UiO : Department of Informatics
University of Oslo

Protecting Sensitive Data from System Administrators

Magnus Tranum - mtr@student.matnat.uio.no

Master Thesis spring 2014

Protecting Sensitive Data from System Administrators

Magnus Tranum

Master Thesis – Spring 2014

Abstract

This thesis is an attempt to limit the access to personally identifiable data by system administrators. The goal is to prevent unauthorized, and illegal access so as to conform to legal requirements, preventing both rogue admin vulnerabilities and unintended access which could place the offending administrator in an uncomfortable position.

This is accomplished by finding the optimal point in which to make an implementation, researching relevant measures to implement a solution at the Cancer Registry of Norway (KRG) and creating a module using the existing development norms at KRG.

The module is tested by placing it in an artificial environment which simulates attempted forms of access by an imagined rogue admin, as well as several actions which could lead to unintentional access.

The module was completed with one missing aspect due to unforeseen hurdles. Emulating the missing aspect demonstrates that the module as it is designed prevents unwanted and unintended access to personal data within the specifications presented, resulting in a layer of security which was not previously extant.

Table of Contents

List of Figures

List of Abbreviations

AD/ADC	- Active Directory (Controller)
AOP	- Aspect Oriented Programming
ASE	- Adaptive Server Enterprise
BPMN	- Business Process Model and Notation
CIA	- Confidentiality, Integrity and Availability
CTSS	- Compatible Time-Sharing System
DB	- Database
DBQ	- Database Query
DMZ	- Demilitarized Zone
EPJ	- Elektronisk Pasientjournal (Electronic Patient Journal)
HCSIRT	- Health Computer Security Incident Response Team
HF	- Helseforetak (Health undertaking)
ISO	- International Organization for Standardization
IP	- Internet Protocol
IT/IKT	- Information Technology / (Norwegian equivalent)
JVM	- Java Virtual Machine
KDB	- Kvalitetsdatabasen (Quality Database)
KISS	- Keep it simple, stupid
KNEIP	- Kreftregsiterets Nye IKT Platform (The Cancer Registry's New Electronic IT Platform)
KREMT	- Kreftregisterete E-MeldingsTjeneste (The Cancer Registry's Electronic Message Service)
KRG	- Kreftregisteret (The Cancer Registry of Norway)
LAN	- Local Area Network
MD5	- Message Digest 5
MIT	- Massachusetts Institute of Technology
NHN	- Nasjonalt Helsenett (National Health Net)
OSI	- Open System Interconnection model
PNS	- Personal Number (encryption) Service
RSA	- Rivest, Shamir and Adleman cryptosystem
SQL	- Structured Query Language
SSL	- Secure Sockets Layer
TOGAF	- The Open Group Architecture Framework
UFW	- Uncomplicated Firewall
UML	- Unified Modeling Language
US	- United States
VDI	- Virtual Desktop Infrastructure
VLAN	- Virtual Local Area Network
VM	- Virtual Machine
VPN	- Virtual Private Network

Preface

I would like to thank my primary supervisor Jan Nygård for the opportunity to complete my thesis at the Cancer Registry, for help and guidance in understanding the systems in place and for guiding me when I needed to make the big decisions.

I would like to thank my secondary supervisor Arne Maus for his help proofreading, correcting my disastrous sense of structure, several interesting conversations on the nature of security and other things and giving me motivation when I needed it most.

I would like to thank Therese Hüber, Tormod Eriksen, Geir Danielsen, Basit Ahmed Khan, Steinar Auensen, Frank Rønning, Leif Magne Eriksen and Sølve Monteiro for many meetings and discussions surrounding the systems, network and security at the Cancer Registry, without whose help this thesis would never have been possible.

I would also like to thank my fellow master student Kim Kheim Ho Xuan for giving me inspiration and lending me an understanding ear during several rants.

Finally I would like to thank my family for their support, and the love of my life for her endless patience and understanding.

Part I

Introduction

Chapter 1 – Introduction

1.1 – Motivation

Protecting critical or sensitive data from System Administrators, who in essence have access to everything, is a daunting task that has no fixed solution. A 2010 Verizon Data Breach Investigations Report found that in 2009 insiders participated in 48% of all known data breaches, and about 12% of those were rogue system/network administrators(Baker et al., 2010). While this number seems small, evidence supports that they do facilitate the bigger breaches as the elevated privileges offer greater opportunity for abuse. Note also that there are certainly a high number of unknown breaches, but no reliable numbers exist to estimate these for obvious reasons. With recent high-profile incursions by System Administrators as seen in the Edward Snowden incident(Greenwald, MacAskill, & Poitras, 2013) and many of the documents hosted on WikiLeaks, the need for more focus on precisely this threat has been put into stark relief.

There are of course existing solutions that specifically target abuse from elevated users, but the challenge is to have a solution which both conforms to their custom in-house systems, legal requirements and strikes the right balance between security and ease of use for administrators in their daily tasks. Regular logging and authorization is insufficient when an administrator can manipulate the environment to both gain access and mask their tracks. With no existing suite that includes this specific functionality there is a security gap that needs filling.

The motivation then is the desire is to prevent unauthorized access to the specific databases containing personal information from potential rogue admins, as well as System Administrators who are loyal. This is important not only because loyalty can never be completely assured, but also because administrators should not have to see data they are not supposed to in order to protect them from legal repercussions, blackmail and possible data on relatives and acquaintances.

1.2 – Goals

The goal of the thesis is to prevent unauthorized access to personally identifiable data by System Administrators, or if all else fails discover the unauthorized access. I want to uncover which methods of security would be most effective and how it can be implemented most efficiently within the systems at KRG.

1.3 – Outline

The thesis is divided into chapters, with this chapter presenting the outline of the remaining chapters. The early chapters present the background needed to complete the goal and the later chapters present the work done followed by results and a conclusion.

Chapter 2: Computer Security Basic

Gives an overview of the history of Computer Security followed by an outline of basic concepts. Covering abstract security concepts, internal and perimeter security, the human element and pragmatic concerns such as security in code. This chapter also presents where this thesis fits into the larger context of Computer Security as a whole.

Chapter 3: Security at the Cancer Registry, the Context

Presents an overview of the security structure at KRG, briefly covering physical security and delving into IT security components. These include the zone division of the internal network, credential management and authorization, how sensitive data is managed and the policies currently in place.

Chapter 4: The KRG IT Environment

This chapter presents the environment at several levels, beginning first with how cancer reports and data extracted from them move through and interact with the KRG systems. Then the system architecture is presented and the systems in which the relevant sensitive data is manipulated are explored further by delving into their technology architecture.

Chapter 5: Existing Solutions Targeting Rogue Administrators

Presents several solutions currently known to be in use which specifically target elevated user abuse. None of these alone can prevent it entirely, but they are reasonable attempts at limiting the occurrences which are often implemented in some form or combination in other security systems.

Chapter 6: Deciding on a Solution

In this chapter an example of a perfectly secure system is presented, along with a description of why it is an unviable solution. Then a description of the powers of System Administrators for context is followed by a description of the systems relevant to the thesis and a narrowing of the scope so that a solution for implementing can be approached. Several assumptions and prerequisites are presented and a solution is decided on.

Chapter 7: Implementing the Solution

In this chapter a description of assumed services followed by which services will need to be implemented to achieve the goal are presented, and a detailed account of each facet of the solution is discussed. This includes the progress of the actual implementation along with code examples to demonstrate how it was done.

Chapter 8: Testing and Results

The implementation is tested in an artificial environment to demonstrate the viability of the solution presented. Each service is tested independently, though some services trigger others so the entire solution is tested with a collection of inputs. The implementation works as expected, catching and analyzing the desired service calls, and the results presented show this.

Chapter 9: Conclusion

The conclusion of the thesis, presenting what went well, what did not go as expected along with presenting the contribution to existing work.

Part II

Background and Context

Chapter 2 – Computer Security Basics

When delving into modern computer security, which is the subset of information security that applies to computers and networks, some concepts need to be understood before one can begin to consider how to attack and solve the problem at hand. Looking then at network and data security you have to make a few basic assumptions(Paquet, 2012; Various)**.** Firstly, modern systems are very large and interconnected. They are often open to exploitation as they run a mix of software which are not always fully compatible, and not always up to date. It is the simplest thing in the world to make an entirely secure system, but it would have to be physically separated from everything else with no way to interact with it, which is not a viable option in almost all instances. As a result, the possibility of remote access through the internet by an individual source or even distributed botnets is a continuous reality which is at the very core of understanding the risks of providing proper security. Even with an internal network which is 'separated' from the internet as a whole with firewalls and strict policies, this is something which must be taken into account as no amount of precautions short of physical separation will guarantee immunity to external intrusion. Even some internal networks are also becoming so large that the principles that are used for the internet must be taken into consideration when dealing with internal security.

Secondly, systems and applications connected to these networks are becoming increasingly complex making it more difficult to properly analyze and secure them. Even many of the simplest computer programs are undecidable with respect to termination, and so approximations must be used when trying to decide whether they are secure enough. The problem naturally becomes exponentially harder the more complex the systems that they are applied to become. As a result the possibility of vulnerabilities are ever increasing and you have to take this into account when implementing any software or system which implements some security policy.

2.1 – A Brief History

Computer security has been a concern ever since the dawn of electronic computers, and crackers attempt to find vulnerabilities as soon as new technologies and systems become available. Arguably the first breach of security as it relates to computer technology occurred in 1903, when the magician and inventor Nevil Maskelyne disrupted a public demonstration of Guglielmo Marconi’s purportedly secure wireless telegraph technology by sending insulting morse code messages through the auditorium’s projector(Marks, 2011).

A few decades later, in 1939, Alan Turing, Gordon Welchman and Harold Keen developed ‘the Bombe’, which was based on Maria Rejiwski’s ‘Bomba’, in order to crack German Enigma codes(Kozaczuk, 1984). The Enigma machine used a reliably small key space, making it vulnerable to brute force attacks, and with this development it became apparent that encryption keys needed to be sufficiently long to stay practically unbreakable by contemporary computational power.

Later still, in the 1960’s, William D. Mathews from MIT found a vulnerability in a Multics CTSS, the first documented software vulnerability, which disclosed the contents of the password file by running multiple instances of the system text editor(Karger & Schell, 1974). The location of the password file was supposed to be secure, but this exposed the problem of having passwords in clear text if a cracker actually manages to gain access to it. This resulted in enciphered versions of user passwords to be made for the first time, a development which became central to all future password management.

Much of the early work of studying computer security was financed and done by the US Department of Defense, from the 50’s on, though as computers became more readily accessible in the 80’s the focus had to expand beyond ‘simply’ securing the technologies and begin to take into account human error. It became easier to guess passwords than attempt to decrypt them(Morris & Thompson, 1979), and as networks grew and finally the Internet came into being in the 90’s, the human element became an unavoidable cornerstone in understanding and ensuring computer security.

To sum it up, modern computer security has to take into account a combination of technological and social understanding to create systems that are not only compliant to standards and regulations, but also prevents accidental breaches as the result of human fault.

2.2 - CIA

As a result of these realities, the field of computer security is by necessity vast and multifaceted, but at its core are three basic requirements: *Confidentiality*, *integrity* and *availability*, also known as the CIA triangle(Andress, 2011; Miami, 2006). All three factors are important to keep in mind when coming to a decision on my implementation, though they are only abstracts.

2.2.1 Confidentiality

Providing confidentiality of data guarantees that only users who are authorized can access sensitive information. This is probably the most obvious aspect of network security, and it is also the one which is most often attacked. Providing data confidentiality is necessary as disclosure of the sensitive data may result in some form of loss or damage, such as identity theft, loss of business, regulatory fines or legal complications. In the case of KRG the main issue is identity protection with the possibility of regulatory fines and legal complications due to breaches of security of personal data.

Ensuring confidentiality hinges on two aspects. Logical and physical securing of data.

- Physical security hinges on ensuring that server rooms are secured, that unauthorized personnel are kept out of sensitive areas and that workstations are locked down when users are not present.
- Logical security includes authorization, authentication and encryption. Authorization is the process of determining who should have access, authentication is ensuring that only authorized people are able to gain access to the data through the use of passwords and user identification methods and encryption is ensuring the data itself is adequately protected through the use of an appropriate algorithm. The algorithm strength is often decided by the key length, and most modern algorithms are so strong that a conventional computer or even most server farms have no chance of decoding it before new algorithms or keys are in place.

Looking specifically at encryption there are two main focus, storage encryption and transmission encryption.

- Storage encryption encompasses any form of encryption done on static data. It can be encryption of anything from entire physical hard drives to specific files and folders.
- Transmission encryption refers to the action of encrypting files for transmission over a network from one entity to another. Data is encrypted, sent and then decrypted at the receiving end.

The practical implementation of either is the same, and the decision of whether to encrypt files as soon as they are created on a device, or only encrypting it when needing to be transmitted depends on the type of data, its usage or even policy decisions. As an example, when transferring data the source and destination (server and client, sender and receiver) must be able to encrypt and then decrypt the message. This is accomplished either through the use of symmetrical or asymmetrical encryption methods. Symmetrical encryption means that the source and destination have a shared key. This is the most common form of encryption, as the key can easily be shared among authorized personnel and it requires little computational power. Asymmetrical encryption means that the source has one key and the destination uses a different one to decipher the cipher texts. Also referred to as a public/private key pair, it is more resource intensive and the use of a public key must be authenticated. Switch out the terminology of sender and receiver with write and read and it is equally applied to storage encryption.

Encryption is already in place at KRG, and it might be necessary to understand the various forms of encryption and how they are applied before implementing a possible solution.

2.2.2 Integrity

Providing data integrity means ensuring that it is not modified by any unauthorized party, that any modifications are traceable and any data to be read or received is an accurate representation of the source data. In relation to information systems there is no assurance that the content is correct or true, only that whatever is input is preserved and accurately represented.

- The complexity of transmission integrity depends on the number of steps it has to go through before reaching its source. The most banal requirements are that any lines of transmission are intact, the hardware functions properly and the receiving environment understands what you are entering. When you add distance and layers of transmission protocols such as networks or even the internet the complexity grows exponentially. Data can be intercepted through man-in-the-middle attacks and this needs to be taken into account to ensure the data arrives unmodified by malicious intent. Data integrity during transmission can be accomplished through the use of the correct protocols, passive checks such as hashing or through active monitoring systems which attempt to detect anomalies in data transfer rates or content. Hashing is widely used as it is generated mathematically from the source data, and can be used to verify the data upon arrival. Also, it is important to ensure the integrity of the mode of transmission itself through the use of secure connections such as SSL, certificates or Virtual Private Network (VPN) tunnels.

- Storage integrity is also essential, but is not always quite so easy to attain. At the software level integrity is mostly ensured through the use of constraints. Since research shows that most current widespread file systems don’t provide sufficient protection against data integrity problems(Prabhakaran et al., 2005), databases are the preferred mode of storage when it comes to data that requires strict integrity.

Beside these two key areas there is source integrity. Impostors may masquerade as a legitimate source making you enter your username and password or give your shared key thus compromising any other level of security in place, so ensuring source integrity is important, even in internal networks. In addition to secure connections, you can employ tertiary levels of authentication such as authenticators, used in many European banks, or visual verification such as employed by Bank of America.

2.2.3 Availability

Providing uninterrupted access to data for users is of utmost importance, though it is quite beside the focus of this thesis. Suffice to mention that availability is the act of ensuring crypt key availability for authorized personnel, reliable uptime through robust hardware, redundant connections and defense against distributed attacks.

2.3 - Internal Security

Now that the abstract basics of security have been covered, it’s time to move on to look at the whats and hows of implementing these concepts on an internal network or system. The whats are described by security policies, while the hows are covered by procedures, standards, baselines and guidelines. Once these are in place a security model can be produced to conform to the policies laid out. A security model maps the policy requirements into a set of rules that can be followed by a computer or network system, and defines the level of security to be implemented.

2.3.1 Security policies

The set of rules, practices and procedures dictating how sensitive information is managed, protected and distributed in a network are called policies. Policies are the ‘what’ of system security, decided by management and given to the relevant departments to implement to the best of their ability. Trust is usually a big part of any security policy, as people are necessarily a central part of any security system. Finding the balance between regulation and control with productivity goals is an ongoing process in any organization. An important point to note is that policies must be concise and to the point, ensuring they are as unambiguous as possible.

2.3.2 Standards

Standards are recognized best practices, principles and frameworks which are agreed upon on an industry-wide scale. They are like policies strategic in nature, as they define system parameters and processes without going into specifics. There are many standards, and they often overlap so deciding on a collection of appropriate standards for a given system is important.

2.3.3 Procedures

Procedures are the low-level documents providing systematic instructions on how the security policies are to be implemented into a system. Usually drafted by individual departments on the basis of policies set by higher management, procedures need to provide maximum information to the users and are therefore very detailed in nature. They also go in depth on how to apply the standards and guidelines of a security program and are usually the de-facto source of referral when describing security requirements.

2.3.4 Baselines

A baseline is the minimum level of security required by a system. In order to conform to the absolute minimum security requirements across any system in a business, a user would look at the baselines. An example of a baseline would be that each Ubuntu server must be updated to at least version 12.0.4 and include UFW firewall.

2.3.5 Guidelines

Guidelines are similar to procedures in that they are tactical in nature and are a list of actions for users, though they differ in that procedures are usually mandatory while guidelines are just that. Guidelines are often used as references to help users perform certain actions in ways that are recommended for optimal security and efficiency, while not being strictly mandatory.

2.4 - Perimeter Security

Once the security policies have been agreed on, the procedures distributed and the models implemented, a boundary marking the end of the secure internal network and the larger external and untrusted network, such as the internet, must be defined(Northcutt, Zeltser, Winters, Kent, & Ritchey, 2005). This is called the perimeter, and perimeter security is vital to ensure that your, hopefully adequate, security measures are not contaminated from the outside. This is entirely outside the scope of the thesis, though I will give a short overview of the nature of perimeter security for the sake of a complete picture.

2.4.1 Defining the Perimeter

In recent years there has been a great deal of change in the opinions of perimeter security as the very nature of a network is changing and becoming increasingly uncertain. There is still a case to be made for a strong defined perimeter, but more and more it is becoming a landscape of layers. Instead of having a single perimeter defining the clear end of the network you could have several layers of networks, one contained inside the next. Despite the ambiguity, it is still necessary to define where the borders go, make a clear distinction between the untrusted outer networks and the trusted inner ones. There is in other words still a need for an outermost layer of protection. This outer perimeter is often accompanied by a Demilitarized Zone or DMZ(Thomas, 2004), which is a physical or logical subnet that contains and exposes an organization's external-facing services without giving external connections direct access to the internal network. A DMZ is designed to provide an extra layer of security so that you can ensure your network, systems and data remain secure and operational even if the DMZ services are attacked and compromised.

2.4.2 Security in Layers

Inside the outer perimeter the number of layers depend entirely on the organization and can range from one to several dozen. This division of everyday resources and restricted resources is what would define any internal network layering, and any additional layers would be the result of grades of sensitivity and required permissions for access. National militaries often operate with many layers. This all seems quite simple, but with the recent blossoming of VPN's(Ferguson, 1998) and virtual desktops defining the borders has become a complex process. A computer on the untrusted network can use a VPN connection to gain access all the way down to secure layers of the network, and the implementation of this must be carefully planned.

2.5 - Other Aspects

2.5.1 The Social aspect

When considering any form of information security, it is sometimes easy to forget the vital social component(Frangopoulos, Eloff, & Venter, 2008). At the end of the day, humans are interacting with the hopefully well designed system and this has its own vulnerabilities. Humans are habitual, they have routines and interests that might override regular caution and this can be exploited by a patient intruder. Analyzing the trends of employees can give the intruder a target to aim at, and more often than companies would like this works very well. Spoofing an expected email from a friend or coworker which pushes on a vulnerability gleaned from the footprinting can open the door to installing malicious code and gaining a foothold in even the most well designed systems, and this possibility must always be taken into consideration.

Attack from outside intruders with these tools is not directly relevant to my focus, but a clever rogue admin may use such tactics to hide their identity and grab credentials of another user. Regular users would be useless, but the credentials of another admin would be very useful if the rogue admin wished to take an indirect route in their desire to acquire data. So while the actual act of a social attack is beyond my scope, its potential effects should be taken into account.

2.5.2 The Domino Effect

When looking at how systems communicate with one another over a network, you think of the OSI reference model. It is designed to enable the different layers to work independently from one another to accommodate changes in the evolving technology. Each layer is responsible for a specific function, and information flows up and down each subsequent layer as data is processed. Unfortunately if one layer is compromised, communications are compromised without the other layers even being aware of it. A breach in the physical layer would percolate up and compromise all layers above it. Security is only as strong as its weakest link, and any implementation I make must attempt to be equally secure at any level it touches. I would necessarily have to rely on lower layers already being sufficiently protected, and have knowledge of their current state at KRG.

2.5.3 The Security Wheel

Probably the most important thing to keep in mind is that network and information security is an ongoing process that goes through a continuous cycle of developing, implementing, monitoring, responding, testing and improving. An implementation today may not be sufficient in 5 years, and keeping up to date is a big task.

2.5.4 Virtualization

Virtualization(Marshall, 2011) is a rapidly growing trend in modern systems, as it provides an easily implemented and easily manageable environment in which to compartmentalize computing tasks, servers and networks. It is energy efficient, reduces data center footprints and potentially increases uptime. With a central control panel you can quickly get an overview of the health and performance of every virtual server in your business and be able to perform maintenance from a central location. VM's are mobile, robust and cloneable, making disaster prevention and recovery a simple task, and the ability to sandbox any application or system is very useful when testing or benchmarking. KRG is following the trend and has implemented virtualization on several of its systems with great success. It might be useful to implement some form of virtualization layer, as it could possibly enable proper control while hiding the actual contents. In which case System Administrators would then have to be able to get it back up and running without being able to change its functionality in order to circumvent its restrictions.

2.6 - Security in programming:

When creating a new program it is very easy to just push it out and ignore any potential unwanted interactions, as is quite normal when programming for an assignment or for your own personal use. Applications do not live in a vacuum however, and in order to prevent unwanted reactions you should always keep secure coding practices in mind(Seacord, 2011). Coding securely is also known as defensive programming, and the art of programming defensively is simply to assume that every call and every operation may be given incorrect output and thus must be able to handle such an eventuality. In Java(Oracle) you would use a try-catch implementation, or perhaps you would use extensive if or case chains. The result should still be that if the incorrect input is given the program should not suffer any problems as a result.

A few things worth keeping in mind:

- Validate the input, especially from untrusted data sources. Abusing unchecked inputs are the most common ways of exploiting vulnerabilities.
- Heed compiler warnings. Your code may run, but that memory leak might come back to bite you.
- KISS. Making overly complicated code is unnecessary, can make it harder to spot mistakes and can open for more and different exploits.
- Check your privilege. Don’t give your program root access just because it is easier. Try to keep your application at the lowest privilege level as much as possible, and secure any point at which it enters higher privilege levels.

- Isolate unrelated code. Closely tied to privilege, ensure that code from different sources is compartmentalized and private access is not given publically.
- Release unused resources. The garbage collector might pick it up, but before it does someone might find a way to exploit its existence.

There are many other more specific points to heed, but these are some of the biggest, and some of the ones that tend to get overlooked a lot. Often simply because it requires more effort.

2.7 – Summary

This chapter covered several concepts important to the understanding of security. The CIA triangle is at the very core, and permeates every consideration of security. As a result they will have to be kept in mind going forward.

The concepts of internal security in the form of policies, standards, procedures, baselines and guidelines are all important, and knowing which are applicable to a given situation can mean the difference between a great system and a broken one. These concepts are of course very dependent on what you are implementing and where you are implementing it and in the case of this thesis the ones currently in use at KRG will have to be included along with any which are specific to the programming language and framework that will be used.

Perimeter and layer security is useful to know when considering access from the outside, and is a necessary consideration for any system in a modern business. This thesis will only concern itself with data once it is in a system, and thus perimeter security will not be a relevant concern.

As for the other aspects, some are more important than others. The social aspect will be very important to keep in mind going forward, as even accidental and tangential access are undesired. Security in programming will also naturally be of central importance. The domino effect and security wheel will not be entirely unimportant, but will fall mostly outside the scope of the thesis as it currently stands. Finally, virtualization will likely not be considered further as the implementation will be on the system level, not on the user level where virtualization is currently in place at the KRG.

Chapter 3 – Security at the Cancer Registry, the Context

KRG is an institute in the Norwegian healthcare sector which gathers data on all patients being diagnosed and treated for cancer in Norway, processes it such that it is stored in a structured manner, then presents statistical reports and does research based on said data. Each healthcare facility that performs diagnostic tests of and treads patients for some form of cancer, be it of pathological or clinical nature, is required by law to send a report to the KRG, which ensures maximal coverage and accuracy of data used for research.

Many businesses, including KRG, operate primarily on a system of necessary trust when it comes to their System Administrators in respect to security. This is often enough as a breach of this trust usually entails very harsh punishment such as loss of employment, criminal charges and not being able to get similar employment in the future. Despite this, every so often a rogue administrator will break the trust and use their powers for some personal gain, be it to earn money, get revenge or simply cause havoc. Even with proper security measures, at some point trust must be a part of the equation, and the trick is to find the balance where there is adequate security to prevent disastrous breaches while also keeping restrictions from making it impossible to work efficiently. In an attempt at improving security protecting against rogue admins at KRG, I will first have to get an overview of the current security measures in place at KRG, as well as which systems interact with the data flow of personally identifiable information. Once this is completed I will document which technologies are used on the relevant systems and decide on the optimal approach for a solution.

Beginning then with the current security measures at KRG. As with many businesses, there are several layers and facets to the security system in place at KRG. The most abstract division is between physical security and IT security. Physical security is simply the physical access restrictions to hardware, while IT security covers system and network security against both outside and inside threats.

3.1 - Physical security

Physical security is out of the scope of my thesis, but is included for completeness. At KRG the primary form of physical security is through the use of access cards. Each employee has their own access card, which are currently authorized and distributed by the administration. Outside of working hours a pin code is required in addition to the card, simply to prevent abuse of lost or stolen cards. A select few personnel have access to additional internal doors, such as the server room. Access to the server room is restricted to IT management, and is where the server relevant to my thesis is located. Card access through the outer doors into KRG is logged by the building security.

3.2 - IT Security

3.2.1 - Network structure and zones

KRG operates with a multilayered security structure on the network. Beginning from the outside:

The internet

- An outer firewall
- A DMZ, hosts services accessible from the internet such as the website and webmail.
- An internal firewall
- The internal network.

On the internal network there are several 'zones', also defined as LAN's or subnets, each of which have separate rules for access and perform different roles. The main zones of interest are *Internal*, *Sensitive*, *Screening* and *Management,* the structure of which is shown in Figure 1. In addition there is a second internal DMZ which use will be described later.

-Internal:

The *Internal* zone is where most of the user clients are located. On this zone you have access to all the daily tools you may require for most job related tasks. You have access to relevant software, to your own 'home' folder on the network, various shared project folders and drives, and the internet. Users often have restricted control over their computer, with only some few having local admin rights.

-Secure:

The *Secure* zones are where all sensitive data is hosted. The secure zones are technically two zones comprised of Sensitive and Screening. Several virtual servers host databases with such data, any software or services required to manipulate sensitive data when required, as well as sensitive files stored in file servers. No Internal zone clients are connected directly to the *Secure* zones, and they is blocked off from the internet except through secure VPN tunnels. User activity is severely restricted on any *Secure* desktops.

-Management:

This zone is only used by System Administrators, and is primarily used for management of the other zones and their content. *Management* is completely separated from the internet. Activity on *Management* machines is restricted for normal users as in the other zones, but if you log in as a domain admin you are in essence not restricted at all.

3.2.2 - Connections between zones

The rules that govern the connections to and from zones as well as between zones is controlled by the internal firewall. The rules cover four aspects:

The routing, which is the physical routing of traffic. What goes to which subnet.

- The tunneling, ensuring that physically separated locations can operate in the same zone, usually through VPN's.
- The ports, which ports can be used.
- The policies, who and what can access which services and when.

The firewall logs all network traffic between zones. Domain admins can however manipulate the rules and logs on a firewall.

In addition to the firewall which governs the connection rules, there is the second internal DMZ which controls several services. These handle communications between zones, antivirus scans of files coming from the internet and moving between zones and pushing updates to clients. These services operate within the framework of the firewall rules.

Finally users can connect to internal zones from the outside through the use of a Virtual Desktop Infrastructure (VDI) through a VPN tunnel. These connections are quite restricted and require a single use code from a Rivest, Shamir and Adleman (RSA) authenticator.

Clients on the *Internal* zone can connect to the *Secure* zone through the use of a VDI solution. The VDI allows a user, sitting in for example *Internal*, to log into the *Secure* zone and work there without risk of unwanted interactions between the user's computer and *Secure* servers. In order to transfer files between *Internal* and *Secure* you need to use a secure file transfer system which checks each file for integrity, for malicious content and logs the transfer. There is a size limit on the files allowed through the system, and any larger files need to be manually transferred by a domain admin. The domain admin then typically performs an integrity check (md5 checksum) and AV scan before transferring. This form of transfer is not logged specifically, though any traffic between zones is logged in the firewall. As with *Internal*, users can connect from the outside with a VDI through a VPN, using login and an RSA authentication code.

The *Secure* zones are very strictly controlled, and as such have limited communications in and out of it, as already partially described above. In addition to these connections it has a portal connection through which clients at hospitals around the country can access certain agreed on data by logging in with authorized credentials, and send files for storage in the *Secure* zones. Any files the hospitals send or are allowed to access are placed in the portal, and then have to be manually transferred back and forth by the domain admins. Within the Secure zones, Screening is a secondary zone used by employees who screen for cancer at various locations around the country and is outside the scope of this thesis.

Additionally there is a zone which is secure, but not affiliated with the *Secure* zones as such, called NHN or National Health Net. Internally this zone acts as a DMZ to outside systems in which external operators within the healthcare sector can gain access to resources internally at KRG. More specifically it is a VLAN or Virtual Local Area Network which allows for transfer of sensitive data without risk of security leaks. The connection from KRG to the outside goes through HCSIRT, or the Health Computer Security Incident Response Team which dynamically monitors data traffic to intercept any attempted breaches in the healthcare sector. This zone is out of scope and will not be discussed further.

Management is a zone apart from the rest. It has no access to the internet, but has otherwise complete access to all other zones. A domain admin can access every zone and manipulate them and their content as required or desired. The only cases in which additional login credentials are required is if an admin wishes to connect to a specific server as a user. Admins consciously keep *Management* access points secure and use them only when required, but once logged in the activity on this zone is not restricted or specifically logged beyond the normal logging done by the firewall and local event log.

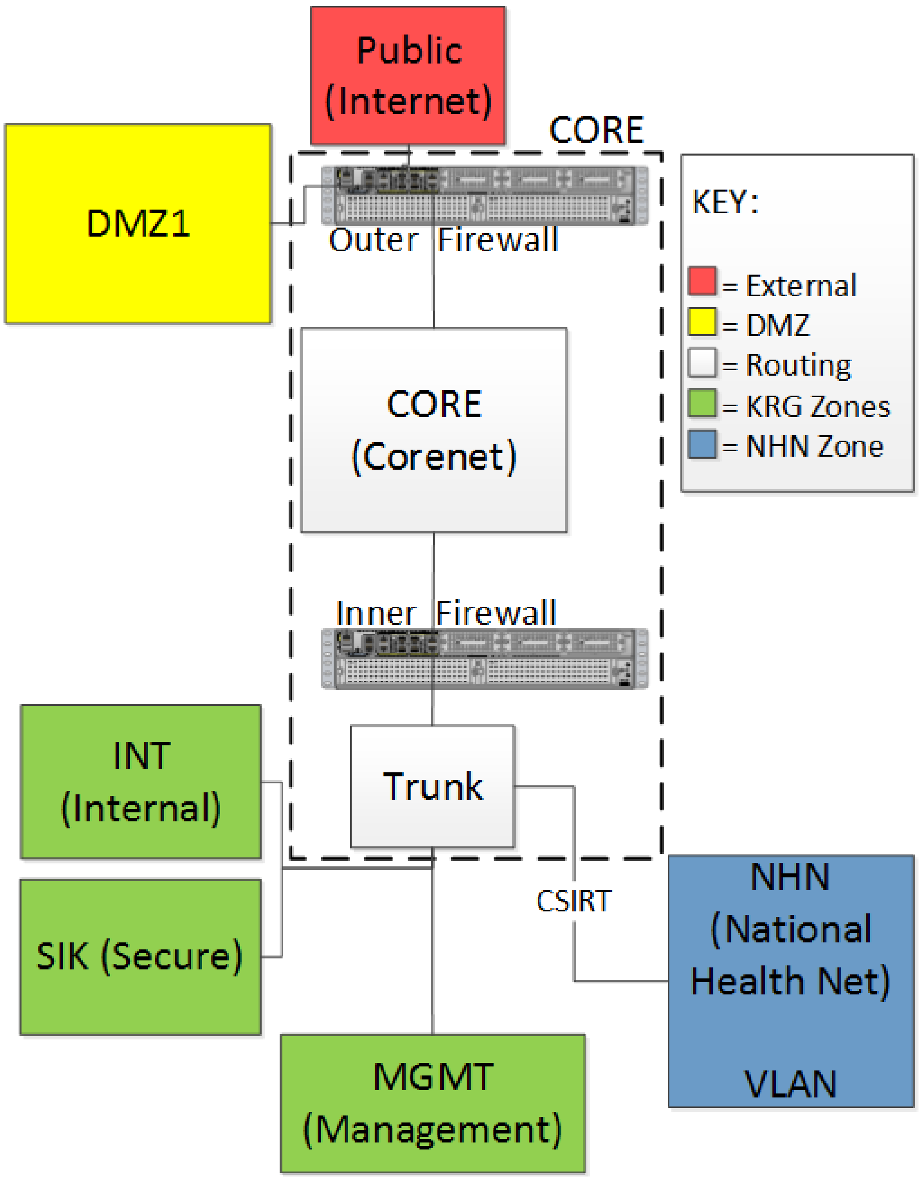


Figure 1: Structure of internal network at KRG, with relevant zones.

3.2.3 - Credentials and access restrictions

User credentials are controlled by a Windows Active Directory Controller (ADC). The ADC checks the entered username and password every time a user attempts to connect to a zone either internally or through a VPN, or use a cross zone service such as the file transfer service, and either allows or denies the connection. All logins are logged. User credentials are authorized by the KRG administration and entered into the ADC by a domain admin. Below is a diagram which illustrates the basic structure of permissions as controlled by the ADC at KRG.

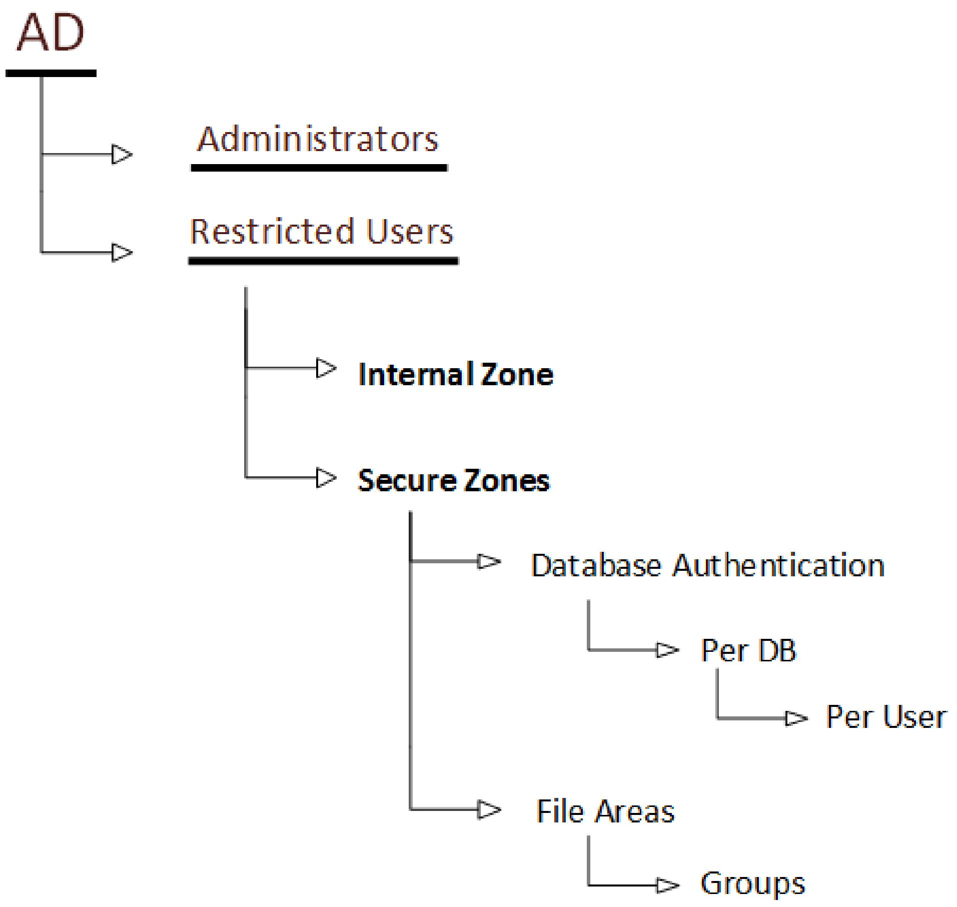


Figure 2: Authentication Structure at KRG

The authentication structure for databases and file areas under the Internal Zone is the same as for the Secure Zones, they are merely collapsed in the diagram. The structure for databases is also the same under Administrators, as each database has individual authentication irrespective of the ADC.

In addition to the ADC, databases have their own access tables which govern permissions specific to that database and these are managed by Database Administrators responsible for any given database.

Finally, there are some systems which have internal authentication that are apart from the control of the ADC. For example the *Sensitive* zone has a database which contains the credentials of users which are allowed to connect to it, which the login service matches to.

If you connect a computer to an Ethernet port inside KRG, without login credentials you currently have access to the internet and can see shares and machines on the internal KRG network, but cannot access them. This behavior will be changed in the near future to a required login and security scan even internally in the building. This because of the surge in personal computing devices which employees desire to connect to the network and the difficulty in ensuring their integrity without checking them manually first along with the possibility of then accessing your own shares from personal devices.

3.2.4 - The sensitive data

When a cancer message arrives electronically or is scanned into the system, they are stored for further processing. The personally identifiable data, the ‘personal number’, is then masked from the scanned images, encrypted and the result is placed in three different locations. The unmasked original image remains in a file system which has restricted access. The masked image, which is used for further research and statistical analyses, is placed in a Structured Query Language (SQL) database. In addition the personal number is encrypted and placed in a separate SQL database along with metadata extracted from the report image. The relation between the masked and unmasked image as well as the encrypted data is handled by the FILM-IMAGE number of the relevant image. Only in rare instances will a user be allowed to gain access to the unmasked version of the image. Any queries to the databases and any group or user modifications are logged.

Looking at the current situation at KRG, it appears that there are areas where the current system is vulnerable should a rogue admin decide to take matters into his or her own hands. Looking at the zone structure, the most obvious line of attack appears to be through the management zone as this grants admin-level access to the other zones. Once in the management zone,

an admin could create a dummy-user with admin rights, add it to the permission table of the databases containing sensitive data and use it to gain access, covering their own tracks as far as any logging of database access goes. If they further deleted the system logs in management it would be further possible to cover their tracks in the creation of the account. In any case, whether by accessing using their own or a dummy account, access to management is the biggest weakness of the system as it currently stands.
Despite this, the management zone must exist in order to have the ability to properly manage and maintain the system efficiently, so a solution must work with this in mind. I believe that the problem can be solved with only a few added measures. The most important is a more thorough logging of access and use. Currently, system logs are the default logging of the windows environment, the logs inherent in the Active Directory software used to manage users and access logs on each individual database. This could be expanded by implementing a monitoring system that logs the actions taken by System Administrators while performing their duties using the management zone. Naturally, such a program along with its log would have to be protected from tampering and be robust enough to endure any attempt at fooling or corrupting it.

3.2.5 – Policies

The Norwegian healthcare sector, including KRG, implements a standard known as 'Normen'(Normen, 2013) (more formally 'Norm for informasjonssikkerhet' or 'The Information Security Norm'), which is built on the ISO 27001 standard, and expanded to fit the legal and formal requirements for the handling of patient data and management of health information systems in Norway. This standard describes the requirements for any computer system within Norwegian health care, from the abstract overview, through the implementation and to how to guarantee the implementation works as intended.

3.3 – Summary

This chapter covered physical and IT security at KRG, only briefly mentioning physical due to it being out of scope while delving into detail surrounding the IT security.

The zone structure and their interconnectivity was described as it is vital to know where the sensitive data of interest is located in the secure zone, and how they are accessible, especially from the management zone. This is a vulnerability to consider.

How credentials are managed and understanding how to utilize them in the implementation will be very relevant.

The sensitive data and where it is located in the system is what the entire thesis revolves around, so this knowledge will be integral during the implementation process.

The final part discussed in this chapter, the ‘Normen’ policy will naturally have to be kept in mind going forward.

Chapter 4 – The KRG IT Environment

Before implementing any of the solutions proposed, a more in-depth grasp of the systems used at KRG is needed, which in concert with the security structure will determine my implementation approach. This chapter will delve deeper into the specifics of the data flow, the data processing, information systems and information technologies used at KRG. This is essential to narrow down my scope to the relevant systems of interest, as well as finding out which approach will be optimal to achieve the desired effect.

For this chapter I will use the structure provided in 'The Open Group Architecture Framework' (TOGAF) as a platform with which to divide the information into manageable categories(Josey, 2011). TOGAF is a framework which is made to provide an approach for designing, planning, implementing and governing the information architecture at an enterprise level. It is high level and holistic, typically modeled at four levels: Business, Application, Data, and Technology, which makes it ideal for getting a clear overview of a system in a structured manner. The structure of the TOGAF Architecture Development Method (ADM) can be seen in figure 3.

I will focus primarily on phases B, C and D, which are Business Architecture, Information System Architecture and Technology architecture respectively.

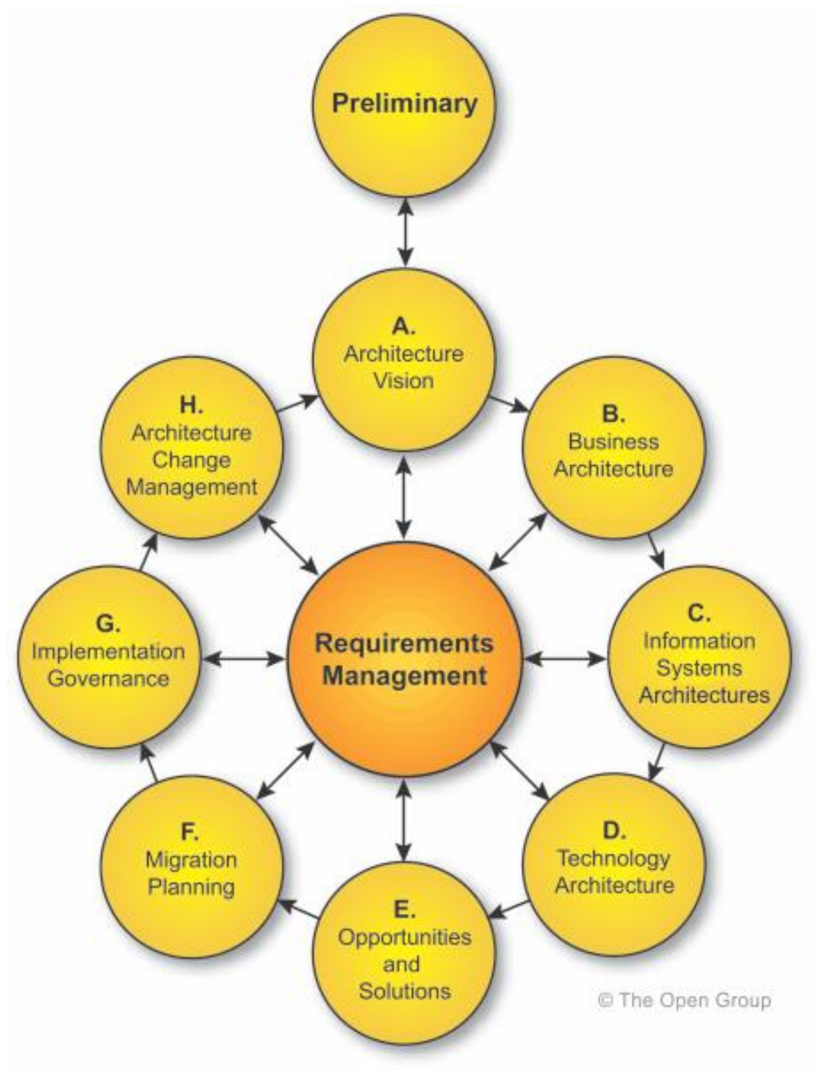

Preliminary
A. Architecture Vision
B. Business Architecture
C. Information Systems Architectures
D. Technology Architecture
E. Opportunities and Solutions
F. Migration Planning
G. Implementation Governance
H. Architecture Change Management
Requirements Management
© The Open Group

Figure 3: TOGAF requirement wheel

4.1 - Data Flow

Looking at the data flow of the pertinent data, I will present an overview of all systems which personally identifiable information interacts with. First I have a diagram of the data flow at a Business Architecture level, or phase B, which shows the data flow with respect to both relevant entities and time. This was created with the Business Process Model and Notation (BPMN) language, as this is the standard used for modelling business processes at KRG.

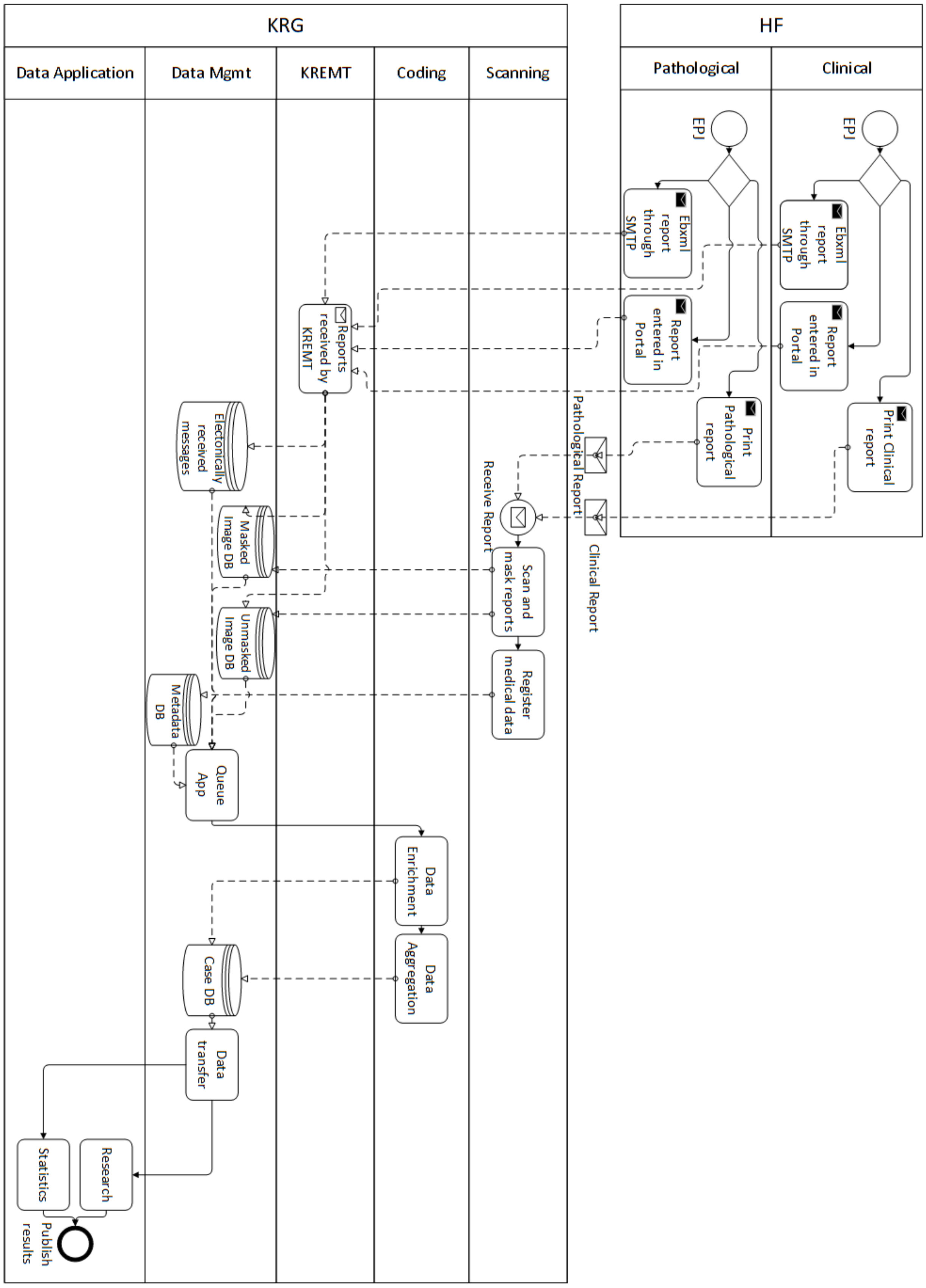


Figure 4: Data flow of clinical and pathological reports at KRG

Figure 4 shows the business processes involved in data flow of personally identifiable data at KRG. The reports regarding diagnostics or treatment originate from the various hospitals, here labelled as HF(HelseForetak or Health Institutions), and are created in the EPJ(Electronic Patient Journal). A copy of these is sent to the KRG either by printing them and shipping through regular mail, through the use of an electronic mail service or by filling in a report form on a portal hosted by the KRG. Electronically received messages are processed and the metadata placed directly into databases while the physical messages are scanned, masked and the metadata typed into an application called MottattMelding which dumps it and references to the scanned images into the relevant databases. The data is then enriched and aggregated for further use in statistical work or research.

4.2 - System Architecture

Moving on to the Information System Architecture level, or phase C, I have created a diagram using Unified Modeling Language (UML) which shows the data flow with respect to the information systems that interact with it.

Figure 5 shows the current structure of the systems and databases relevant to the message flow at KRG. Physical messages are scanned and processed before the relevant data is stored in a Metadata database while the personal number of patients is stored in encrypted form along with this metadata. The data is then enriched and further assigned to 'cases', which can then be used for research or statistical analysis. Each person has at least one, and potentially many cases and each case describes one incidence of suspected or determined cancer. Different forms of cancer in a patient are defined as 'separate' and become individual cases, except where one is a metastasis of another in which case they can be grouped together in one case. There are numerous complicated grouping rules beyond this, but these will not be described further as they are out of scope for this thesis. Note that each case may have several messages assigned to it describing everything from discovery to treatment.

The dotted lines outline the borders of the two main systems for managing messages at KRG as they will be structured in the future. KREMT (Kreftregisterete E-MeldingsTjeneste), which translated means 'The Cancer Registry's Electronic Message Service', will handle the arrival of messages and the processes involved in that, while a new system called KNEIP(Kreftregisterets Nye Elektroniske IKT Platform), which translates to

'The Cancer Registry's New Electronic IT Platform', will cover KNEIP's old roles as well as handle message transportation and further processing and management. These are further explained in the next subchapter.

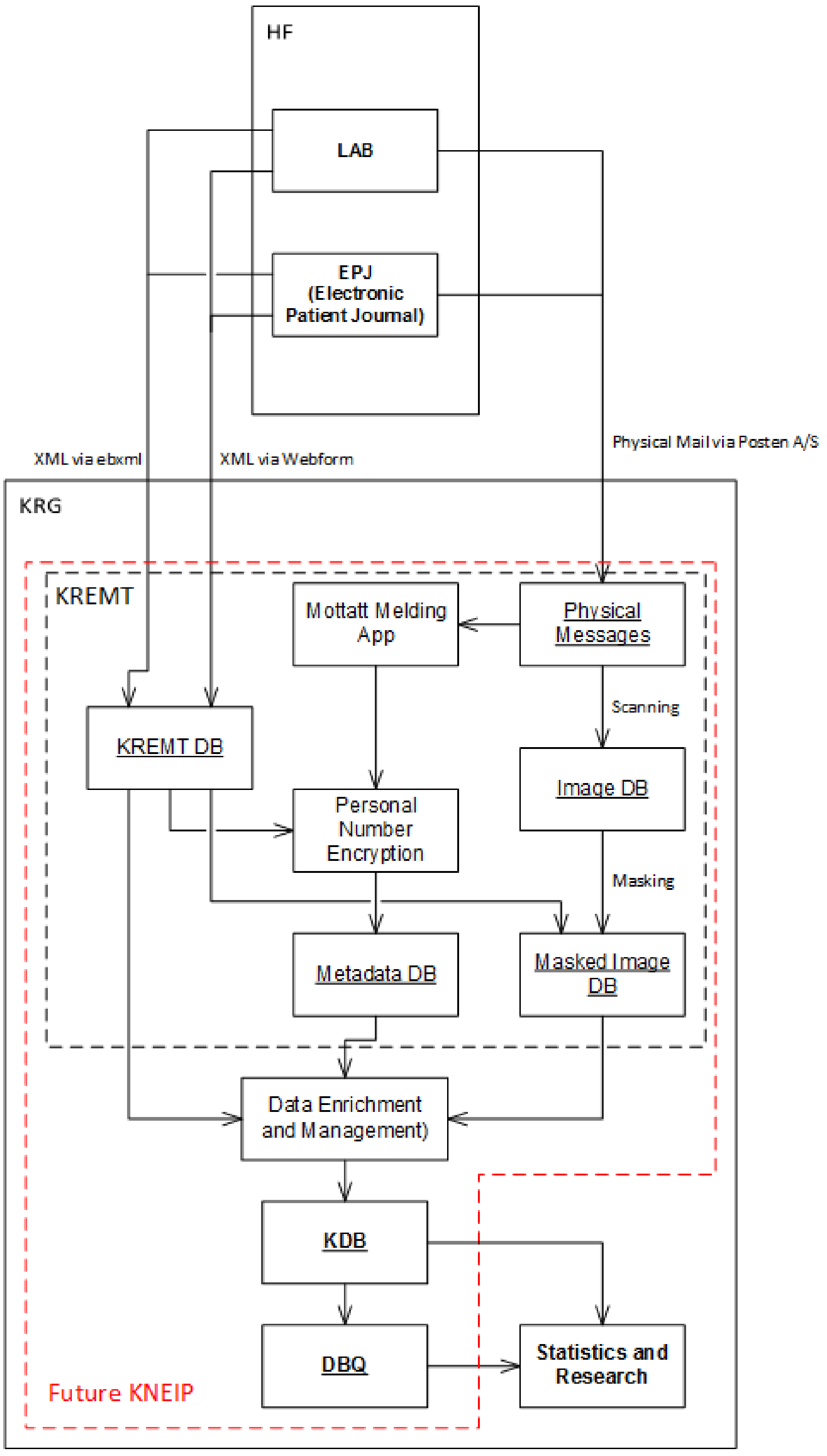


Figure 5: Systems architecture overview

4.3 - Technology Architecture

Finally we get to the technology architecture that define the systems at KRG. Currently, the system that handles messages and reports that arrive at KRG is KREMT. Initially I was going to use this architecture as the basis for my implementation, however KNEIP is currently being developed and will replace parts of KREMT in the near future. KNEIP has a broader scope than KREMT, as it will also replace other systems at KRG and become the de facto data management system in the future. A part of this is the combination of the existing databases into one logical unit. Where there now is KDB and DBQ, two separate databases, there will in the future be one database, consisting of a merge of the old with some alterations, which will be integrated into KNEIP instead of being a separate entity as the current ones are to KREMT. KREMT functionality will also eventually be replaced by KNEIP, and will receive messages arriving at KRG which are then passed on to other modules for further data processing and management.

First a few practical points. KNEIP is being developed in Java and is using the Spring framework during development. Since a development environment is already up and available, this decides the issue on implementation language, environment and framework for me quite readily. Also, KNEIP is not yet fully functional, so assumptions will need to be made in relation to functionality and services. I will be discussing and getting feedback on any such decisions with the development team so as to conform to their visions for the KNEIP system.

4.3.1 – Access management in KNEIP

Access management in KNEIP is handled by internal services that use Active Directory (AD) for authentication of users, and an internal table exclusive to each database for access rights and user control. As an example a user (or administrator) logs in to the KNEIP system, his user is checked in AD and is authenticated as a legitimate user. Once logged in the user attempts to access a specific database, and then his access right are checked in the access table. The user is granted access and the request is sent to the relevant services which manage the requested interactions.

4.3.2 – Spring

Spring is the framework in which KNEIP is being developed, and is necessary to understand when going forward with the implementation. It is a comprehensive programming and configuration model for modern Java-based enterprise applications. It provides infrastructural support at the application level, focusing on the ‘plumbing’ of enterprise applications so that focus can be given to application-level logic. It provides dependency injection, aspect-oriented programming, relational database management, authentication, logging and much more.

4.3.3 – Java and Maven

Java is a high-level, object-oriented programming language similar to C++, but simplified to eliminate language features that cause common programming errors. Java source code files are compiled into an architecture neutral bytecode format, which can then be executed by a Java interpreter, also known as a Java Virtual Machine(JVM). Since the program runs on the JVM and not directly on hardware, it is portable across all platforms that have a JVM installed, removing the necessity for recompiling code for separate hardware architectures. This is only part of the story though, as the compiler also combines similar code, shifts code around and changes the names of everything to fit its own logic. The compiled code does in fact often look utterly different from the source code, the only guarantee being that the execution order and the consistency of outputs with a given input are guaranteed. In addition, once the JVM runs the bytecode it may decide to further compile the bytecode of often used code into machine code to increase efficiency using a form of Just In Time compilation(JIT). This along with clever selection of stack and heap placements, cache management and threading java code once run can be as efficient as code created in the lower level C language, only without the need for any memory management due to the Java garbage handler which continuously runs in the background purging the heap of unlinked data.

Maven is a build management tool for java development. It enables full configuration of build paths, compile outputs and packaging structures along with automatic downloading and integration of required libraries, bean control and a plethora of third party plugins for expanded features. It uses Convention over Configuration, so that you don’t need to define the tasks you want to do as long as the files are where Maven expects them to be or as defined by the developer.

4.3.4 – Other technologies

In addition to spring, KNEIP uses several other technologies. It uses Apache Camel for open source integration, routing and mediation rules, Query dsl for type-safe SQL queries, Hibernate for JObject and Relation persistence and query services, Tomcat for the application server, Apache Active MQ for the integration patterns servers and Vaadin for the Java web application framework.

The databases are Sybase ASE servers which use a standard SQL syntax.

4.4 – Summary

In this chapter the data flow, system architecture and technology architecture of systems related to the relevant sensitive data was presented. The data flow and system architecture are quite important as they provide an overview of where the sensitive data is stored and moved within the systems at KRG, which was central in finding the point at which a solution should be implemented. It is clear from this that the implementation should be made within the new KNEIP system, as it will be the system which interacts with the personally identifiable data.

It then follows that the technology architectures presented are all central to deciding on how to implement the solution. Knowing what controls the authentication, the Spring framework and Java language the KNEIP system is developed in and which other technologies that may be relevant will be directly relevant going forward, as will become clear in chapters 6 and 7.

Chapter 5 – Existing Solutions Targeting Rogue Administrators

Before continuing on to deciding on a solution and implementing it, it would be prudent to have a look at existing solutions which are currently in use as parts of the effort at restricting the ability of system administrators abusing their power.

As computer systems and networks have progressed, there has been an ever expanding battle between hackers who wish to disrupt services, destroy systems or steal data, and the people who do their best to protect against such malice. The tools of the trade become progressively more advanced and comprehensive, defending against a wider variety of attacks and tightening the noose, but with a constantly changing landscape of systems and services it is a battle that will never truly be over as new technologies and software bring new weaknesses to be discovered and patched. Because of this constant threat from the outside, the development of defenses against insider threats has not been in focus in quite the same way. While comprehensive security measures do take insider threats into consideration, a single administrator can cause catastrophic amounts of damage to a company's data integrity, data confidentiality and reputation if there is inadequate security protecting against such threats in particular.

5.1 – Common solutions

There are several solutions which aim at preventing the above mentioned rogue admins(Messmer, 2010).

5.1.1 – Role compartmentalisation

One is a strict compartmentalising of administrator roles. While this can give extra security against tampering and make it harder to access certain data, it will not in fact eliminate the possibility to access personal data as we would want. Also, giving no one administrator complete access means that several administrators are required to perform certain tasks and it makes each person more invaluable and reduces redundancy, something which might be undesirable with the small size of KRG's IT staff. If one administrator was ill and was needed for a critical operation, it could cause other security issues and seriously hamper the operation of the computer systems.

5.1.2 – Secure logging

Another, which also does not directly prevent administrators from accessing data, is monitoring the activity of System Administrators by use of logging that they cannot disable or altered. This may discourage acts of 'curiosity' and give leadership the ability to get an easy oversight of what is being accessed, how, when and by whom. This can be used to great effect as a way to prevent some undesired access and have a log of who accessed any and why. Since System Administrators are bound by a confidentiality agreement, this may be sufficient in the eyes of the law, as it gives the ability to audit the logs to ensure the confidentiality of patient data. However, this has a 'big brother' feel to it, and can breed distrust if a System Administrators decisions are questioned by leadership who do not have knowledge of how the system works. Trust is important in an organization such as KRG after all.

5.1.3 – Automated pattern analysis

A third is similar to the second, but removing the human monitoring element except in cases of suspicious breaches. An automated monitoring system which looks for patterns of access, automatically blocks access or requires confirmation from multiple admins to perform certain tasks. This is a more acceptable form of oversight as far as trust goes, while still having the required monitoring. This could help greatly in restricting access to the relevant personal data, but again it does not completely eliminate it as regulations require.

5.1.4 – Encryption and key management

Looking also at the more conventional solution of data security which we already employ, encryption, there are some simple ways to enhance its effectiveness. Since System Administrators often have access to algorithms and keys, encrypting data will not always work. One solution could be to require multiple keys to access certain data, ensuring no one admin has both. Another would be to entirely remove System Administrators from the algorithm and keys by delegating the responsibility to a specialized employee, or a member of the leadership whom has 'clearance' and can be trusted to handle them properly. While admins can still copy the data if no further levels are implemented, at least it will be worthless without proper decryption keys, especially if encryption tokens have timeouts.

5.1.5 – Multi person control

Going in an entirely different direction from the compartmentalizing of System Administrator roles as discussed earlier, a two-person control model, also known as a four-eyes authentication model, can be applied. In such a system one can mostly offset any attempt by malicious administrators, as certain administrative tasks will always require two people to complete. This system could also be extended to data and encryption access as outlined in the previous paragraph. If any single administrator can only see half of any data, algorithm or key, it becomes a jumbled incoherent mess which is a relatively simple way to make actual breaches much harder to pull off. The cited paper also points out how role separation can be circumvented by System Administrators altering their own roles to fit their desires, a valid point which would have to be separately evaluated.

5.2 – Summary

There appears to be no single way to achieve the ideal goal of completely removing System Administrators from personally identifiable data, as the very nature of their jobs require a level of access that enables them to access and tamper with this data in undesirable ways. As explored above, there are existing ideas on how to restrict, monitor and protect data, and to achieve a satisfactory level of security a compilation of several of these is likely needed. There is of course always progress in the field of security in general, and looking at new ways to apply current knowledge or new avenues of thought looks to be an interesting prospect. With further research, done over time with better resources and more depth there are certain to be some way to handle the somewhat unique problems facing the KRG in this particular area, though there is consensus that at some level the human element simply has to be trusted to follow the rules set before them.

A possible approach is twofold in addition to the above mentioned; a combination of automated logging of interactions with critical personally identifiable data and a multi-person control model. The keystone that then remains is the access to the encryption algorithms/keys, logs and administration of said, as all the preceding is moot if an admin can circumvent it by going in the backdoor. To close this hole the multi-person control model could be implemented in such a fashion that some information and some interactions can only be accessed and performed with input from more than one. There are other avenues of approach, and they will be researched and assessed in order to find the optimal solution for this problem.

The implementation challenge will be learning about how the final approach can work in concert with existing frameworks and environments and maybe how to implement certain unique data manipulation tools. This will be discussed in detail in the next two chapters.

Part III

Securing Sensitive Data at KRG

Chapter 6 – Deciding on a Solution

There are several possibilities for a good solution at KRG, but deciding on one depends on several factors. As mentioned in chapter 5, there is a fine line to tread when ensuring there is enough security, without making it too overbearing and constricting. At some level there needs to be some form of trust, else it will be impossible to maintain the systems in question. In addition, in the case of this thesis, it cannot be too comprehensive as there is simply not enough time to cover every possible scenario. As a result I will be making several assumptions, narrow the scope of my implementation and define a few requirements outside this scope which would need to be met were it to be deployed.

6.1 – A completely secure system

At one extreme it is possible to make security water tight, but this will make it utterly impossible to maintain. Let's go beyond simply an isolated computer with input only and look at a scenario where you have a database which manages the confidential data and an application which is supposed to be the sole method with which to normally access it. This scenario is shown in figure 6.

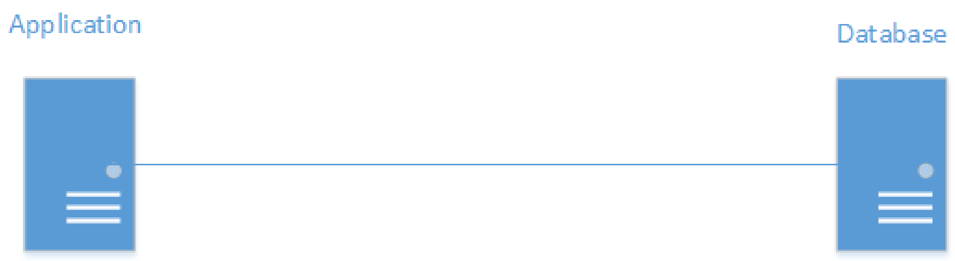


Figure 6: Application connecting to a Service

- Storing the confidential data:
    - Firstly, it is possible to create an encrypted database which is practically impossible to decrypt within our lifetime, simply by applying a strong enough cypher. This is trivial.
    - Secondly, it is possible to have an access table in the database which cannot be accessed or edited by anyone, by creating the

    list and then making the user which created the database inaccessible by corrupting its credentials. This is theoretically possible through byte corruption of the database data at the location of the root user credential storage.
  - Thirdly, before corrupting the administrator account, it is possible to ensure that no user in the authorized user list has any form of administrative access.
  - Lastly, you can ensure that all users have passwords that are strong enough to withstand any attempted brute force break.

- Accessing the database:
  - It is possible to set up a secure tunnel between the system and the database to avoid any attempted man in the middle attack or similar intercept, even from the management zone. To do this, you set up the tunnel between the system and database, creating the encryption keys on a standalone computer which is not connected to any network to eliminate any possible tampering. Extract the keys with a USB, apply them and then wipe both the computer and USB. You also have to ensure the database only accepts connections through this tunnel, which would have to be done before the database management user was corrupted.

- Securing the system:
  - Firstly it is possible to create a security application which intercepts and monitors any services which interact with the secure connection to the database, and perform checks to ensure they are legitimate. The source code would have to be kept confidential and either deleted once complete or isolated on a non-networked computer. This to prevent tampering with and 'updating' the module.
  - Secondly the entire system would have to be created to spec, ensuring no methods which are not specified in the security module have access to the database and then the same isolation or deletion would have to be done.
  - Finally any user credential passing would have to be encrypted to avoid in-system intercepts before they are sent through the tunnel.

You now have a database, which is theoretically impossible to break into, and impossible to access unless you were among the initial authorized users, but impossible to maintain or repair should it become necessary. Also, you can never create any new authorized users. You also have a secure connection which cannot be intercepted but again cannot be changed or maintained in any way. You finally have a system which is impossible to access and spoof, but locked and impossible to maintain or upgrade.

Assuming the job was done correctly, you can now be almost sure that your data is secure from access by System Administrators, unless the authenticated user credentials are compromised, but if anything were to go wrong or any changes needed to be made you would be stuck. This is clearly an untenable situation which cannot be the basis for any sound system.

6.2 - The powers of a System Administrator

The example of a secure system given in 6.1 – A completely secure system may sound more extreme than necessary, but many fail to realize the absolute power a full access System Administrator has if they have the requisite knowledge. They are not bound by the systems in place that a normal user has to utilize. Even ignoring physical access to the servers, they can access all services directly, completely circumventing most security protocols. They can disable or tamper with any security protocol they cannot circumnavigate, making it look like nothing ever happened. They can in the same vein frame other users, leaving no trace of their involvement. They could set up invisible listening programs which intercept and decrypt internal communications, since they have the keys, compromising credentials and services.

In short a System Administrator can access anything, period. This is why most businesses compartmentalize roles such that System Administrators are not truly full access, such that they do not possess encryption keys or only have access to some services etc. Even with some limitations like this, given the skill and enough time they can usually find ways to gain access as long as they have the powers needed to maintain the system properly.

6.4 – The relationship between KNEIP and the Personal Number Encryption Algorithm

As mentioned in 4.3 - Technology Architecture, a collection of old systems will be replaced by KNEIP in the near future, and once it is in place KNEIP will be the only system which has direct access to the service which manages the encryption and decryption of personally identifiable data. This service will henceforth be referred to as the Personal Number Service(PNS). The System Architecture structure will look something like what is shown in Figure 6 with KENIP replacing the application and the PNS replacing the database with the addition of a management zone which stands astride both the KNEIP system and the PNS as they are located in the Secure zone.

In the strictest sense, the PNS is included in the KNEIP system, but it is physically separated and thus it is beneficial to imagine it as such. KNEIP will handle all daily connections while Management has the ability to connect to either in order to perform maintenance and manage the hardware. The critical point to take away from this is that the implementation should be such that the zone which access is attempted from should have no bearing on the integrity of the security measures.

6.4 – Drawing the line

Knowing what was discussed above, I have to make a few decisions and define the scope of the implementation.

Firstly, System Administrators must be able to do their job. It is entirely untenable to overly restrict their ability to perform maintenance and manage the systems which they are obliged to service and make available to the people and systems at KRG. I will therefore not implement any security at the lowest level. Any changes at this level will have to go through the KRG leadership to be decided upon, and will have to be implemented as new policy to be enforced.

Thus, if the implementation is to provide an adequate level of real-life security, it will be dependent on a few changes which would have to be decided on as policy and deployed. While this is not something I can directly influence or implement, it is central to the integrity of the solution and eventual implementation and must therefore be presented in full.

There are some prerequisites to this effect:

- The PNS module is a part of KNEIP and cannot be accessed other than through services within the KNEIP system.
- The PSN algorithm is hardcoded and encrypted, thus modification cannot be done on the fly.
- A database to be used for secondary logging has a secure connection to KNEIP.
- This database is managed by a dedicated Database Administrator.
- Any connection used to administer the logging database must occur through the KNEIP system.
- Logging Database Administrators cannot edit the data sent to the database, only read it.
- The encryption keys used to set up the connection and the Database Administrator credential vault are managed by personnel separate from the System Administrators.
- The algorithm and encryption protocol for the PNS are managed by separate personnel which do not have other Administrator powers which would enable them to circumnavigate the general security and logging currently in place.

These prerequisites should not be invasive enough to noticeably inhibit the current ability of System Administrators or whomever administers the service to perform the majority of their duties, as they are merely a compartmentalisation of a select few roles. Note that even though the PNS is expected to be an integral part of KNEIP and only accessible through use of services within KNEIP, the limitation of access to the PNS algorithms through role compartmentalisation could by itself potentially eliminate the danger of access to it regardless of the source of the connection.

Note also the mention of a Service Administrator credential vault. To clarify, currently, a list of the credentials of all Administrators is stored in an encrypted key vault which is maintained and managed by the System Administrators.

For the sake of this thesis, I will assume these prerequisites are met and narrow the scope to include only the KNEIP system. Within this scope I can finally begin to decide on and implement a solution which achieves the desired goal.

6.5 – A solution

As presented in Chapter 5, there are several existing avenues of approach when it comes to securing data and systems from System Administrators. Any one of them has been done before, and many businesses operate with their own set of approaches which cater to their respective needs. My contribution will be to combine custom versions of these to achieve the aforementioned goals in the framework available at KRG. Specifically provide an adequate level of security within the scope set forth previously, while still maintaining a limited impact on the daily tasks of System Administrators at KRG.

Looking first at the state of the system, within the scope, before any security is in place.

- The PSN can only be accessed through KNEIP.
- The access to the PSN is controlled by the KNEIP user management, which are linked to the AD authentication servers.
- KNEIP is a modular system, where services manage any action within it. Each service has specific roles, and no two services perform the same task. More details in the next chapter.

Moving on to the actual implementation, taking all the prerequisites and assumptions set forth thus far into consideration, I will need to achieve a few goals. Firstly I will need to ensure that there is no possible way for a System Administrator, within the scope, to tamper with the PSN algorithm. Secondly I will need to ensure that any such attempt is caught and logged, and this logging needs to be secure and separate from the default system log. Thirdly I will need to ensure that certain authorized access is truly authorized through the addition of an internal list of authorized users, and my implementation will have to cover both System and Database Administrators. Finally I will have to ensure that the security service itself is safe from tampering.

As a result of the above, I envision 4 distinct parts to my implementation:

- A service call intercept to monitor activity.
- A secondary ghost log, stored in a secure location.
- A multi person control model when certain actions are requested along with an internal authorization list.
- Maintaining the Integrity and Availability of my security module.

In addition to these four parts, an issue arises with the fact that authentication is controlled by the ADC, as a System Administrator could circumvent attempts to limit user creation in KNEIP by adding authorization directly to the ADC, which could then be propagated to KNEIP. To combat this an internal list of authorized personnel should be created and maintained by the implementation. It will mirror the actual list of users in the KNEIP system but in order to update them the KNEIP user creation services MUST be used as well as the list be checked whenever the multi person control model is called so that dummy accounts will not be allowed to function as substitutes. This helps ensure the scope is limited to KNEIP without fear of external circumvention.

As mentioned in passing earlier, the ghost log and internal authentication list database must be protected from tampering. It is possible within the Sybase ASE databases to create Database Administrator accounts that can manage the database as a whole without being allowed to edit or even view the data on any given table. Doing this, and then keeping the credentials of the DB creator account confidential, should minimize the possibility of any tampering by the Database Administrators. This must be taken into account when the database is to be created.

There are several more things to take into consideration when going forward with this solution, and the will be discussed in the next chapter where the actual implementation of each part and the application as a whole will be presented in depth.

6.6 – Summary

In this chapter I first presented an example of a completely secure system. It also happens that in order to achieve this it also becomes a completely unmanageable system bordering on useless.

Following a summary of the absolute powers of a System Administrator and the relationship between KNEIP and the PNS a list of assumptions and prerequisites were presented which are either expected to come to pass or are otherwise required for the implementation to achieve the desired goal. These are obviously assumed in the next chapter where the solution will be implemented.

Finally a solution within the given confines was presented, a combination of method intercepts, secure logging and a multi person control model. Given the assumptions and prerequisites it is possible to have control over the ability of System Administrators to access personally identifiable data at KRG. This is the template used for presenting the implementation in Chapter 7.

Chapter 7 – Implementing the Solution

The goal is to develop and implement a service for the KRG KNEIP system which can catch, log and possibly prevent abusive access to the database which contains personally identifiable data, by elevated users. I will take a snapshot of the KNEIP system currently in development, containing only the services which will be relevant as well as making several assumptions on services which will be made in the future, and make a development environment in which to implement and test my solution. In order to achieve this the service needs to include the four parts mentioned in the previous chapter.

First is the intercept several services which interact with the PNS service or commands sent to certain services which are not part of normal operations, such as editing or deleting logs and attempting to tamper with the security service. However, KNEIP is still in development and much of the final functionality is yet to be implemented. As a result command intercepts will be tested by creating several assumed dummy services which represent the functionality I wish to monitor and running several attempts to access these.

Second is to log such unusual attempts in an additional 'ghost' log which cannot be edited or deleted, and can only be viewed by certain personnel. This operation will be entirely separate from the logging in place in the system, and will store the logs in a database with restricted access to ensure Confidentiality and Integrity.

Third is to implement a multi person control model. This means requesting authentication by a second Database Administrator or other authorized person for some select few actions. This will require the most work as it goes beyond the intended functionality of Spring security. It will require the creation and maintenance of an authentication queue for any required authentications and some way to access it and authorize requests.

Fourth, and last, is to ensure that my own service cannot be tampered with and the logging of any attempt to do so.

While KNEIP implements normal authentication, a user access table and logging, my implementation goes beyond that and specifically targets elevated user abuse in regards to a specific set of data. This is not functionality that is available through either the Spring framework or Java as a whole, and will certainly be a new extension not previously extant at KRG.

7.1 – Assumed Services

Due to KNEIP being in-development, parts of it are very bare and incomplete, and much functionality is not yet in place. As a result I will assume the existence of several services, also known as methods or classes, which the implementation will require. These assumptions are made from consultation with the development team and mostly reflect the current vision for the final product. Due to the modular nature of KNEIP it is trivial to alter, add more of or remove these at a later date, so these assumed methods will be sufficient to prove the viability of the module. I will list relevant method calls and classes, describe their intended functionality as well as including any arguments or return values.

- In KNEIP Activity Logs:
    - Void delete(userName, timestamp)
      Assumed method which allows for the deletion of log file entries. Not a common feature but it will enable the testing of a basic intercept and the logging of this event along with backing up the deleted log entries without hindering the action.

- In KNEIP User Control
    - KneipUser fetchUser(methodID)
      A method which returns the user which called the intercepted method. This is necessary to add data to the secondary log as well as ensure any four eyes authentication knows to not allow authentication from the same user twice.

    - Void createUser(id, firstName, userName, comments, roles, ownedLocks, projectGroup)
      Assumed method which simulates the creation of new users in the system.

    - Void setRoles(userName, roles)
      Assumed method which simulates the setting of new roles in existing users. Both of the above are useful for testing analysis of argument content. If the new user or existing user

is given a role of administrator my module will request a secondary authentication by a separate authorized user.

- In System Manager
    - Void stopService(id)
      Assumed method which handles the stopping of system resources, in case of maintenance or upgrades which require such. If id matches the id of certain modules then it must be denied as it could be an attempt at circumventing the security measures maintained by it.

- In Error Manager
    - genericError
      A class which represents future error objects. These are returned if the attempted method call is rejected and the system handles them further to display to the user. Contains a constructor which sets the message, and a method getMessage() which returns the relevant error message. The message is simply a placeholder for future 'error type' functionality.

- In User Authenticator
    - Boolean AuthAttempt(token)
      A method which emulates a future authentication framework. This will be used in testing the four eyes authentication.

- In ID Control
    - Long createID()
      A method which creates new process ID's for new services initiated in the system.

7.2 – Services and resources to be implemented

While a solution and its parts have been decided on, the implementation of these will require the creation of several new services which will make up the structure. The package that they will all be part of is called admCtrl, an obvious shorthand. They mostly follow the parts envisioned, and are as follows.

- admCtrl
  The ‘main’ class, which initiates the remaining services, calling the system ID creator and passing relevant arguments to each.

- secAspect

  The Aspect class which contains all the service intercepts, comprised of pointcut definitions and advice methods. Further described in 7.3.

- ghostLog
  The ghost log logger. Contains methods for registering logs and searching the log database.

- fourEyes
  The multi person control authenticator module. Intended to control authentication of a secondary authorized user.

- secAuth

  Internal authenticator which manages the admCtrl authentication list. Secondary authentication checks separate from AD.

In addition a ghost log/internal authentication database will need to be created and deployed for the implementation to be complete. Each ‘part’ and their relevant services will be covered in more detail in the following subchapters.

7.3 – Intercepting service access

Since KNEIP uses a highly modular design, where every major factor is controlled by a separate service it is possible to target specific methods in these services without extensive rewrites of lower level code, though this introduces many cross-cutting concerns. Spring supports Aspect Oriented

Programming (AOP), which is often used for logging, enabling easy interception of method calls. AOP allows for a separation of cross-cutting concerns, and works by writing aspect classes which specify a pointcut on which to perform an advice which allows injection of code before, during and after any method call as well as determining whether the method is allowed to run at all. A pointcut is simply a predicate that matches a joinpoint, where a joinpoint is the term describing a point during the execution of a program, such as a method call or the handling of an exception. An advice is a specification on what is to be done at any given pontcut, such as running code before execution, during execution, after execution or even preventing execution and returning an error.

Spring AOP is simpler to use than the de-facto AOP extension for Java, namely AspectJ, however it is somewhat limited only allowing 'before' advice on method-execution pointcuts. This is not enough for this project, as it requires the ability to deny the execution of a method as well as extract the method ID to fetch the calling user. In light of this I will be using the full AspectJ extension.

There is also a decision to be made in respect to when the weaving will occur. Weaving is the terminology given to the act of injecting code around method calls which AOP in essence does. Default is compile time weaving, which injects the code into the system at compile time and will then be an integral part of it. Another method is load time weaving, where it is woven once the system is booted up. The final is runtime weaving, where the code is dynamically woven when required.

Both load time and runtime weaving are useful in some cases, but since this is a security application which should ideally not be disabled, I have decided to implement it with compile time weaving. This requires the use of the AspectJ compiler when compiling the system, but this is a trivial matter.

Enabling full AspectJ support requires the addition of a line to the Maven dependency file, along with a list of bean definitions which AspectJ needs to see the classes it needs to interact with. Part of the Maven config file is shown in Figure 8. Beans defined in Maven are created as objects on system startup and can then be manipulated and tested on, removing the need to create the objects in the test file as you would with a simple test framework. Relevant constructor arguments are passed here, and the secure logging database connection is also defined. I have censored the location, username and passwords for the test database for security reasons.

```
<?xml version="1.0" encoding="UTF-8"?>
.
*Some schema definitions*
.
.
.
  <aop:aspectj-autoproxy/>

  <bean id="ghostLog" class="net.krg.kneip.common.domain.security.admCtrl.ghostLog">
    <property name="dataSource" ref="dataSource"/>
  </bean>

  <bean id="fourEyes" class="net.krg.kneip.common.domain.security.admCtrl.fourEyes"/>

  <bean id="acLog" class="net.krg.kneip.common.domain.common.log.ActivityLog"/>

  <bean id="kUserCtrl" class="net.krg.kneip.common.domain.security.kneipUser.KneipUserCtrl"/>

  <bean id="sysMgr" class="net.krg.kneip.common.domain.system.SystemManager"/>

  <bean id="admCtrl" class="net.krg.kneip.common.domain.security.admCtrl.admCtrl">
    <constructor-arg type="long" value="0000"/>
    <constructor-arg ref="kUserCtrl"/>
    <constructor-arg ref="acLog"/>
    <constructor-arg ref="ghostLog"/>
    <constructor-arg ref="fourEyes"/>
    <constructor-arg ref="secAspect"/>
  </bean>

  <bean id="secAspect" class="net.krg.kneip.common.domain.security.admCtrl.secAspect">
    <constructor-arg ref="ghostLog"/>
    <constructor-arg ref="fourEyes"/>
    <constructor-arg ref="kUserCtrl"/>
    <constructor-arg ref="acLog"/>
    <constructor-arg ref="sysMgr"/>
  </bean>

  <bean id="dataSource" class="org.springframework.jdbc.datasource.DriverManagerDataSource">
    <property name="driverClassName" value="com.sybase.jdbc4.jdbc.SybDriver"/>
    <property name="url" value="jdbc:sybase:Tds:***DBIP/DBloc***"/>
    <property name="username" value="******"/>
    <property name="password" value="******"/>
  </bean>
  .
  *some remoting config*
  .
  .
  .
</beans>
```

Figure 7: Maven configuration file

Enabling AspectJ requires additional compiling with the AspectJ compiler, but there is less overhead than with Spring AOP which will be beneficial regardless of weave time.

Additionally there is the choice of implementing the AOP either as regular Java code with annotations or in xml form. Either way is as good as the other and the choice depends more on preference than any other factor. For the sake of readability, and my own familiarity, the implementation of the intercept will be done in Java.

In order to perform any form of advice, I will first have to declare pointcuts to determine joinpoints of interest. Pointcuts are declared with the @ annotation, followed by a specification of the type of pointcut, such as ‘execution’ which defines that the pointcut will be matched to the execution of a method, and then the path to the joinpoint. This definition is followed by the name of the pointcut which can later be used to perform advice on.

Advice is also declared with the @ annotation, which points to the desired pointcut. In this implementation the ‘around’ advice has been chosen as it allows for complete control over the called method, including whether it can run at all. This is then followed by a method which contains the desired code to run before, during and after the joinpoint such as logging and further authentication. Each method must return an object, as allowing the execution of the joinpoint requires the returning of a ProceedingJoinPoint object which points to the original method call or in the case of a failure or denial, the return of an Error object that the KNEIP error management will catch.

The required pointcuts can be found in the excerpt of the aspect class in Figure 9.

```
@Aspect
public class secAspect implements DomainEntity {
  .
  .
  .

  //Pointcuts

  @Pointcut("execution(* " +
              "net.krg.kneip.common.domain.common.log.ActivityLog.deleteLog(..))"+
                 " &&args(userName, timestamp))")
    private void logDelete(String userName, Date timestamp) {}

  @Pointcut("execution(* " +
              "net.krg.kneip.common.domain.security.kneipUser.KneipUserCtrl.createUser(..))"+
                 " &&args(id, firstName, userName, comments, roles, locks, projectGroup))")
    private void userCreation(Long id, String firstName, String userName, List<Comment>
                              comments, List<KneipRole> roles, List<KneipLock> locks,
                              ProjectGroup projectGroup) {}

  @Pointcut("execution(* " +
               "net.krg.kneip.common.domain.security.kneipUser.KneipUserCtrl.setRoles(..))" +
               " &&args(userName, roles))")
    private void userRoleSet(String userName, List<KneipRole> roles) {}

  @Pointcut("execution(* " +
               "net.krg.kneip.common.domain.system.SystemManager.stopService(..))" +
                  " &&args(ident))")
    private void serviceStop(Long ident) {}

  .
  .
  .
}
```

Figure 8: Service Intercept Pointcut Declarations

As seen in above, the number of pointcuts required are few, which is due to the demands of the prerequisites that allowed for a narrowing of the scope. Going through the pointcuts, what they intercept and what the advice methods for each will do:

- The deleteLog pointcut is, as it states, the call to a method in the ActivityLog service which allows for log deletion. More an exercise in theory than likely implementation, but a method which may exist nonetheless. Even if the method exists, it is not desirable that logs disappear and thus it must be intercepted.

    - The advice method will log the attempted occurrence by creating a deletion log entry in the ghost log database. It will contain a timestamp, the name of the user that called for the deletion and the contents of the original log which are; the timestamp of the original log entry, the user name of the user the original log entry concerns and the log message of the original log.

- The createUser and setRoles pointcuts concern themselves with methods in KneipUserCtrl which relate to the creation of new elevated users in the KNEIP system. While authentication occurs through the ADC, the system itself has its own user list. These need to be intercepted and the requested action analysed to decide whether further authentication will be required. These methods will need to be called to allow users to access the PNS, as will be further detailed in 7.5.
    - In both of these cases the list of roles will be looped over, and if they specify a System Administrator role, a Database Administrator role or a Developer role, the method will call for further authentication by invoking fourEyes, checking the internal user list for matches, as well as logging the occurrence in the ghost log. If authentication fails the original method call will be denied and an error object will be returned, if authentication succeeds it will be allowed to proceed.
    - Additionally if the creation of users with, or modification for roles that allow access to the decoding method of the PSN the method will intercept and log the occurrence and ask for a four eyes authentication.

- The stopService pointcuts are again quite self-explanatory. They must be intercepted and analysed, such that any attempts at stopping the services presented in 7.2 be denied. Attempts to stop the PNS must be authorized, as further detailed in 7.6.
    - If the service ID of the service to be stopped matches the ID of any of the services in this implementation, it will be outright denied. If it matches the ID of the PSN, it will request further authentication and upon success be allowed to proceed. The occurrence will of course also be logged.

An example of advice to be performed on the pointcuts, specifically the advice in the case of log deletion, can be seen in Figure 10.

```
@Aspect
public class secAspect implements DomainEntity {
   .
   .
   .
   //Advice to run on pointcuts
   @Around("logDelete(userName, timeStamp)")
          public Object logRemoval(ProceedingJoinPoint pjp, String userName,
                                              Date timeStamp) throws Throwable {
      System.out.println("Log Delete intercepted."); //Testing

      ActivityLogService tmp = (ActivityLogService)pjp.getThis();//Ref  to calling object instance

      try {
         String log = tmp.getLogEntry(userName, timeStamp);
         String user = kc.fetchUser(sm.getUserId(tmp)).getUserName();
         gl.addDelLog(user, userName, timeStamp, log, "Testing"); //Create ghost log entry
      } catch (Exception e) { //Will happen if required services or ghost log DB are down.
         System.out.println("Exception caught: " + e.toString()); //Testing
         return new genericError(e); //Handled by KNEIP error handler
      }
      return pjp.proceed(); //Method call allowed to proceed as normal
   }
   .
   .
   .
}
```

Figure 9: Advice example, logDelete

The logDelete advice fetches the reference to the calling object, then tries to fetch the username of the user which called this object for logging, before calling the ghost log service. Currently the catch handles generic errors, though this can easily expanded to include more specific errors as needed. The return is simply the reference to the called object, and the proceed() call allows it to do just that.

7.4 – Secondary secure logging

Secure logging is necessary to avoid tampering by a rogue administrator who wishes to cover their tracks. It is essential that this logging not be accessible through normal means, and as presented in subchapter 6.5, a useful method is to store them in secure databases. As mentioned in Chapter 4, the databases at KRG are implemented with Sybase ASE and use a standard SQL syntax. Given the prerequisites presented the implementation is relatively trivial as it requires a simple SQL query which enters a log entry into the relevant table. The database will be divided into several different tables, one for each log type in order to keep searches simple and organized.

The location of the database can be managed in two ways. Either stored in a flat configuration file which the program reads, or hardcoded in. A flat file makes altering the location of the database easier if ever needed, but it opens for tampering in the form of setting up a dummy server and simply changing the contents of the flat file, thereby compromising the integrity of the logs. The location will therefore be hardcoded into the configuration file. It is unlikely that the database will ever change location as KRG has complete control over its internal IP tables, making this concern an acceptable risk. In the event of the database crashing, being disabled forcefully or changing location unannounced the application will return false, causing the rejection of the attempted method call. This is rather harsh at first glance, however this implementation will rarely be invoked due to its limited scope, and as thus the impact will be minor.

Finally it would be important to have security updates of the database periodically. There is already a system for backing up databases to several remote locations at KRG, and adding this database to that backup schedule would ensure proper backups and prevent the physical removal of the entire database by a particularly determined interloper.

Figure 10 shows an excerpt of the ghost log class, with local variables and the log deletion logger.

```
@Entity
public class ghostLog implements DomainEntity {

   //Local variables
   private Long ghostID;

   private String delTable = "AALogDelete"; //Table for storing logs on system log deletions
   private String userEditTable = "AAUserEdit"; //Table for storing logs on user creation and edits
   private String serviceTable = "AAStopService"; //Table for logs on service termination attempts

   private JdbcTemplate jdbcTemplate;
   .
   .
   .
   //Log entry in case of system log deletion
   public void addDelLog(String user, String origUser, Date origTimeStamp,
                   String origLog, String note) throws Exception {
      System.out.println("Adding Deletion Log."); //Testing
      if(user.length() > 8) {
         note += "| User name SQL injection attempted. name was: " + user;
         user = "REPLACED";
      }
      try {
         Date date = new Date();
         //Then write out the sql statement
         String sql = "insert into " + delTable + " (Timestamp, UserName, OrigTimStamp, " +
               "OrigUserName, OrigMessage, Note) values (?, ?, ?, ?, ?, ?)";
         jdbcTemplate.update(sql, date.toString(), user, origTimeStamp,
               origUser, origLog, "Testing"); //Perform statement
      } catch (Exception e) {
         throw e; //Throw exception for secAspect to catch further
      }
   }
   .
   .
   .
}
```

Figure 10: Ghost Log class variables and log entry example

7.5 – Multi person authentication

As decided, for some of the intercepted actions further authentication is necessary so a system for handling this must be made. When such an intercept occurs, the fourEyes service will be called and it will hold the method call until such a time that it determines the secondary authentication is valid.

In order to gather such a secondary authentication the implementation would make use of the already existing helpdesk system available to administrators. The service maintains a ‘queue’ by routing a message to the helpdesk system where the message will lay in a dedicated inbox until answered. It is possible to add a time limit for answering these requests, but for this iteration that will not be considered.

The helpdesk software allows for secondary login prompts, and the message would initiate one such and return an encrypted credential token to fourEyes. The service will ensure the credentials do not match the credentials of the method caller, then send the credential token to KNEIP’s authentication module which validates the login with AD. If it fails, a new message will be sent to the queue once more to await another attempt. It is possible to have an arbitrary amount of retries, but in line with other accepted security protocols there will be the standard 3 tries. If it passes the method call will be released and the requested operation will be completed while the calling user will be notified of the success.

In addition it will be necessary to check that the authentication token returned from the helpdesk does not belong to the Administrator who initiated the original method call to begin with. Two separate users need to be part of the multi person authentication naturally enough.

For the PSN service no action will be needed. Modification is only possible before compile time, meaning the service would have to be stopped to modify it. If this were to change to allow on the fly modification, an intercept and four eyes control of this action would have to be implemented.

Finally, the internal authentication list must be checked whenever fourEyes is called. It will be stored on the same database as the secure logging, and will be a black box table which Database Administrators do not have the ability to view or edit. Since four eyes is required for super user creation or elevation this very list will ensure that any new authorized users added to the list will not be done by dummy users created in the ADC.

If the authentication fails, the method call will be denied and an error will be returned for the KNEIP error manager to handle further.

An example of an advice which calls fourEyes can be seen in Figure 11, in this case an intercept of an attempt to stop a service. The input is evaluated, and if an attempt to stop the PNS is attempted, it requires further authentication and fourEyes is called. The return value of the requestAuth method is an int for the sake of logging, and will always attempt at most 3 times.

```
@Around("serviceStop(ident)")
        public Object selfPreservation(ProceedingJoinPoint pjp, Long ident) throws Throwable {
    System.out.println("Service Stop intercepted."); //Testing

    SystemManagerService tmp = (SystemManagerService)pjp.getThis();//Ref to calling object
instance
    Long PSNid = 5L; //Since PSN has not been implemented, dummy id
    int authAtt = 0;

    try {
      String user = kc.fetchUser(sm.getUserId(tmp)).getUserName();

      if(ident.equals(id) || ident.equals(ac.getId()) || ident.equals(fy.getId())
         || ident.equals(gl.getId())) {
        gl.addServiceStopLog(user, tmp.getServiceName(ident), false, 0,
                              "Not allowed to stop Security Services");
        return new genericError("Can't let you do that, Dave.");
      } else if(ident.equals(PSNid)) {
        authAtt = fy.requestAuth(user);
        if(authAtt > 2) { //Failed authentication
          gl.addServiceStopLog(user, "PSN", false, authAtt, "Authentication failed");
          return new genericError("Seems you failed.");
        } else {
          gl.addServiceStopLog(user, "PSN", true, authAtt, "");
          return pjp.proceed();
        }
      } else {
        gl.addServiceStopLog(user, "Other", true, authAtt, "");
        return pjp.proceed();
      }
    } catch (Exception e) {
      System.out.println("Exception caught: " + e.toString()); //Testing
      return new genericError(e); //Handled by KNEIP error handler
    }
}
```

Figure 11: Advice which evaluates input and calls fourEyes

Unfortunately the implementation of this service in its entirety is currently not feasible as the new helpdesk system was just recently deployed and there was not sufficient documentation available on how to make use of the potential login prompt functionality in time for the completion of this thesis. Despite this a dummy version of the fourEyes method will be created for testing purposes and the completion of this module will be discussed in subchapter 8.2 on future works.

7.6 – Securing the security

An important part of any security application is to ensure that the service itself is not tampered with. If the module, or parts of it, go down then any security it attempted to implement would be rendered quite useless, limited or even cause a nonfunctioning system.

As mentioned in subchapter 7.1, the decision was made to implement compile time weaving. This means that the module can never be simply 'disabled', as the code is woven into the system during compile time and will be there permanently. Only if the services that contain the joinpoints are disabled will the code be rendered inoperable, but then the services will be unavailable so any attempt at using them will automatically fail.

The services can go down by mundane means, but in an attempt to immunize it from deliberate tampering the implementation will block any attempts at overtly disabling the services. As previously presented, the intercept catches any attempt at disabling any service, and if the service ID matches the ID of the ghost logger or the four eyes authenticator, it will deny the operation and return an error.

Despite this however, if some of the services the woven code relies on go down anyway then the system will fail if the calls are not handled appropriately. It might cause instability or crashes, and that is not acceptable in a system which requires reliable uptime. The solution is to always use a try and catch for every call. This can be seen in Figure 10 where the logger is called. If the try fails, then the catch will perform its code. The result of such a failure will be the denial of the attempted method call and an error being returned. The use of this requires that every method throws Exeption so that if any internal failure occurs, then it can return an error message for the catch to process. There can be several catches for each try, covering several different forms of failure, though this implementation only implements a generic error.

The PSN itself must also be protected from tampering. Even though there is a premise that dedicated personnel manage the algorithms and keys implemented in the PSN, it could still conceivably be possible to disable the service and replace it with another service which sends data to third parties without the normal security catching this. To combat this the ID of the PSN service will be made available to this implementation at system startup so that any attempt at stopping this could be intercepted. Unlike with stopping the security services, this attempt will not simply be stopped. This gives slightly greater flexibility instead of demanding the recompilation of the entire system in order to alter it. In the event of a future policy change requiring the modification of this algorithm it will instead require secondary authentication through the use of four eyes. One of the users in this authentication MUST be the PSN Administrator, and the implementation will ensure this.

Additionally, since the ghost log is a database an opening for injection attacks becomes possible, and it is important to not let any user input commands be allowed directly into the database as an unchecked passed variable. With this implementation, the only variable that can potentially be tampered with is the username of the caller. A rogue admin could conceivably create a user with name '; TRUNCATE TABLE tablename'. This could be a problem if the implementation simply added a string together to send unsanitized to the database, but using the jdbcTemplate.update command the database will check each incoming field data to ensure the format is correct, avoiding SQL injection vulnerabilities. It would be useful to notify administrators of some of the more blatant attacks such as the attempt to delete the log as shown above. As luck would have it, all usernames at KRG are short to the point that they are all far below the number of characters this requires. Even if the table name had only one character, the minimum length of the command would have to be 18 characters, and old ADC naming limitations prevented any usernames above 15 characters in length. As a result, short usernames became policy, usually 2 to 4 letters from a combination of first and last name initials and this practice has continued to this day. Now, modern versions of AD allow much longer usernames, but due to the naming conventions in place a simple check for username length can be implemented and if above, say, 8 characters a message would be instantly sent to some administrator in addition to it being logged normally.

7.7 – Summary

After presenting a list of assumed services for the unfinished KNEIP system, this chapter proceeded to describe the implementation of the three modules and overarching internal security in a structured manner. AOP was used to implement a service intercept system which catches the attempted triggering of certain actions and performs various analysis which is logged. In some of the cases, such as admin user creation within the KNEIP system, a four eyes authentication is called and forces additional authentication. With the exception of the Four Eyes module, all desired functionality was implemented successfully and will be tested in the next chapter.

Part IV

Testing and Results

Chapter 8 – The testing

In order to evaluate the functionality and feasibility of the implementation, some testing is necessary. As there are four primary parts to the implementation, the testing will also take a form corresponding to each respective part.

The framework used in the testing is Junit, which is a Java testing framework which is also build into Spring. It allows for a range of tests which cover both successes and failures with varying input parameters and assertions as well as simple method calls and printouts. It allows for the creation of a dedicated test class, placed in a separate 'test' structure in the project which can be called by the IDE at any time as defined in the maven configuration to initiate the system and run the included tests.

Policy requirements and prerequisites can of course not be tested in this thesis, and will as such be assumed to function as described. Instead the testing will focus on the functionality of the implementation as described in the previous chapter.

Figure 12 shows the setup of the test class which fetches references to the services of interest from the beans definitions and sets up several constants for use during testing. The @Before annotation defines the method which runs before any others, and is useful for preparing references and variables.

```
@RunWith(SpringJUnit4ClassRunner.class)
@ContextConfiguration(value = "classpath:META-INF/spring/aspectConfig.xml")
public class tester{
   .
   .
   .
   @Before
   public void setUp() {
      al = (ActivityLogService) ac.getBean("acLog");
      actrl = (admCtrlService) ac.getBean("admCtrl");
      gl = (ghostLog) ac.getBean("ghostLog");
      fy = (fourEyes) ac.getBean("fourEyes");
      sa = (secAspect) ac.getBean("secAspect");
      kc = (KneipUserCtrlService) ac.getBean("kUserCtrl");
      sm = (SystemManagerService) ac.getBean("sysMgr");
      sm.setId(9L); //Force id for testing
      ic = new IDControl();
      tmpDate = new Date();
      adm1 = new KneipUser(); //An administrator
      adm2 = new KneipUser(); //Some other administrator
   .
   .
   .
}
```

Figure 12 - Testing set up and preparation

8.1 – Interceptor and logger test

The interceptor and logger test have been combined as calling even the simplest of the methods to be intercepted necessarily calls the logging mechanism. While they were tested separately to begin with, the finished implementation cannot do one without the other, as intended.

The testing of the interceptor is relatively simple. The tester simply calls the methods which correspond with the implemented joinpoints and check if they are in fact intercepted by the interceptor.

In order to test the logger a test database is created with the required fields and attempts to call the methods at each of the joinpoints are performed with varying forms of input to check that the correct data is stored on the database.

The test was run by ensuring the objects existed, then calling an attempted log deletion which is supposed to be intercepted and logged. Figure 13 shows the testing code for this, figure 14 shows the printout results and figure 15 shows the result in the database.

```
@Test
  public void testIntercept() throws Exception {
    //Tests AoP intercepts and ghost logging
    System.out.println("Starting test.");
    assertNotNull(sa);
    assertNotNull(gl);
    al.deleteLog("testUser", tmpDate);
    System.out.println("Test completed!");
    .
    .
    .
  }
```

Figure 13: Intercept and logging test code

```
Starting ghost log test.
Log Delete intercepted.
In fetchUser
Adding Deletion Log.
In deleteLog.
Ghost log test completed!
```

Figure 14: Intercept and logging test printout

Timestamp	UserName	OrigTimStamp	OrigUserName	OrigMessage	Note
Tue Apr 22 11:4	adm1	Apr 22 2014 11:	test	Some log entry1	Testing

Figure 15: Log Deletion attempt log database result

As the results show, the intercept and logging functionality works as intended. Once the call is attempted, the AOP code catches it, finds out the calling user, calls ghost log to log it and then allows the method to run. A note on the missing data in the timestamps in the database, this is due to a too short field for these values as they are currently mere char fields and not timestamp fields in order to avoid a compatibility issue during testing. This would be corrected in any implementation for deployment.

8.2 – Four Eyes test

Due to this section not being implemented fully, some aspects cannot be tested at this time. However the concept can be tested through sending a few authentication attempts through a dummy service that emulates successes and failures depending on the input. To achieve this I edited the fetchUser dummy method such that I could pre-set which user it would return, and depending on which user I set the fourEyes test method will return either failure or successes in emulation of an authentication attempt. This will then in essence be more a test of the analysis of the data in the intercept method to ensure that fourEyes is called when and only when it's supposed to be called. With this data the concept can be demonstrated even with missing functionality.

The first few tests will be on an attempted service Stops, beginning with an attempt at stopping one of the security services. Since the KNEIP ID generation is not in place, I have predetermined the ID's of each of the security services. admCtrl being 0L(Long), ghostLog being 1L, fourEyes being 2L and secAspect being 3L. Figure 16 shows the test code where I set the user to be an administrator and 'he' is attempting to stop the ghost logger. Figure 17 shows the resulting printout, and Figure 18 shows the resulting log in the Service Stop Log database.

```
@Test
public void testIntercept() throws Exception {
    .
    .
    .
    //Tests foureyes intercept for service Deletion
    System.out.println("Starting service deletion test.");
    assertNotNull(fy);
    adm1.setUserName("adm1");
    kc.setTestUser(adm1);
    System.out.println("Setup done.");
    sm.stopService(1L); //This is the ID of the ghost Logger, set for testing purposes
    System.out.println("Service deletion test complete.");
    .
    .
    .
}
```

Figure 16: Security Service Stop test code

```
Starting service deletion test.
Setup done.
Service Stop intercepted.
In fetchUser
Adding Service Stop Log.
Service deletion test complete.
```

Figure 17: Security Service Stop test printout

TimeStamp	UserName	TargetService	Allowed	AuthAttempts	RejectionReason	Note
Fri Apr 25 13:2	adm1	ghostLog	FALSE	0	Not allowed to stop Security Services	Testing

Figure 18: Security Service Stop test log database result

As the test results show, the intercept works exactly as intended. In this case fourEyes did not need to be called as the attempt was immediately rejected. Moving on to an example which requires further authentication, I set the second admin user, which will trigger the simulated event of a failed authentication attempt in the dummy fourEyes module. I will forgo printing the test code, so figure 19 shows the resulting printout and Figure 20 shows the results in the logging database.

```
Starting service deletion test 2.
Setup done.
Service Stop intercepted.
In fetchUser
Requesting secondary authentication
Adding Service Stop Log.
Service deletion test 2 complete.
```

Figure 19: Failed PSN Service Stop test printout

TimeStamp	UserName	TargetService	Allowed	AuthAttempts	RejectionReason	Note
Fri Apr 25 14:0	adm2	PSN	FALSE	3	Authentication failed	Testing

Figure 20: Failed PSN Service Stop log database result

Again the tests return the expected results. The intercepts accurately analyze the input and call the fourEyes module. Since adm2 was the caller the dummy module emulated 3 attempted authentications, which is the max allowed and then returned causing the intercept module to log the attempt and return an error instead of allowing the execution to proceed, as evident of the lack of a printout showing ‘In stopService’.

More tests were run on other permutations and each returned expected results, but it is unnecessary to simply list several pages of similar printouts. The above suffice to demonstrate the method and exemplify the results.

8.3 – Self security tests

Some of this was touched on in the previous section, such as emulated authentication attempts due to invalid authenticating users in the fourEyes service. Authentication errors will not be tested further as these are all to be handled either by KNEIPS existing authentication system or the fourEyes module. Regular users would never be allowed to call the stopService method by KNEIP, while fourEyes would ensure at least one of the users attempting a stop of the PSN service was a PSN administrator and followed the internal user list.

First in this test I will disable one of the services required by this implementation, to simulate some error or attempt at tampering that somehow worked. This will be emulated by injecting a null pointer into the implementation and seeing what happens. Figure 21 shows the printout when the ghostLog service cannot be found.

```
Starting service deletion test 2.
Setup done.
Service Stop intercepted.
In fetchUser
Exception caught: java.lang.NullPointerException
Service deletion test 2 complete.
```

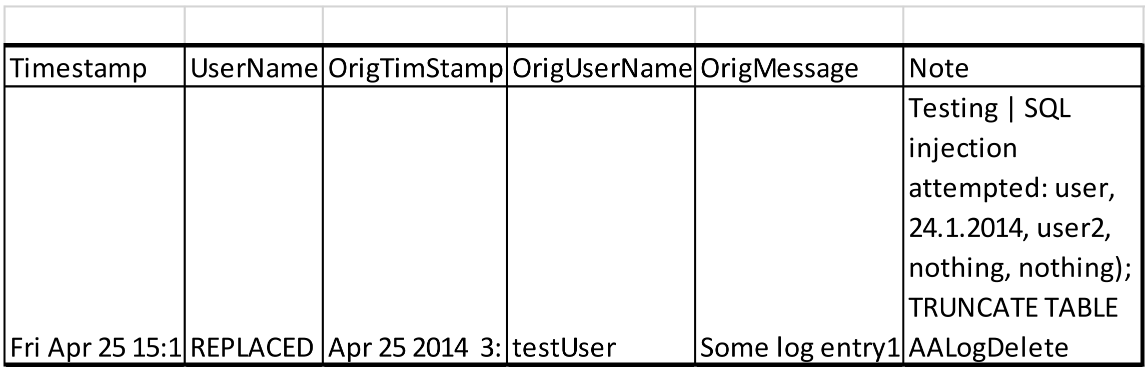
Figure 21: Attempt at calling disabled security service.

The final test from the fourEyes testing was used as an example, and here a simple null pointer exception was caught, which then resulted in the called method not being allowed to run and the error being percolated up the system to be caught by the KNEIP error handler.

Next up I attempted an SQL injection. Since the logger sanitizes the input through the use of a prepared statement this is not a real fear due to the use of prepared statements as well as the 'insert into' statement not being very prone to abuse. However as mentioned in subchapter 7.6 it would be useful to specifically take note of blatant attempts even though they are logged. So if an admin who knew the structure of the log database created a username '*user, 24.1.2014, user2, nothing, nothing); TRUNCATE TABLE AALogDelete;*' which would wipe the database if it was not sanitized it is far above the set limit. Figure 22 shows the printout and Figure 23 the result in the database.

```
Starting SQL injection test.
Setup done.
Log Delete intercepted.
In fetchUser
Adding Deletion Log.
Blatant, lengthy SQL injection noticed
In deleteLog.
SQL injection test complete.
```

Figure 22: Blatant SQL injection attempt test



Timestamp	UserName	OrigTimStamp	OrigUserName	OrigMessage	Note
Fri Apr 25 15:1	REPLACED	Apr 25 2014 3:	testUser	Some log entry1	Testing | SQL injection attempted: user, 24.1.2014, user2, nothing, nothing); TRUNCATE TABLE AALogDelete

Figure 23: Blatant SQL injection attempt log entry

As evident from the printout, the attempt was caught and the log reflects this. In a future version the catch would send a message to the helpdesk system for the administrators to see, so that it could be further investigated, though that functionality is not in place for the same reasons that fourEyes communication with the helpdesk is not.

With these two bases covered the security module should be secure from tampering within the given scope.

Part V

Conclusion

Chapter 9 – Conclusion and further work

9.1 – Conclusion

This thesis has shown that deciding on the correct balance of security vs usability is a complex problem with no universal solution. Any solution needs to be tailored to the specific organization and a practical solution to securing sensitive data against System Administrators needs to be implemented specifically. Therefore, the specific context of the KRG was presented in length, and background information had to be understood before narrowing the scope sufficiently. In the end, this thesis presents a feasible solution which achieves the intended goal.

The solution as presented, given the assumptions and requirements, will be able to prevent access to the PSN service by unauthorized System administrators. It adds a new facet where there previously was nothing, and due to its modular and focused nature is cheap to implement and simple to upgrade or expand in the future. With the combination of secure connections, demands for access to be granted within KNEIP so that an internal overview can be monitored, an intercept of services which could both directly and indirectly grant access, secure secondary logging and reporting and a multi-person authentication I believe that all relevant avenues have been covered. The implementation ensures that attempts from within KNEIP are caught and handled, while the prerequisites prevent external direct tampering.

The solution is of course not absolute, as the social aspect is still in play and if a rogue administrator gains the credentials of another administrator, then the system would not prevent access. It is regardless far superior than what is already extant, and the fact that it would be known that this specific form of abuse is logged separately and securely will also certainly help deter abuse attempts, potentially making rogue entities think twice.

The testing showed that the implemented solution worked as intended without any obvious problems. Every test returned a success and all outputs were as expected. Despite the incomplete fourEyes module I deem the implementation to be a success, and with the completion of said and some further testing could be ready for deployment.

Due to the self-contained nature of the modules and the precautions taken, knowledge gained from this thesis or insight into the source code should not compromise the implementation in any manner.

9.2 – Further Work

As mentioned the completion of the fourEyes module would be of immediate interest for future work. Figuring out how the helpdesk receives messages and handles the triggering of login screens would be necessary to complete the functionality the security implementation requires. If not the helpdesk then some other solution such as a self-made application would be feasible.

The implementation of a scoring system of how ‘important’ an action is could potentially prioritize the reporting of certain actions. This in addition to a counting mechanism could add a trigger for actively sending a message to the administrators if a certain action is performed often, or more than normal in a short period of time instead of simply logging it and waiting for someone to read the log. This was considered for the thesis, but not included due to time constraints.

This implementation is quite limited in scope, but with what has been created it can be expanded to monitor and control further services and actions for specific user groups in specific circumstances. The service could also potentially be ported to other systems to perform similar or different checks on various services.

One interesting avenue of future work could be to see if it is possible to implement a method of multi person control and ghost logging at a lower level such as weaving code directly into the ADC and intercepting authentication attempts at the system level. This would require a thorough understanding of the source code of any given ADC, root access to inject such a service and insight into how an ADC communicates with the rest of the systems it interacts with.

Another expansion of functionality would be to expand the error handling to include more specific actions in response to specific errors. As it is now it simply throws generic errors for the KNEIP error manager to catch, but how it processes these is unknown as of yet and it might be desirable to have more control over these actions in the case of a security application. In addition it would be beneficial to implement the notification of relevant users of specific failures. If a service goes down send a message to the developers or if the logging database goes down notify the database Administrators. A messaging system would have to be chosen, but it could be of great utility in specific circumstance.

Additionally it might be interesting to explore the possibility of creating and backing up log databases dynamically from within the running application as needed, circumventing the need for database Administrators to create the databases and preventing the leak of the database root user credentials.

Bibliography

Andress, J. (2011). *The Basics of Information Security: Understanding the Fundamentals of InfoSec in Theory and Practice*: Elsevier Science.

Baker, W., Gouldie, M., Hylender, C. D., Niemantsverdriet, J., Novak, C., Ostertag, D., . . . Service, M. a. w. o. t. U. S. S. (2010). Verizon 2010 Data Breach Investigations Report (R. team, Trans.) (pp. 66).

Ferguson, P. (1998). The Internet Protocol Journal. *What Is a VPN - Part 1, 1*.

Frangopoulos, E. D., Eloff, M. M., & Venter, L. M. (2008, 18-20 November 2008). *Social Aspects of Information Security.* Paper presented at the ISSA Regional Social Security Forum for Africa.

Greenwald, G., MacAskill, R., & Poitras, L. (2013). Edward Snowden: the whistleblower behind the NSA surveillance revelations, *The Guardian*. Retrieved from http://www.theguardian.com/world/2013/jun/09/edward-snowden-nsa-whistleblower-surveillance

Josey, A. (2011). *TOGAF Version 9.1 Enterprise Edition, An Introduction* (pp. 13).

Karger, P. A., & Schell, R. R. (1974). Multics Security Evaluation: Culnerability Analysis (E. S. D. (AFSC), Trans.) (pp. 156). L. G. HANSCOM AFB, MA 01730: Hanscom AFB.

Kozaczuk, W. (1984). *Enigma: how the German machine cipher was broken, and how it was read by the Allies in World War Two*: University Publications of America.

Marks, P. (2011, 27 December 2011). Dot-dash-diss: The gentleman hacker's 1903 lulz. *New Scientist*.

Marshall, D. (2011). Top 10 benefits of server virtualization. Retrieved from InfoWorld website: http://www.infoworld.com/d/virtualization/top-10-benefits-server-virtualization-177828

Messmer, E. (2010). Biggest insider threat? Sys admin gone rogue. Retrieved from Computerworld website: http://www.computerworld.co.nz/article/362247/biggest_insider_threat_sys_admin_gone_rogue/

Miami, U. o. (2006). confidentiality, integrity, availability (CIA). http://privacy.med.miami.edu/glossary/xd_confidentiality_integrity_availability.htm

Morris, R., & Thompson, K. (1979). Password security: a case history. *Commun. ACM, 22*(11), 594-597. doi: 10.1145/359168.359172

Normen, S. f. (2013). *Norm for Informasjonssikkerhet, Helse-, omsorgs- og sosialsektoren*. Oslo: Retrieved from http://helsedirektoratet.no/lover-regler/norm-for-informasjonssikkerhet/Documents/Norm%20for%20informasjonssikkerhet%20i%20helse-%20omsorgs-%20og%20sosialsektoren.pdf.

Northcutt, S., Zeltser, L., Winters, S., Kent, K., & Ritchey, R. W. (2005). *Inside Network Perimeter Security (2nd Edition) (Inside)*: Sams.

Oracle. Secure Coding Guidelines for the Java Programming Language, Version 4.0: Oracle Corporation.

Paquet, C. (2012). *Implementing Cisco IOS Network Security(IINS 640-554) Foundation Learning Guide* (2nd Edition ed.): Cisco Press.

Prabhakaran, V., Bairavasundaram, L. N., Agrawal, N., Gunawi, H. S., Arpaci-Dusseau, A. C., & Arpaci-Dusseau, R. H. (2005). IRON file systems. *SIGOPS Oper. Syst. Rev., 39*(5), 206-220. doi: 10.1145/1095809.1095830

Seacord, R. (2011). Top 10 Secure Coding Practices. Retrieved from Confluence website: https://www.securecoding.cert.org/confluence/display/seccode/Top+10+Secure+Coding+Practices

Thomas, T. M. (2004). *Network Security First-Step*: Cisco Press.

Various. Information Security. 2013, from http://en.wikipedia.org/wiki/Information_security