.1 computes for each new event the distance to all events in the point cloud by invoking a sub algorithm described in Algorithm 4.2. For this comparison purpose, we implemented a weighting function in order to compare each parameter of the new event with each parameter of all events in the point cloud. An event consists of five parameters; *processID*, *userID*, *operation*, *path* and *date*.

An implementation hurdle was the realization of the conceptualized weighting function. The weighting function compares each parameter of the new event with each parameter of the events in the point cloud and weighted the parameters considering the similarity and diversity. The parameters processID, userID and operation are simple to weight because they only can be equal or not equal. If they are equal, they are weighted with 2 and if not, with 0.

A little more difficult is the comparison with the date parameter which is a timestamp of the creation date of the event. If the date value of the event is greater than the date value of the new event, than the date value of the new event is substracted from the date value of the event. If the result is 0, than they are weighted with 2, otherwise with 0.

The comparison with the path parameter is the most difficult part of the pattern recognition algorithm because a semantic comparison of the prototype is not possible at the moment. A path consists of a sequence of strings like */home/user* or */usr/bin/echo* and in order to simplify the comparison, the path parameters have to be transformed into an integer value. By doing this, a difficulty occurred with the semantic of the paths. To determine the similarity and diversity, a further sub algorithm is used which compares each sign of the string on equality. If all signs are different, except the first root sign /, we can say that the two paths are completely different and weight it with 9, on a scale of 1 for totally equal and 10 for totally different. If all signs are equal, we weight it with 1.

The sub algorithm uses this path value to weight the path parameters. If the result is less than 2, than they are weighted with 2, if the result is less than 6, than they are weighted with 1, otherwise with 0. All weighted values are calculated together and in turn weighted again on a scale of 1 for equality and 10 for diversity.

The difficulty of this approach is, that it is obvious to see that it may not always work and several weighting functions are possible to be implemented. We have arbitrary chosen a weighting function that use the weights the result with 2 for similarity, 1 for result that are neither similarity nor diversity and 0 for diversity. Different weights are at least possible.

6 Conclusion and Outlook

The aim of this diploma thesis was to conceptualize a method for Security Breach Detection using File Access Monitoring and Pattern Recognition. For the implementation of this concept, a novel Breach Detection System (BDS) was designed and implemented in form of a prototype.

The focus of this thesis was to examine file operations at the Unix-like operating system, and derive patterns from the system behavior in order to determine the legitimacy of the file operations. Therefore, as a first step, a brief overview was given on the underlying file concept, as well as on basic knowledge about computer security, pattern recognition and pattern matching technologies.

Currently existing protection mechanisms for detecting, analyzing, and handling security breaches were analyzed, and implementation examples were presented. Typical implementation examples are Intrusion Prevention Systems (IPS) and Intrusion Detection Systems (IDS), which aim to monitor the network and the computer system against malicious vulnerabilities, thus ensuring the security and confidential information of organizations, private persons, and governments.

The aforementioned concept is based on the idea of determining whether it is possible to derive, through a pattern recognition algorithm, patterns about the system behavior of file operations. A pattern consists of various events, and describes the allocation of system behavior on patterns that are generated at a file access event. An event denotes the properties of a process that is executed by a user or a running program. This means that each pattern represents a certain system function of an underlying operating system. Based on these patterns, it should be identified on which files processes are accessed, in order to determine the legitimacy of the file operations.

The proposed BDS prototype was implemented in Java, and web service was used for the client-server communication, which provides an application-to-application communication. In a learning phase, the system behavior was monitored continuously in a controlled state, and patterns were constructed and defined from the information on system behavior. In an application phase, the current system behavior was recorded while the system was running in a normal environment (not secured) and compared against existing patterns, in order to initiate countermeasures if necessary, when detecting illegal system behavior.

As examined in the evaluation chapter, the first version of the prototype has some implementation limitations as well as performance problems which depend on that the fact that the development time of a thesis is restricted, and not all of the intended functions could be completely implemented. Currently, the prototype supports only one pattern that is

examined. In order to efficiently detect and analyze security breaches, the prototype has to be improved in a way that multiple patterns are supported. It must be noted that the pattern recognition algorithm and the underlying data structures have to be changed.

The computation of the distance function is currently explicit as a cartesian product of all events. The drawback of this approach is that computation for a large amount of events is too time-consuming and not efficient enough at the moment. Therefore, an implicit approach for the computation of the density function would increase the efficiency. The density function could be created as a polynomial function during the learning phase.

A sub algorithm of the distance function, the weighting function compares each parameter of a new event with each parameter of the events in the point cloud. As already mentioned in the evaluation chapter, the approach of the weighting function could perhaps not be applicable for every approach. Therefore, a more general weighting function has to be improved and implemented. A further improvement for the weighting function would be the use of *Dewey Decimal System (DDC)* which is an international widespread universal classification system[1] and would enable content-based semantically correct comparison of the path parameters of an event.

To generate a pattern, information about the system behavior is required. Lsof (List open files) is a tool which provides this knowledge by listing all information about files that are currently opened by a process or a user. The difficulty with lsof is that not all open files can be found, because during examination of the process table, the state of some processes with files open might have changed, without lsof being able to recognize it[2]. Therefore, in this first version of the prototype, lsof is used only as a provisional solution. For further implementations, the prototype should communicate directly with the kernel.

The evaluation chapter has shown, that the current approach with the usage of fields, has some limitations. A limitation is the issue that the definition of the current approach which only covers one area, does not enable relations between events and representations of events as a sequence of file operations. Therefore, an idea for a further development regarding the limitations of the current approach would be the use of a vector in a multidimensional space. A vector would enable to define a pattern not only as an area in the point cloud, but as a sequence of file operations in various directions and a new event has to match within this vectors. One of the five parameters of an event is the date parameter and represents one dimension in the multidimensional space. A vector would enable to define the temporal behavior of events as a direction between events and to depict where the event starts and ends. A deviation of the direction of the vector or if an event goes beyond the vector would lead to a breach.

Figure 6.1 shows at the left side a point cloud with a sequence of events in which all events would be in the point cloud and at the right side two vectors with sequences of events in which each vector covers a sequence of events.

[1] http://www.oclc.org/dewey.en.html
[2] ftp://lsof.itap.purdue.edu/pub/tools/unix/lsof/FAQ

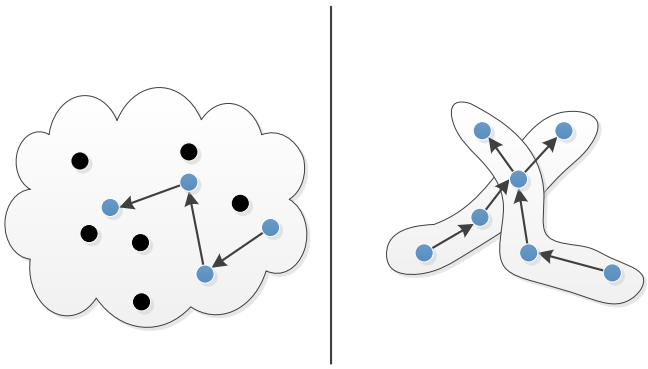

Figure 6.1: Point Cloud with a sequence of operations (left) and two vectors with sequences of operations (right)

In order to provide an effective complex pattern recognition engine, other mathematical constructs have to be used instead of a field and even vectors would not be enough. For this purpose, an approach is required that consider possible probabilities and state transitions and give the opportunity to enable coherences in events.

The positive side effect of this implementation is that properties of a system can be recognized which are not obvious to figure out, for example how precisely enough should the SELinux rules be defined or how should be the access permissions of a directory or how many users should exist in order to determine the access permissions accurate enough. In a pattern database, it is possible to see how the system behaves, which files are opened, and which files are written.

This could be a good basis to implement prevention mechanisms and set file permissions that could possibly not be easy to determine. Rather than its use as a Breach Detection System, an expanded possibility could be its use as a support system for configuring prevention mechanisms and file permissions. Additionally, it could be helpful for security administrators to recognize the system behavior in order to subsequently determine preventive mechanisms.

Bibliography

[A.W06] S. A.Walberg. Finding open files with lsof. *IBM developerWorks*, 2006. URL http://www.ibm.com/developerworks/aix/library/au-lsof.html. (Cited on pages 13 and 14)

[BH02] J. M. N. M. Benjamin Hoherz, Silvio Krueger. *Intrusion Detection Systeme in Firewalls*. Master's thesis, Fachbereich Informatik Universität Hamburg, 2002. URL http://agn-www.informatik.uni-hamburg.de/papers/doc/bacarb_hoherz_krueger_menne_michaelsen.pdf. (Cited on pages 19, 20 and 31)

[Bie05] T. Biege. Intrusion Detection Systeme Ein Überblick. 2005. URL http://users.suse.com/~thomas/papers/IDS/Intrusion/Detection/Systeme-EinÜberblick.pdf. (Cited on pages 27 and 28)

[CSI11] Computer Crime and Security Survey 2010/2011. Technical report, Computer Security Institute, 2010/2011. URL http://gatton.uky.edu/FACULTY/PAYNE/ACC324/CSISurvey2010.pdf. (Cited on page 15)

[CSO11] D. CSO, CERT. Cybersecurity Watch Survey: Organizations need more skilled cyber professionals to stay secure. Technical report, CSO, CERT, Deloitte, 2011. URL http://www.sei.cmu.edu/newsitems/cybersecurity_watch_survey_2011.cfm. (Cited on page 15)

[Deb02] H. Debar. An Introduction to Intrusion-Detection Systems. 2002. URL http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.101.7433&rep=rep1&type=pdf. (Cited on pages 24 and 25)

[Den87] D. Denning. An Intrusion-Detection Model. *Software Engineering, IEEE Transactions on*, SE-13(2):222–232, 1987. doi:10.1109/TSE.1987.232894. URL http://ieeexplore.ieee.org/stamp/stamp.jsp?tp=&arnumber=1702202&isnumber=35884&tag=1. (Cited on pages 23 and 24)

[Dra04] O. Dragon. The What and Why of Intrusion Detection Systems. 2004. URL http://www4.ncsu.edu/~kksivara/sfwr4c03/projects/OliDragon-Project.pdf. (Cited on page 23)

[EB09] M. Eckert, F. cois Bry. Aktuelles Schlagwort "Complex Event Processing (CEP)", 2009. URL http://epub.ub.uni-muenchen.de/14902/. (Cited on page 46)

[Eck12] C. Eckert. *IT-Sicherheit: Konzepte - Verfahren - Protokolle*. Oldenbourg, München, 7., überarb. und erw. aufl. edition, 2012. URL http://swbplus.bsz-bw.de/bsz350724911inh.htm. (Cited on pages 17, 28 and 29)

[Garo8] M. Garrels. *Introduction to Linux- A Hands on Guide*, 1.27 edition, 2008. URL http://www.tldp.org/LDP/intro-linux/intro-linux.pdf. (Cited on page 11)

[GW] K. S. Gabriel Welsche. Grundlagen Sicherheit. Technical Report Revision: 1.1.2.11. URL http://www.linux-services.org/selflinux/html/grundlagen_sicherheit.html. (Cited on page 34)

[Hilo1] M. Hildebrandt. Intrusion Detection am Beispiel von Snort. 2001. URL http://www.pro-linux.de/artikel/2/1121/intrusion-detection-am-beispiel-von-snort-teil-1.html. (Cited on page 23)

[Kapo7] M. Kappes. *Netzwerk- und Datensicherheit: Eine praktische Einführung*. Teubner, Wiesbaden, 2007. URL http://nbn-resolving.de/urn/resolver.pl?urn=10.1007/978-3-8351-9202-7. (Cited on pages 16, 17, 20 and 21)

[Kri98] R. Krienke. *UNIX für Einsteiger: eine praxisorientierte Einführung*. Hanser, München, 2., verb. aufl. edition, 1998. (Cited on page 11)

[LB12] M. L. R. S. M. S. B. S. B. F. Lars Baumgaertner, Pablo Graubner. Mastering Security Anomalies in Virtualized Computing Environments via Complex Event Processing. *The Fourth International Conference on Information, Process, and Knowledge Management*, 2012. URL http://www.accept-projekt.de/publications.html. (Cited on page 30)

[Ley11] P. D. F. Leymann. The Web as Platform - lecture notes of University of Stuttgart (IAAS), 2011. (Cited on pages 41 and 42)

[Meio7] M. Meier. *Intrusion Detection effektiv!: Modellierung und Analyse von Angriffsmustern*. Springer Berlin Heidelberg, Berlin, Heidelberg, 2007. URL http://nbn-resolving.de/urn/resolver.pl?urn=10.1007/978-3-540-48258-1. (Cited on pages 23 and 25)

[MPB+12] C. Modi, D. Patel, B. Borisanya, A. Patel, M. Rajarajan. A novel framework for intrusion detection in cloud. In *Proceedings of the Fifth International Conference on Security of Information and Networks*, SIN '12, pp. 67–74. ACM, New York, NY, USA, 2012. doi:10.1145/2388576.2388585. URL http://doi.acm.org/10.1145/2388576.2388585. (Cited on page 21)

[Peto8] H. Peterreins. Mustererkennung. In *Grundsätze soliden Investierens*, pp. 20–22. Gabler, 2008. doi:10.1007/978-3-8349-8144-8_4. URL http://dx.doi.org/10.1007/978-3-8349-8144-8_4. (Cited on page 17)

[SBo8] W. Stallings, L. Brown. *Computer security: principles and practice*. Pearson Prentice Hall, Upper Saddle River, NJ, 2008. URL http://bvbr.bib-bvb.de:8991/F?func=service&doc_library=BVB01&doc_number=015782366&line_number=0002&func_code=DB_RECORDS&service_type=MEDIA. (Cited on pages 16, 20, 23, 27, 28 and 29)

[SB10] M. A. S. M. Saira Beg, Umair Naru. Feasibility of Intrusion Detection System with High Performance Computing: A Survey. International Journal of Advances in Computer Science, 2010. (Cited on page 21)

[SFSF11] M. Schmidt, S. Fahl, R. Schwarzkopf, B. Freisleben. TrustBox: A Security Architecture for Preventing Data Breaches. In *Parallel, Distributed and Network-Based Processing (PDP), 2011 19th Euromicro International Conference on*, pp. 635–639. 2011. doi:10.1109/PDP.2011.44. (Cited on page 22)

[SP04] R. Stevens, H. Pohl. Honeypots und Honeynets. *Informatik-Spektrum*, 27(3):260–264, 2004. doi:10.1007/s00287-004-0404-y. URL http://dx.doi.org/10.1007/s00287-004-0404-y. (Cited on page 30)

[ST13] P. E. Schukat-Talamazzini. Mustererkennung Vorlesung im Sommersemester 2013, 2013. URL http://www.minet.uni-jena.de/fakultaet/schukat/ME/Scriptum/lect01-intro.pdf. (Cited on page 18)

[Thr12] ThreatMatrix. Cybercrime Battle Basics Online Account, Transaction and Device Protection. Technical report, ThreatMatrix, 2012. URL http://www.threatmetrix.com/docs/Whitepaper-Cybercrime-Defender.pdf. (Cited on page 21)

[Upa11] S. Upadhyaya. Mandatory Access Control. In H. van Tilborg, S. Jajodia, editors, *Encyclopedia of Cryptography and Security*, pp. 756–758. Springer US, 2011. doi:10.1007/978-1-4419-5906-5_784. URL http://dx.doi.org/10.1007/978-1-4419-5906-5_784. (Cited on page 22)

[Ver13] Verizon. Data Breach Investigations Report 2013. Technical report, Verizon, 2013. URL http://www.verizonenterprise.com/resources/reports/rp_data-breach-investigations-report-2013_en_xg.pdf. (Cited on page 9)

Alle URLs wurden zuletzt am 12.11.2013 geprüft.

Declaration

I hereby declare that the work presented in this thesis is entirely my own. I did not use any other sources and references that the listed ones. I have marked all direct or indirect statements from other sources contained therein as quotations. Neither this work nor significant parts of it were part of another examination procedure. I have not published this work in whole or in part before. The electronic copy is consistent with all submitted copies.

(Zeynep Öztürk)