A model for a mobile operating environment

by

Jaco Louis du Toit

THESIS

Submitted in fulfilment of the requirements for the degree

PHILOSPHIÆ DOCTOR

in

Computer Science

in the

Faculty of Science

at the

University of Johannesburg

SUPERVISOR: DR. ID ELLEFSEN

OCTOBER 2018

A MODEL FOR A MOBILE OPERATING ENVIRONMENT

Acknowledgements

During the journey in developing the work of this thesis, several people played a very important role. I would like to take this opportunity to thank them all:

- My wife, Ilse. We had several obstacles that we faced during the career change that influenced my PhD. I am so happy that I could face these obstacles with you by my side. You are my pillar and I love you very much.
- My kids, Izelle and Marli. You may not read this, but maybe one day in the future you can laugh at all the things your dad envisioned.
- My supervisor Dr Ellefsen. Thank you for making time, even after you started your journey in industry, to still assist with this work.
- My mentor, Prof von Solms. I appreciate all the time that we can talk about academia, research and in general the passion for cybersecurity. Thank you also for showing me the path towards completion.
- The University of Johannesburg, for giving me the opportunity to accomplish something special in my life.

Abstract

In a connected world companies are facing many information and cyber security challenges. Users are using multiple computing devices for both business and personal use. The multiple devices create an environment where business and personal data becomes difficult to separate. The Neo model describes a mobile operating environment that supports a special mobile device, called the Neo device. The Neo device enables a user to use the device for both personal and business purposes. Personal and business information are strongly isolated with high levels of control for the data owner. The Neo model further describes a computing environment that allow users to use one device in many different scenarios using many different input and output peripherals, while maintaining a high level of security.

TABLE OF CONTENTS

Introduction

- Chapter 1 Introduction

The end game

- Chapter 2 Model Overview

Literature study

- Chapter 3 Isolation properties in mobile operating systems (SCP)
- Chapter 4 Isolation techniques in other technologies (SCP)
- Chapter 5 An overview of peripheral security (MAP)

Bridge

- Chapter 6 The drivers for a new model

Building the Neo model

- Chapter 7 Overview of the Neo Model
- Chapter 8 The Secure Container Property
- Chapter 9 The Mutual Authentication Property
- Chapter 10 Container life cycle
- Chapter 11 Novel Implemenations

Evaluation and summary

- Chapter 12 Potential implementation of Neo Model
- Chapter 13 Conclusion and future work

Chapter 1 INTRODUCTION

1.1 Introduction and Background

Today's computer users may have a number of computing devices. The computing devices range from office desktop or laptop computers to smartphone or tablet devices. The user uses computing devices for both business and personal purposes.

The reasons for the different computing devices vary, but one of the reasons may be that some devices are easier to use given a specific situation. Users that make use of public transport may find that a smartphone or tablet computer is easier to manage while travelling than a laptop computer. The same user may switch to a desktop computer when working at a desk at the office. The desktop computer allows the user to use the computer with an external screen, hardware keyboard and mouse.

It is inevitable that the use of different computing devices blurs the line between personal applications and data and business applications and data. When a user goes to a meeting and uses his tablet computer to make notes or present information, the tablet computer is used to produce or consume business information. When the same user uses his desktop computer, the data from the tablet computer needs to be transferred between the tablet computer and the desktop computer.

A common way to share data between computing devices is to make use of some type of cloud storage solution, such as iCloud (Apple Inc., 2014) or Google Drive (Google Inc., 2014) (Yang et al., 2013).

On 31 August 2014 pictures from a number of celebrities were stolen from the Apple iCloud service (Charles, 2014). Before this incident, DropBox (DropBox Inc., 2014) has had a number of security breaches where unauthorised people gained access to DropBox accounts (Ferdowsi, 2011).

According to a report in the Washington Post via information provided by Edward Snowden, the National Security Agency (NSA) collects information directly from United States service providers like Google, Microsoft and many more (Gellman & Poitras, 2013).

The above statements and many more instances such as these demonstrate that data stored in the cloud are vulnerable to a number of attacks. It is the author's personal experience that some companies have policies prohibiting users from transferring company data to cloud storage. The companies enforce these policies by blocking access to cloud storage providers at the firewall level.

To complicate the scenario many mobile devices are owned by users and not by the company. In recent years, the use of personal devices for business use created the Bring Your Own Device (BYOD) problem (Gartner, 2018). When a company allows an employee to use a personal device for business purposes, the company may require a certain level of control not only over the company data but also over the device itself.

Users, on the other hand, may be hesitant to allow companies full control over their devices, specifically if it infringes on their privacy. The isolation of not only company data, but also the isolation of the control aspect that companies require is important.

The question that can be asked at this time is whether it may be possible to minimise the dependency on cloud storage. One of the methods that may address the question is to minimise the number of computing devices to just one device. If a user only has one device, then there is no requirement to exchange data between devices. This solution, however, may not be optimal, since users like to use devices with specific Input\Output (I/O) peripherals for specific scenarios.

The advantage that mobile devices have over desktop or laptop computers is that they are easily carried and provides a smaller form factor. The form factor is defined as the physical size and dimensions of the hardware (Oxford University Press, 2018). Mobile devices, unfortunately, may have limited connectivity to I/O peripherals, which may cause users to use it in only certain cases.

Connecting different I/O peripherals to computing devices is not foreign when using computing devices such as desktop computers and laptop computers. Mobile devices, such as smartphones and tablet computers are rarely seen as replacements for desktop computing devices. Even though it is possible to connect different Input/Output (I/O) peripherals to mobile devices, the processing power, storage capacity and memory of mobile devices are, at the time of writing this thesis, limited.

The use of personal devices may also introduce further risks when users connect different types of input and output (I/O) peripherals to the computing device. Wired I/O peripherals have a certain level of risk exposure to devices such as hardware keyloggers or screen grabbers. Hardware keyloggers can capture input by users, exposing sensitive information such as passwords. Screen grabbers have the ability to capture screenshots that may expose sensitive data.

Users that have the ability to use company data and applications may inadvertently expose sensitive information when users use data on displays in an environment not authorised by the companies (Ali et al., 2014).

An exploratory study in 2017 showed that 48% of 174 respondents were observers during shoulder surfing incidents and 33% noticed that they were being observed during shoulder surfing incidents. The study further showed that shoulder surfing happened the most during commuting using public transport and that even friends, acquaintances and family all were observers in shoulder surfing incidents (Eiband et al., 2017). Another study in 2014 showed that only in 0.3% of the 3410 situations, did the users perceive the shoulder surfing incidents (Harbach et al., 2014).

The shoulder surfing situation can be exasperated if users are allowed to connect bigger screens to their computing devices while commuting, or if they inadvertently leak information to family while using their computing devices in environments not controlled by the company.

Another problem with I/O peripherals is that not all I/O peripherals manage security in the same way. In 2016 a number of non-Bluetooth wireless keyboards were identified to be vulnerable to a vulnerability called KeySniffer (Bastille Networks Internet Security, 2016). The KeySniffer vulnerability allowed attackers to capture keystrokes from a number of commercial wireless keyboards using equipment of less than a $100. The KeySniffer devices implemented inadequate controls that minimise the risks of man-in-the-middle attacks.

These cases emphasise that mutual authentication is imperative to ensure computing devices authenticate I/O peripherals, but also that I/O peripherals ensure that they are successfully authenticating the computing device. The secure transmission of data between the computing device and I/O peripheral is also important. Companies may even go further and express the importance of ensuring data transmissions between I/O peripherals and computing devices may not be exposed to applications that do not belong to companies.

The above discussion lead towards the specific problem that is addressed in this thesis.

1.2 Problem Statement

The present problem faced by users and companies is that the sharing of information between devices require the use of potentially untrusted cloud storage solutions. Since users may be using personal devices to access corporate data, businesses need to isolate and control not just the data, but also the programs and applications belonging to the company.

1.3 Hypothesis

It is hypothesised that it is possible to design a secure mobile operating environment that facilitates the use of a mobile device for both personal and business purposes, using hardware that is not readily available today but is possible in the future. The mobile device runs a purposely designed mobile operating system that ensures different form factor input/output (I/O) devices can connect to the special mobile device. The mobile operating environment ensures that the information on the device is classified and isolated. Access and secure communication are established on an application and data layer, instead of just the operating system level. Communication between the mobile device and I/O devices are mutually authenticated.

1.4 Objectives of thesis

The primary objective of this thesis is to develop a model that describes a secure mobile operating environment that can be used for both personal and business purposes. This secure mobile operating environment must allow users to use one system for both personal and business purposes, ensuring personal and business applications and data are protected from each other and are controlled by the data and application owner. To help fulfil the primary objective, a secondary objective has to be defined, in order for the primary objective to be met.

Primary Objective: Develop a model that is called the Neo model that describes a secure mobile operating environment that can be used for both personal and business purposes. The Neo model must ensure that personal and business data and applications are isolated from each other. The Neo model must further ensure that data owners have full control over their data and applications.

The secondary objective of this thesis is to ensure that the user can interact with the data and applications in a secure mobile operating environment using various input and output (I/O) peripherals.

Secondary objective: Expand the Neo model to allow multiple I/O peripherals to be used securely within the mobile operating environment.

1.5 Approach

This section describes the approach taken during the research conducted for the thesis. The aim of this section is not to describe the structure of the thesis, instead it focusses on the process and makes mention of where it applies in the thesis. The structure of the thesis is described in section 1.6.

The approach taken during the research broadly follows the principles of design science research (DSR). DSR identifies six activities necessary during the DSR process. The activities in the process and how it applies to the research in this thesis are (Gregor & Hevner, 2013) (Peffers et al., 2007):

1. **Problem identification**. The first step in any research is to identify the problem that the research intends to solve. The problem background and problem identification has already been covered and is found in section 1.1 and 1.2.
2. **Define research objective**. Gregor and Hevner (2013) describes the process of defining objectives by the studying existing literature. The approach taken in this research is to formalise a primary and secondary objective, defined in section 1.4. An extensive literature study is part of the thesis to help identify existing architectural models and mechanisms that ensure isolation of data and applications. The focus is on the isolation models and mechanisms in existing mobile operating systems, but

also in other technologies such as virtualisation and abstraction. The literature study also evaluates the risks and connection security of I/O peripherals. The focus is on wired and wireless I/O peripherals.

3. **Design and develop**. The outcome of the literature study is to define a set of drivers or deliverables that must be achieved or adhered to in the design of the Neo model. The drivers are derived from the security aspects, both positive and negative, of the existing mechanisms and models. The drivers act as input into the Neo model to ensure the Neo model address the existing shortcomings and produce a model that is more mature than the existing approaches.
4. **Demonstration**. The Neo model was first demonstrated and opened for peer review in 2015 at the SAI conference in London in the form of a paper (du Toit & Ellefsen, 2015). This first demonstration of the Neo model allowed for feedback on the high-level principles of the Neo model.
5. **Evaluation**. The Neo model is evaluated using two mechanisms. The first mechanism is through peer-review acceptance and feedback. Peer-review evaluation occurred through journal and conference articles. After the SAI conference, the further aspects of the Neo model was presented at the International Information Security South Africa Conference (du Toit & Ellefsen, 2015) and IST Africa conference (du Toit et al., 2016). A journal article was also accepted, which specifically evaluated the implementation possibility of the Neo model (du Toit & Ellefsen, 2017). The second method of evaluating the Neo model is by using arguments. Throughout the designing of the Neo model aspects are continuously argued, but Chapter 12 and Chapter 13 specifically argue the case for the Neo model.
6. **Communication**. The last step of DSR is communication. The aim of communication is to communicate all the aspects of the research to an audience. This thesis fulfils the activity of communication.

The six activities are communicated in the thesis, the structure of the thesis addresses each of the above activities and are described in the next section.

1.6 Structure of thesis

The structure of the thesis is grouped into five main parts. Each part is described in more detail in sections 1.6.1 to 1.6.5. The five main parts are:

- **Part 1. The end game.** This part introduces the reader to the eventual secure operating environment that is the result of parts 2, 3 and 4.
- **Part 2. The literature study.** The literature study derives a number of design drivers that affect the design of the model.
- **Part 3. The bridge**. This part links the derived design aspects to the sections of the thesis that addresses the design aspects.
- **Part 4. Building the Neo model.** This part designs the secure operating environment given the results from part 2.

- **Part 5. Evaluation and summary**. The Neo model is evaluated in this section and it provides a summary of the results of the research conducted.

The first of the parts provide a summary of the identified secure operating environment.

1.6.1 The end game

Chapter 2 provides a summary of the identified secure operating environment. The chapter acts as a map that describes the end goal. The concepts introduced in the chapter is a result of the work done in the rest of the thesis. The reader is introduced to the secure container (SCP) and mutual authentication properties (MAP). These two properties act as cornerstones for the Neo model. The Neo model is built on top of SCP and MAP. Another important aspect described in the chapter is the aspect of a ***hypothetical*** mobile computing device, called the Neo device. This device does not exist today but forms part of the Neo model.

1.6.2 The literature study

The literature study spans three chapters. The main focus of the three chapters is to identify existing isolation mechanisms and derive a number of drivers from these chapters.

Chapter 3 focusses on the isolation properties of existing mobile operating systems. The focus is on Apple iOS and Android operating systems. Both these operating systems use the principle of application isolation but use different mechanisms to achieve the isolation. There is also a difference in the mobile operating system support sphere, which has a major influence on the general security of the mobile operating systems.

Chapter 4 evaluates the isolation techniques that have been implemented using other technologies like virtualisation and container-based virtualisation.

Chapter 5 investigates the security landscape of I/O peripherals. Both wired and wireless I/O peripherals are evaluated and the threat landscape is evaluated using three security services from ISO 7498/2 as a guideline.

1.6.3 The bridge

The drivers from the literature study are summarised in **Chapter 6**. The drivers are evaluated and condensed into a set of requirements. The requirements are used in the design of the Neo model.

1.6.4 Building the Neo model

The second last part of the thesis describes the Neo model. **Chapter 7** provides a high-level overview of the Neo model again. It allows the reader to get a view of the major components of the Neo model, without getting into too much detail.

Chapter 8 focusses on the secure container property (SCP) of the Neo model. The SCP is one of the two major properties of the Neo model. This chapter describes the design of the SCP.

Chapter 9 focusses on the mutual authentication property (MAP). The MAP acts as a general property that describes the security built into the communication of the I/O peripherals and the computing device.

Chapter 10 provide more insight into the way in which containers are provisioned, used and finally discarded, if necessary. This chapter provides a greater understanding of the concepts of the SCP and shows how all the components work together to provide isolated computing environments.

Chapter 11 applies the Neo model to two specific use cases. The use cases describe how the Neo model expand the possibilities to solve other problems companies may have. The two use cases focus on are the use of mobile devices as a backup medium. The other use case is the ability of administrators to control access to specific features, data and applications given the physical location of the computing device in the Neo model.

1.6.5 Evaluation and summary

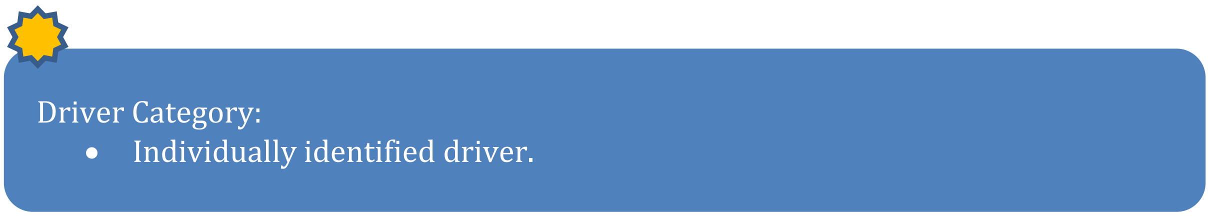
The last part of the thesis evaluates the Neo model in terms of real-world implementation and provides a summary of the work done in the thesis.

Chapter 12 takes the features presented in the Neo model and discusses the implementation possibilities using technology and protocols available at the time of writing this thesis. This chapter provides answers to implementation possibilities.

Chapter 13 acts as a summary and tests the hypotheses and objectives defined in the first chapter.

1.7 Formatting information

In chapters 3 to 5, important drivers are identified that influence the design of the Neo model. When a specific driver or aspect that influence a driver is identified, the driver is written in brackets, with a bold italic font: ***(Individually identified driver)***. The identified drivers are summarised and categorised in a callout as shown in Figure 1.



Driver Category:

- Individually identified driver.

FIGURE 1: SPECIAL CALLOUT (BY AUTHOR)

These call outs provide a mechanism used by the author to keep track of the important aspects discovered during the literature study. They also act as mechanism to ensure that all drivers and aspects are considered during the design of the Neo model.

1.8 Limitations

The Neo model aims to describe a mobile operating environment that is more than just one specific computing component or even one operating system. The model describes a mobile operating environment and the necessary and required components that address the problem defined in this chapter. The devices described in the Neo model are also not devices that exist at the time of writing.

Since the objective of this study is to create a model of a secure mobile operating environment an operational prototype of such an environment is not part of the scope of the deliverables.

1.9 Published papers

The following scholarly articles were published through the work done in this thesis. The related chapters that correlate with the article are mentioned in the following table.

TABLE 1: PUBLISHED PAPERS AND RELATED CHAPTERS

Article reference	Related chapter
du Toit, J. & Ellefsen, I., 2015. A Model for Secure Mobile Computing. London, IEEE, pp. 1213-1221.	Chapter 2, Chapter 7, Chapter 8, Chapter 9
du Toit, J. & Ellefsen, I., 2015. Location Aware Mobile Device Management. Rosebank, IEEE, pp. 1-8.	Chapter 11
du Toit, J., Ellefsen, I. & von Solms, S., 2016. Bring your own disaster recovery (BYODR). Durban, IEEE, pp. 1-12.	Chapter 11
du Toit, J. & Ellefsen, I., 2017. Secure Peripherals in a Converged Mobile Environment. Vancouver, Canada, Springer International Publishing, pp. 296-308.	Chapter 12

The following academic output was also developed through the direct result of the research conducted in this thesis:

TABLE 2: OTHER ACADEMIC OUTPUT

Type	Opportunity	Title	Author\s
Poster presentation	Poster presentation at the University of Johannesburg: Academy of Computer Science and Software Engineering Annual IT Projects Day.	An innovative model for mobile computing	du Toit, J. and Ellefsen, I. (2016, 2017)
Presentation	Oxford university, 2nd Annual Cybersecurity early careers researchers symposium	A Secure Mobile Operating System Model	du Toit, J. (2016)
Conference article and presentation	17th European Conference on Cyber Warfare and Security ECCWS 2018. Oslo, Norway, 2018.	Digital Rights Management to Protect Private Data on the Internet	du Toit, J. (2018)

1.10 Summary

This chapter described the problem addressed in this thesis. The problem states that users need to rely on potential unsafe cloud storage when transferring data between various devices. The objective was defined as creating a model for a secure mobile operating environment, called the Neo model.

The Neo model describes a mobile operating environment that minimises the dependency on cloud storage. It accomplishes the objective by specifying the use of a single computing device that can be used for both personal and business purposes but can interface with various I/O peripherals and isolate company and personal data and apps.

The next chapter provides an overview of the Neo model and introduces the concepts of a secure container property and mutual authentication property. The next chapter also describes the principal features of the Neo model, such as the hypothetical mobile device, called the Neo device and I/O peripheral devices.

Introduction

- Chapter 1 Introduction

The end game

- Chapter 2 Model Overview

Literature study

- Chapter 3 Isolation properties in mobile operating systems (SCP)
- Chapter 4 Isolation techniques in other technologies (SCP)
- Chapter 5 An overview of peripheral security (MAP)

Bridge

- Chapter 6 The drivers for a new model

Building the Neo model

- Chapter 7 Overview of the Neo Model
- Chapter 8 The Secure Container Property
- Chapter 9 The Mutual Authentication Property
- Chapter 10 Container life cycle
- Chapter 11 Novel Implementations

Evaluation and summary

- Chapter 12 Potential implementation of Neo Model
- Chapter 13 Conclusion and future work

Chapter 2 Model Overview

2.1 Introduction

The problem that this thesis addresses, as stated in Chapter 1 section 1.2 page 11, is that users of multiple mobile devices may need to utilise cloud storage to exchange data between the various devices.

It has already been argued that users have multiple computing devices because of the I/O peripheral form factor. Mobile devices with different form factors are used in different situations and physical environments. The unfortunate result of this multi-device scenario is the requirement to exchange information between devices. The information can be business data or personal data that needs to be exchanged.

This chapter provides an overview of the Neo model and describes the progression towards the fundamental properties of the Neo model. The chapter provides a high level overview of the Neo model, which acts as a driving force for the selection of literature study that is found in chapters 3 to 5.

Some of the aspects and concepts described in this chapter are only established in later chapters. These concepts allow the reader a peek into the future of the thesis.

- **Section 2.2** provides the argument and the result that forms the basis of the Neo model that describes a computing device that can be used in various situations and for both personal and business purposes.
- **Section 2.3** introduces the two primary properties of the Neo model. These two properties act as cornerstones for the Neo model and provide direction for the literature study.
- **Section 2.4** summarises the information in sections 2.2 and 2.3 and provides a high-level diagram of the Neo model.
- **Section 2.5** concludes the chapter and provide a breakdown of the topics covered in chapters 3 through 5.

2.2 A one-device ecosystem.

Multiple computing devices are used by many different people for both business and personal use. Smaller smartphone sizes devices are easy to hold in your hand, while bigger tablet sized devices provide a larger viewing area for data. Tablet and smartphone devices have limited input peripheral capabilities and have limited storage and processing power.

The Neo model considers a mobile operating environment where a hypothetical mobile computing device can be used that does not have any built-in I/O peripheral. This hypothetical device is called the Neo device. The Neo device has enough storage and processing capacity to be used for both personal and business purposes. The Neo device has the capability to be docked in a docking station when used at a desk and has a battery that provides electricity when no power outlet is available.

The user interacts with the Neo device using various I/O peripherals. The I/O peripherals connect wirelessly or through the docking station. The I/O peripherals can be any form factor the user requires.

The Neo device connects to the network either wirelessly or through the docking station. The docking station also acts as a power supply and recharges the Neo device's battery.

It is envisioned that when the Neo device is used while mobile, that the Neo device may be carried in a special pouch which may be attached to the user. The user interacts with the Neo device using mobile I/O peripherals.

The Neo model at its core describes a computing device that is seen as a general purpose computing device that can easily be used for both personal and business purposes, both mobile or stationary.

The next section describes the primary properties of the Neo model that assures controlled and secure computing for both business and personal use.

2.3 Two properties.

The Neo device allows itself to be used for both personal and business purposes. The ability to use one device for both personal and business use opens up a number of challenges with regards to ownership of data. This section identifies two important properties that underpins the design of the Neo model.

In a computing environment, data can be categorised according to ownership. In the case of personal use, the data generated by the user belongs to the user and as such the user is defined as the owner of the data. When a computing device is used for business purposes the data consumed and generated belongs to the company.

The Neo model must support the principle of data belonging to multiple owners stored on a device, which may be owned, by another owner. The owner of the data has the right to stay in control of the data and manage the access control of the data.

When data, or applications (apps), that belong to a company is stored on a mobile device, the data or applications are exposed to different threats than if they were only stored on computing devices on the company premises.

Some of the threats exposed to mobile devices are the theft of data and device when a device is stolen. Data can also be stolen through accidental data exposure through shoulder surfing. Unauthorised access by malicious actors, insecure wireless networks and easy to crack authentication mechanisms are all part of the mobile threat landscape.

Data owners, such as companies, require a high level of assurance that business data and applications are sufficiently protected while being used on a mobile device.

All of the above leads to the identification of two properties of the Neo model. These properties are briefly defined in section 2.3.1 and section 2.3.2 and fully explored in Chapter 8 and Chapter 9. The Secure Container Property (SCP) and the Mutual Authentication Property (MAP) is now explored which addresses the above.

2.3.1 The Secure Container Property (SCP).

The first major property of the Neo model is the secure container property (SCP). The SCP ensure that data and applications are grouped according to ownership and isolated from other containers. Furthermore, the owner of the secure container has full control over who, when, where and what can access the applications and data in the secure container.

2.3.2 The Mutual Authentication Property (MAP).

Since the Neo device does not have any I/O peripherals built-in, communication to and from wireless I/O peripherals must be secure. The secure communication of I/O peripherals are grouped under the mutual authentication property (MAP). The MAP ensures three security aspects are enforced when the Neo device communicates with the I/O peripherals. The three aspects are:

- **Mutual authentication**. Mutual authentication ensures that the I/O peripheral is identified and authenticated with the Neo device, but the opposite is also true. Mutual authentication ensures the Neo device is authenticated with the I/O peripheral. The mutual authentication aspect of the Neo model takes it a step further and ensure the authentication extends to the secure container on the device. This ensures that access is only granted to I/O peripherals if the company approved the use of the specific I/O peripheral with the specific secure container.
- **Authorisation**. Authorisation ensures that the owner of the data inside a secure container can decide who can access the applications and data. The authorisation extends to also include the I/O peripherals. This means the owner must authorise an I/O peripheral access to a secure container before the I/O peripheral can access it.
- **Confidentiality**. Confidentiality ensures that when data is travelling between the I/O peripheral and the Neo device that the data is transmitted in such a way that it is nonsensical to any listening devices, except the approved devices. The confidentiality aspect also extends to more than just the Neo device and I/O peripheral. The confidentiality aspect ensures that other secure containers cannot intercept the data and control commands sent between the approved secure container and I/O peripheral.

The SCP and MAP act as the two major design requirements for the Neo model. The next section summarises the Neo model at a high level.

2.4 The big picture.

The SCP and the MAP are the cornerstones of the Neo model. A visualisation of the SCP and MAP assists in a better understanding of these two properties.

Figure 2 provides an overview of the Neo model. The figure depicts a Neo device, which is centrally located in the diagram. There are also a number of I/O peripherals, located in the corners of the diagrams, communicating with the Neo device.

The Neo device hosts a number of secure containers. The secure containers depict the SCP. Secure containers cannot interact or communicate with other secure containers on the Neo device.

The identification and authorisation service enfolds the secure containers and acts as a component that enforces the MAP. The identification and authorisation service ensure all I/O peripherals and Neo device mutually authenticate, the service further ensures communication between the secure container and the I/O peripheral is encrypted.

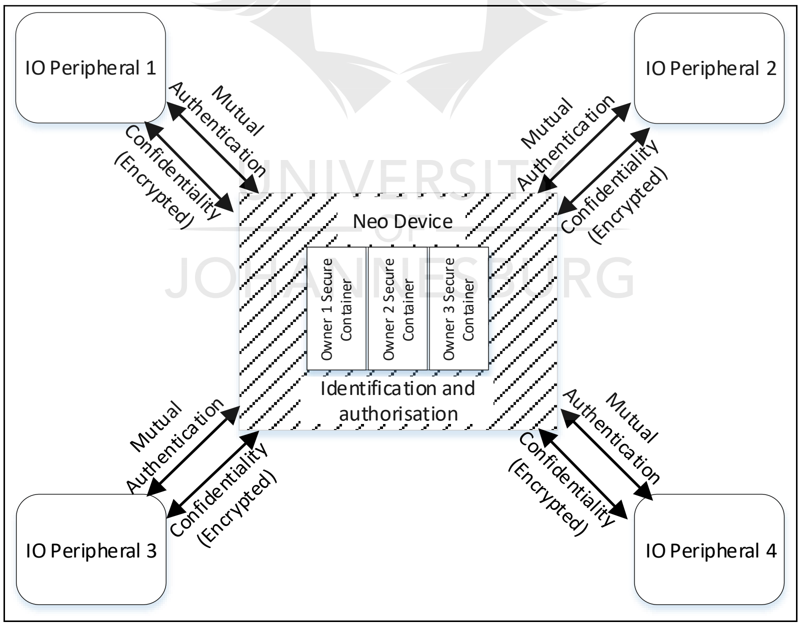


FIGURE 2: OVERVIEW OF THE NEO MODEL (DU TOIT & ELLEFSEN, 2015).

The SCP and MAP can be visualised using Figure 2.

2.5 Conclusion.

This chapter provided an overview of the Neo model. The importance of using just one device for both business and personal use was discussed.

The Neo model uses a hypothetical mobile computing device, called the Neo device. This device does not have any I/O peripherals built-in, instead relies on either wireless I/O peripherals or I/O peripherals connected to a docking station.

The Neo model defines the SCP that ensures data and applications are grouped according to ownership and is stored in a secure container on the Neo device. The MAP ensures communication between Neo device and I/O peripherals are secure.

The SCP provides a level of security using isolation. The isolation property is also available in mobile operating systems in use at the time of writing this thesis. Chapter 3 evaluates the isolation properties of the two major mobile operating systems at the time of writing the thesis.

Isolation techniques are also used in other technologies. These technologies and the way in which isolation is used are explored in Chapter 4. The techniques discussed and used in the Chapters 3 and 4 acts as input to the SCP design of the Neo model.

The MAP concerns itself with the mutual authentication and authorisation of the Neo device and I/O peripherals and the confidentiality of communication between the secure container and I/O peripheral. Chapter 5 describes the existing authentication, authorisation and confidentiality threats for wired and wireless peripherals, as well as the mechanisms that address these threats. The existing mechanisms are evaluated to act as drivers for the MAP of the Neo model.

The next part starts with the literature study of this thesis and starts by evaluating the isolation mechanisms used by existing mobile operating systems and identifying drivers for the Neo model.

Introduction

- Chapter 1 Introduction

The end game

- Chapter 2 Model Overview

Literature study

- Chapter 3 Isolation properties in mobile operating systems (SCP)
- Chapter 4 Isolation techniques in other technologies (SCP)
- Chapter 5 An overview of peripheral security (MAP)

Bridge

- Chapter 6 The drivers for a new model

Building the Neo model

- Chapter 7 Overview of the Neo Model
- Chapter 8 The Secure Container Property
- Chapter 9 The Mutual Authentication Property
- Chapter 10 Container life cycle
- Chapter 11 Novel Implementations

Evaluation and summary

- Chapter 12 Potential implementation of Neo Model
- Chapter 13 Conclusion and future work

Chapter 3 ISOLATION PROPERTIES IN MOBILE OPERATING SYSTEMS (SCP)

3.1 Introduction

In Chapter 1, the problem statement referred to the use of mobile devices for both personal and business use and the sharing of information in either the private or business domains. It was highlighted that data owners need to isolate and control the programs and data belonging to them.

The aim of this chapter is to produce a set of drivers that must be considered when designing a secure mobile operating environment. To help with the identification of the drivers, existing methods used by mobile operating systems to isolate applications and data from each other are discussed and described.

The chapter also investigates some of the existing vulnerabilities in mobile operating systems that allow attackers to steal information outside the scope of a specific application. The existing vulnerabilities are used to evaluate the design methodology and test the effectiveness of the existing models.

The above-mentioned information is necessary to establish a set of drivers that influence the design and requirements for the Neo model.

According to Netmarketshare (2016), Android and Apple iOS make up more than 92% of the mobile market share in 2016. IDC uses device shipments as measurement, show that Android and Apple iOS has 96% of the mobile device market as of the 2nd quarter of 2015 (IDC Research, Inc., 2015). It is clear that Android and Apple iOS are the two major players in the mobile operating system market. For this reason, this chapter focusses on the architecture and design of Apple iOS and Android.

- **Section 3.2** of this chapter investigates the Apple iOS operating system.
- In **section 3.3** the Android Operating System is investigated. Throughout this and upcoming chapters, security drivers are summarised and grouped together in special callouts located at the end of relevant paragraphs. Figure 1 shows how one of these callouts look. Each callout consists of a driver category, with individually identified drivers grouped together. These callouts list the drivers that are taken into account when designing the Neo model.
- **Section 3.4** summarises the callouts as a set of drivers that act as input in the Neo model design.

This section makes use of the call outs that was described in Chapter 1 section 1.7 page 15. Individual drivers are identified and put inside brackets, with bold and italic font and then grouped according to categories in a call out with a blue or dark background.

3.2 Apple iOS Operating System

At the time of writing this document, the Apple iOS Operating System (iOS) (Apple Inc., 2015) was in version 9.0. Aspects described for iOS is based on the features available in version 9.

The advantage that Apple has over its competitors is that Apple controls both the software stack and hardware stack that are available for Apple mobile devices (Apple Inc., 2015). This allows Apple to define and implement both hardware and software security mechanisms to strengthen the overall security of the mobile operating system.

It is interesting to note that Apple iOS is not as popular in security research as Android. This may be because Android is a bit more open, or it may be because Android has a lot more attack surfaces and attack vectors than Apple iOS (Ahmad et al., 2013).

This section describes the following aspects of the Apple iOS and highlights the structure of this section:

- **Section 3.2.1**. How applications are isolated from each other.
- **Section 3.2.2**. How applications share and communicate with each other and the base operating system
- **Section 3.2.3**. Describe how private data is protected from business interrogation using mobile device management systems.
- **Section 3.2.4.** Describes some of the known attacks against Apple iOS that allows attackers to steal information from other applications.

3.2.1 Application Isolation

Apple uses various mechanisms to isolation application processes and data from other applications and to also isolate and protect the operating system from applications. The most important feature in iOS is that each application executes in a sandbox. Figure 3 shows the existence of the App Sandbox inside iOS (Apple Inc., 2015).

Each App runs inside its own sandbox isolating the application code and data from the rest of the system ***(Application level isolation)***. Apart from the App Sandbox, iOS also provides a number of other mechanisms that protect the operating system and other applications from misbehaving applications. The mechanisms are described below.

Design methodology:

- Application level isolation.

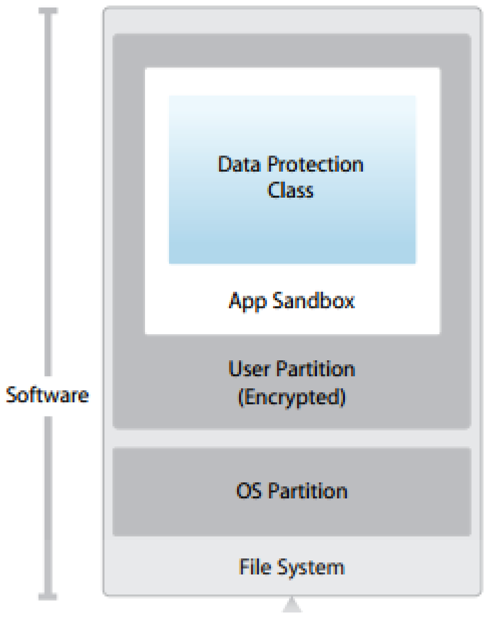


FIGURE 3: THE IOS APP SANDBOX (APPLE INC., 2015)

The mechanisms can be summarised as follow (Apple Inc., 2015):

- **Unique home directory access.** Each application is assigned a random home directory when an application is installed. The application has only access to this home directory on the file system. This ensures that operating system files belonging to other applications are not accessible by each other ***(File system boundaries).***
- **Non-privileged user permissions.** All third-party applications run as a user called "mobile" ***(Same user account)*** in the operating system. The "mobile" user does not have the required permissions to access/use/execute system level processes. When iOS starts it also mounts most of the operating system as read-only ensuring that no permanent changes to system files can be made ***(Non-privileged process execution)***.
- **The entitlement mechanism** ensures a certain level of permissions in the operating system. This mechanism is built-into each application and signed by the developer ensuring that it cannot change once it is installed. The entitlement mechanism warns a user to which shared resources the application has access to ***(Controlled operating system interfaces)***.
- **Background processing** is limited to the operating system provided application programming interfaces (API). This ensures that applications cannot misbehave and impact the performance of the device ***(Controlled processing).***
- **Address space layout randomization** (ASLR) minimises attacks using memory management bugs. When an application starts it is loaded into a different area of memory from other devices. Attacks against memory management bugs require applications to run in a known area of memory. By randomizing memory layout,

these attacks are severely hampered ***(Modern data and code execution protection)***.

- Use **ARM's Execute Never** (XN) feature. The ARM processor has the ability to mark memory areas as no execute. This means that data in these memory areas cannot be used to execute malicious code. This feature further protects the system from applications that require just in time compiling and running of code ***(Modern data and code execution protection)***.

Isolation mechanisms:

- File system boundaries.
- Non-privileged process execution.
- Same user account.
- Controlled operating system interfaces.
- Controlled processing.
- Modern data and code execution protection.

The Apple iOS mobile operating system has a number of mechanisms and features that protect applications from accessing code and data from other applications and the base operating system. A collective term for these mechanisms is to sandbox the applications (Ahmad et al., 2013).

The information required by applications and generated by applications can be shared among applications. The next section describes the protection mechanisms that allow applications to connect to different hardware components and share information among other applications.

3.2.2 Application communication

In section 3.2.1 the mechanisms that isolate applications from each other and the system were described. In many cases, applications provide functionality and features that require access to other application data or system information. Example: A photo manipulation application needs to have the ability to access photos that were taken by the camera application on the iOS device. This section describes the mechanisms that allow applications to share information between the system and each other.

The mechanisms that allow applications to gain access to information beyond the scope of their sandbox are:

- **Entitlements.** Entitlements are the permissions required by an App to function properly (Apple Inc., 2015). **Example**: A navigation App requires at least access to the GPS in order for the App to determine the location of the device. Some Apps may also require access to data produced by other Apps like contacts, photos or any other type of data. ***(Specialized access control mechanisms).***

- **Extensions**. Extensions allow an app to be used in different areas of iOS. An extension can allow an app to become available as a widget on the home screen or allow an app to extend its functionality from within another app. Extensions each run in their own sandbox, which means that calling apps never gain direct access to the parent app for an extension. Another important aspect of extensions it that the entitlements granted to an app also applies to the apps extensions, but not to extensions called by the app. **Example**: Your app can become part of the Apps that show up if you want to share something. You can, for instance, share a document to your app, if your app has a special document editing capability (Apple Inc., 2016). ***(Specialized access control mechanisms).***
- **URL Scheme**. The URL Scheme in iOS allows one app to use the functionality of another app. Example: A custom app can request directions from the Google Maps app, by calling a URL scheme registered by the Google Maps app (Google Inc., 2016). When calling a URL scheme, the calling app passes certain parameters to the called App. These parameters are the only information that flows from the one app to the other. The two apps still run independently without one app accessing the data belonging to another app. ***(Apps cannot start processes belonging to another app directly).*** The called app must also define these URL Schemes during development time. If an app has no URL Scheme defined, then another app can never call this app. ***(Specialized access control mechanisms)***.

Application communication:
- Specialized access control mechanisms
- Apps cannot start processes belonging to another app directly

At the end of the day, it is up to the developer of the app to decide which data is shared with other apps, by developing it into the app as a feature (Thiel, 2016). The developer can use any of the above mechanisms

3.2.3 Privacy protection

Another reason why isolation is necessary for mobile devices is to isolate corporate and personal information from each other because mobile devices are used for both personal and corporate use.

Apple enables corporate access to iOS configuration through mobile device management systems. These systems enable and configure features on the phone that ensure private information stays private and corporate information cannot be misused by device owners. ***(Operating system controlled sharing)***.

The two aspects that are important when using phones in a corporate environment are:

Aspect 1. Can the business gain access to personal data through the mobile device management systems?

Aspect 2. Can the device owner misuse business information that is available on their device?

Aspect 1 provides assurance for a device owner that the corporate cannot gain access to the data belonging to the device owner. Data such as photos and information stored on the iCloud cannot be accessed by the corporate.

Aspect 2 addresses the confidentiality of corporate data. It tries to provide assurance for a corporate that their data cannot be copied from corporate data sources into private data sources.

Apple configures VPN connections per app. ***(Per-app VPN)***. This ensures that data to and from the app can only be exchanged to defined corporate owned end-points (Apple Inc., 2016)

The second feature that enables the isolation of corporate-owned data from personal owned apps is called Open In Management. Through the use of a mobile device management (MDM) system, the corporate defines certain apps as managed apps. These apps are to be used for corporate purposes. The apps installed by the device owner are categorised as unmanaged apps (Apple Inc., 2016). As soon as an app becomes managed, data cannot be transferred between the managed app and any unmanaged app. ***(App categorisation ensures controlled data sharing)***.

iOS provides a number of initiatives to isolate private and corporate apps and data from each other. The mechanisms available for iOS, however, does not allow fine-grained control over corporate data, which may categorise apps into different levels of authorisation and control. You may, for instance, want to not only control the data that flows between apps but also provide some control over the personal areas of the iOS to minimise the chances of data leaks. Example: It is not possible to only enable the corporate apps on the phone in certain physical areas of a building to minimise the chances of unauthorised data capturing. ***(Limited granular access control).***

Privacy protection:

- Operating system controlled sharing
- Per-app VPN
- App categorisation ensures controlled data sharing
- Limited granular access control

The next section describes some of the attack scenarios available for iOS.

3.2.4 Known attacks against iOS.

In the quest to develop the Neo model that minimises common vulnerabilities that exist in today's mobile operating system an understanding of the attack vectors for mobile operating systems is important.

Thiel (2016) identify a number of components that are part of the Cocoa Touch API that is usually targeted by attackers. The Cocoa Touch API is the term used to describe the programming interface used by the iOS operating system (Thiel, 2016) (Lettner et al., 2011).

Miller et al (2012) also identify a number of techniques and components targeted by attackers. The components and techniques identified by Thiel and Miller are listed below, with a short explanation of each (Thiel, 2016) (Miller et al., 2012):

- **iOS-Targeted Web Apps**. iOS apps may contain content provided by web servers. The content is made available using the iOS UIWebView component. Apart from displaying static web content inside apps, it also exposes JavaScript functionality from within apps. The JavaScript can be dynamically downloaded from web pages and can act as an attack vector for iOS. Thiel demonstrates the ability of an attacker to inject JavaScript code into an app that makes use of UIWebView that can expose the contacts on iOS 8.0. ***(Web exposed components)***.
- **Data Leakage**. The data leakage category covers the problems where data leaks into other areas of the operating system, making it available to attackers. One of these components that are available to attackers it NSLog. NSLog is often incorrectly used by developers to output debug information of an app. The debug information is not isolated and is logged in the Apple System Log. This means that if the developer uses this method to output debug information, that may contain sensitive information, then this log information is available to anyone that has access to the iOS device. Most of the methods described by Thiel requires that an attacker has physical access to a mobile device.
- **Injection attacks**. When an application makes use of data entry or javascript, injection attacks become an option for attackers. Injection attacks execute either SQL or JavaScript code given in normal data entry fields of an app. It is the responsibility of an app developer to use proper programming techniques to minimise the chances of executing SQL or JavaScript. These types of attacks normally give an attacker access to the data already available to an app, without having to properly authenticate. A famous example of this was the Skype app that was vulnerable to a cross-site scripting attack and leaked the phone's address book. ***(Injection attacks)***.
- **Return Oriented Programming**. Apple iOS makes use of data execution prevention (DEP) and code signing to ensure that the memory areas allocated for data, like the heap, stack and data areas cannot execute code. This means that

normal buffer overflow attacks that create custom code inside the stack cannot be used in iOS. Return Oriented Programming (ROP) calls already approved functions available in the operating system and app by manipulating the stack registers. This allows an attacker to use either iOS system calls, or app functions out of sequence to leak data (Wang et al., 2013) (Miller et al., 2012). ***(Use complex attacks to expose data)***.

- **Jailbreaking**. Jailbreaking gives a user of an iOS device full control over the device. It allows the user to install apps that have not been approved by Apple. Security researchers require iOS jailbreaking to investigate the security of operating systems and apps. Users jailbreak their phones so that they can break copyright on apps and install stolen or unapproved apps. As soon as this happens, it allows an attacker to load an app that has not gone through the stringent security testing applied by Apple before an app is allowed in the App Store (Miller et al., 2012). ***(Jailbreaking)***.
- **Baseband attacks**. Baseband attacks utilise a modified cellular base station in close proximity of the attacked cellular phone. The base station intercepts the communication of the device and attacks the baseband software stack, which in most cases is a dedicated real-time operating system, separate from the smartphone's operating system. It allows an attacker to take control over the phone's microphone, allows the attacker to intercept voice calls and intercept data flow (Miller et al., 2012)

The above list does not take into account deliberate misuse from a user. Example: The user giving the phone to an attacker and providing the attacker with the password or biometric information to unlock the phone.

It also seems as if the number of malware targeting iOS remains relatively small. In 2014 there were three malware apps detected for iOS and in 2015 there were seven (O'Brien, 2016).

Attack methods and vulnerabilities:

- Injection attacks.
- Web exposed components.
- Use complex attacks to expose data.
- Jailbreaking.

The Apple iOS mobile operating system has a number of mechanisms and features that protects and isolation data between apps. iOS also gives administrators in a corporate environment the ability to isolate and control corporate data, without affecting personal data. The next section investigates the Android operating system, which is a much more open operating system and used by a lot more people than iOS.

3.3 Android Operating System

The Android Operating System (Android, 2016) is an operating system developed for mobile devices by the Open Handset Alliance (OHA). The OHA consists of 78 different companies that are responsible for developing the Android operating system.

Android is available as open source and is available to anyone from a central repository (Google Inc., 2016). Android is extremely popular and holds 65.58% market share for Mobile and Tablet operating systems with the second biggest deployment being iOS, holding 27.24% market share as of June 2016 (Net Applications.com, 2016).

To keep in line with the structure of the section covering iOS, the isolation and security of Android is described in four sections.

- **Section 3.3.1**. The sections include a description of how applications are isolated on the Android operating system.
- **Section 3.3.2** describes how apps are able to communicate with each other despite the isolation model being enforced on them.
- **Section 3.3.3** includes how private and corporate data is isolated from each other.
- **Section 3.3.4** concludes with some of the known attacks against the Android operating system.

3.3.1 Application Isolation

Android also uses a sandbox model to ensure that each app is running in its own sandbox, separated from other apps and the system. Android apps run in the user mode of the operating system and run as specialised Java apps. ***(Application level isolation)***.

Design methodology:
- Application level isolation.

Figure 4 shows the Android operating system architecture. Starting at the bottom the base operating system is running on a Linux kernel. From there various layers provide different functionality until it ends up with the user-installed apps that run at the top layer.

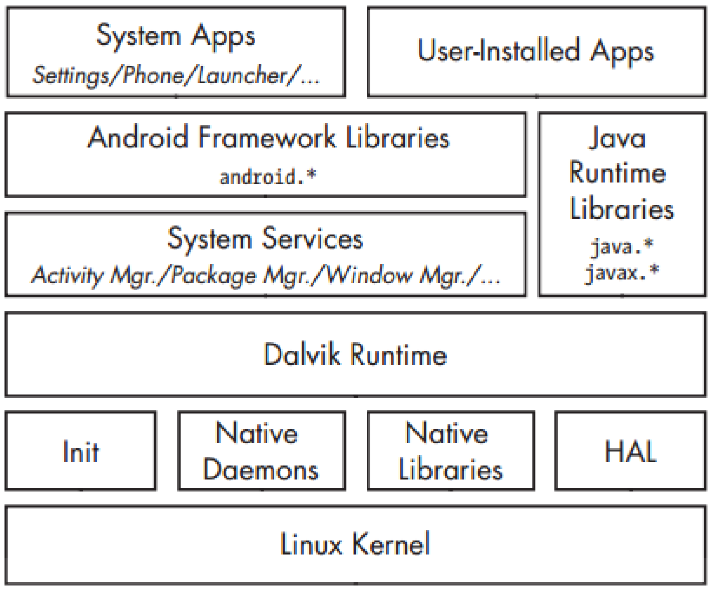
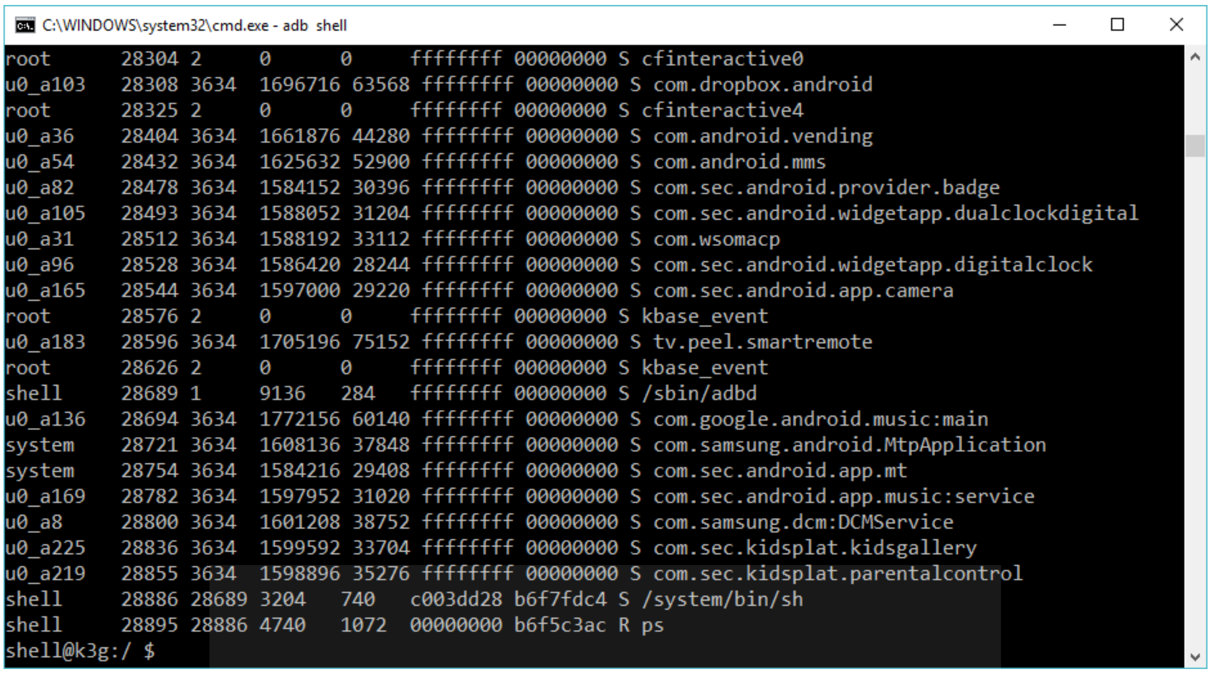


FIGURE 4: THE ANDROID ARCHITECTURE (ELENKOV, 2015)

The mechanism that Android uses to enforce sandboxing is through the use of per-app user identification. Android was originally developed to support only one user per device. That means that the design requirement stated that only one user would use one device and as such we can assume that the person that is using the device is the authorised user. With this in mind, the developers of Android used the multi-user capability in Linux and applied it to the apps that run on the operating system. ***(Per-app user accounts)***.

As soon as an app is installed on Android, a unique user-id is created for that app. The app is also installed in its own directory under the /data partition of the device. All the files of the app, which includes the executable, libraries and data reside under this app specific directory and only the user-id created for the app has access to these directories and files. ***(File system boundaries)***. ***(Non-privileged process execution)***.

When the app runs, it also runs under the context of the app's user id. Figure 5 displays a number of processes running on Android. For interest purposes, the first column is the user-id that the process runs under and the last process is the name of the process. The second line shows that there is a user u0_a103 that is currently running the DropBox app, described as com.dropbox.android in the running processes.



```
root      28304 2     0       0     ffffffff 00000000 S cfinteractive0
u0_a103   28308 3634  1696716 63568 ffffffff 00000000 S com.dropbox.android
root      28325 2     0       0     ffffffff 00000000 S cfinteractive4
u0_a36    28404 3634  1661876 44280 ffffffff 00000000 S com.android.vending
u0_a54    28432 3634  1625632 52900 ffffffff 00000000 S com.android.mms
u0_a82    28478 3634  1584152 30396 ffffffff 00000000 S com.sec.android.provider.badge
u0_a105   28493 3634  1588052 31204 ffffffff 00000000 S com.sec.android.widgetapp.dualclockdigital
u0_a31    28512 3634  1588192 33112 ffffffff 00000000 S com.wsomacp
u0_a96    28528 3634  1586420 28244 ffffffff 00000000 S com.sec.android.widgetapp.digitalclock
u0_a165   28544 3634  1597000 29220 ffffffff 00000000 S com.sec.android.app.camera
root      28576 2     0       0     ffffffff 00000000 S kbase_event
u0_a183   28596 3634  1705196 75152 ffffffff 00000000 S tv.peel.smartremote
root      28626 2     0       0     ffffffff 00000000 S kbase_event
shell     28689 1     9136    284   ffffffff 00000000 S /sbin/adbd
u0_a136   28694 3634  1772156 60140 ffffffff 00000000 S com.google.android.music:main
system    28721 3634  1608136 37848 ffffffff 00000000 S com.samsung.android.MtpApplication
system    28754 3634  1584216 29408 ffffffff 00000000 S com.sec.android.app.mt
u0_a169   28782 3634  1597952 31020 ffffffff 00000000 S com.sec.android.app.music:service
u0_a8     28800 3634  1601208 38752 ffffffff 00000000 S com.samsung.dcm:DCMService
u0_a225   28836 3634  1599592 33704 ffffffff 00000000 S com.sec.kidsplat.kidsgallery
u0_a219   28855 3634  1598896 35276 ffffffff 00000000 S com.sec.kidsplat.parentalcontrol
shell     28886 28689 3204    740   c003dd28 b6f7fdc4 S /system/bin/sh
shell     28895 28886 4740    1072  00000000 b6f5c3ac R ps
shell@k3g:/ $
```

FIGURE 5: PROCESSES RUNNING ON ANDROID (BY AUTHOR)

The multi user-id mechanism ensures that one app does not have access to the application binaries and data belonging to another app. Only processes running in the context of the root user has access to the whole directory structure under the Android operating system.

Isolation mechanisms:

- File system boundaries.
- Non-privileged process execution.
- Per app user accounts.

This section described in basic terms how Android isolated app binaries and app data from each other. The next section describes how apps can share information with each other and gets access to phone features, like the camera or dialer.

3.3.2 Application communication

Mobile apps that do not interface with smartphone functionality or data provided by other apps are very limited in functionality. Android provides a number of mechanisms for apps that enable apps to share data with other apps, use smartphone functionality and access data from other apps.

This section describes the major mechanisms available for Android apps to share and gain access to data and functionality outside their sandbox.

Figure 6 shows three Android apps, each app running as a Dalvik Virtual Machine (Elenkov, 2015). Each app consists of a number of Android components. Four of these components enable the communication of data to and from the Android app.

The four components of interest are:

- **Section 3.3.2.1**. Activities
- **Section 3.3.2.2**. Broadcast receiver
- **Section 3.3.2.3**. Content provider
- **Section 3.3.2.4**. Service

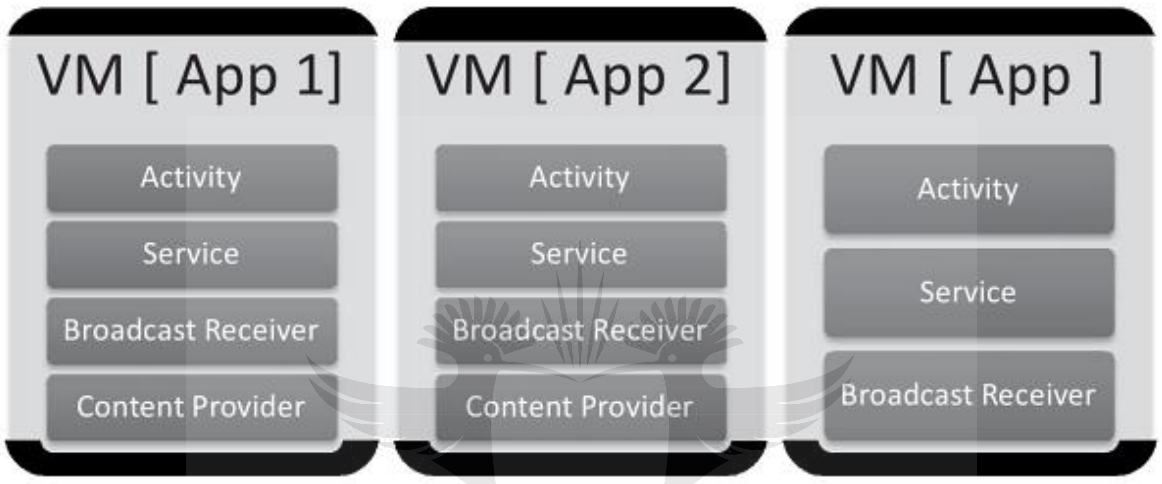


FIGURE 6: ANDROID APP COMPONENTS (MISRA & ABHISHEK, 2013)

Apart from the above four mentioned components, there are also special flags called MODE_WORLD_READABLE and MODE_WORLD_WRITABLE (3.3.2.5) and the ability of Android to store and access data on external storage (3.3.2.6). This brings the total number of components described in the following sections to six.

3.3.2.1 Activities

An Android app consists of one or more activity. An activity is basically a screen. One activity can call another activity within the same app. It is also possible for a user to start activities themselves in an app. Given the correct permissions, an activity can also call an activity belonging to another app (Misra & Abhishek, 2013). ***(Specialised access control mechanisms)***.

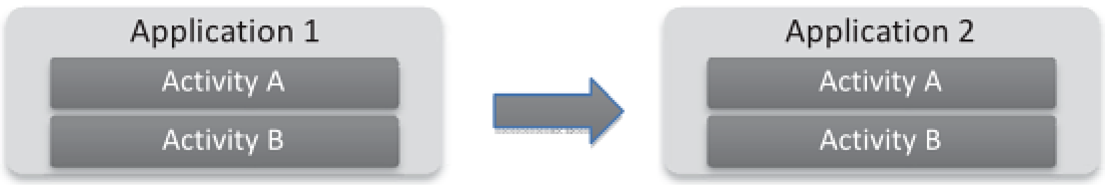


FIGURE 7: ACTIVITIES IN APPS (MISRA & ABHISHEK, 2013)

3.3.2.2 Broadcast receiver

Broadcast receivers work with broadcast senders. Broadcast receivers receive notifications of events that occur in the operating system or other apps (Google Inc., 2016) (Elenkov, 2015). An example of a system-wide broadcast is where the time has changed (ACTION_TIME_CHANGED). If your app is time sensitive, then you can perform the necessary actions given the information provided by the system.

Before an app can receive broadcast notifications, special permissions need to be granted to the app during installation. The app request permissions by specifying the requested permission in the app's manifest file. This permission is then displayed during app installation time and the user must allow the app to receive this information (Misra & Abhishek, 2013). ***(Specialised access control mechanisms)***.

3.3.2.3 Content Provider

A content provider provides general content to other apps. Apps that want to access content from a content provider must request permission during installation from the user. Android has built-in Calendar and Contacts providers and since Android 4.4 it also has a Storage Access Framework, or in other words, a document provider, that support access to files. The document provider has built-in support for Downloads, Images and Videos (Google Inc., 2016).

The document provider allows interaction between apps in a special way. Through the downloads provider, it is possible for the user to receive an email with a document attached to the email. When the user opens the attached document, the document is downloaded to the Downloads provider, which then opens the system picker on the user's device, to select an app registered to open that specific file type.

The advantage of the document provider is that it allows apps to truly share documents between apps and can make use of any type of storage provider, which may include any cloud-based providers. This makes it possible for users to truly collaborate on the same document with each other. The disadvantage of this framework is that if implemented incorrectly with the wrong permissions, confidentiality is at risk. ***(Developers control access to app data)***.

3.3.2.4 Service

A service is a component of an app that has no user interface. Once started, an app can bind to a service through an intent declaration. Normally this is allowed inside an app, in the case where an app consists of both a service (non-user interface) and an activity (user interface, the activity can bind to the service. It may also be possible for another app to bind to a service (Elenkov, 2015). ***(Developers control access to app processes)***.

3.3.2.5 Special flags (MODE_WORLD_READABLE and MODE_WORLD_WRITABLE)

Apart from the document provider mentioned in 3.3.2.3, developers can also build their app, so that when a document is created, special flags can be set for the document. These flags are called MODE_WORLD_READABLE and MODE_WORLD_WRITABLE (Elenkov, 2015).

These flags do exactly what they say, it allows any other app to read or write to the file. This has serious security implications because it opens the file up to any app without having to gain special permissions. From Android 4.2 these two flags have been deprecated in the Android SDK but are still allowed if your app must support Android versions before 4.2 (Google Inc, 2016).

3.3.2.6 External storage

One more mechanism available for Android developers and apps is the ability for an app to store and read files on something called external storage. Android categorise storage as either internal or external. External was originally intended to be removable storage that can be plugged into the Android device. This has however also been implemented by many Android OEM manufacturers as a special partition on the built-in flash memory of the Android device (Google Inc., 2016)

When an app wants to either read or write files to external storage the app must request permission from the user during installation. When an app has the ability to write files to external memory, the files can be either public or private. Private files can only be accessed by the app that created it and public files can be accessed by any app that has permissions to read or write to external memory. Many app developers and OEM manufacturers use this feature to share files between apps, which may create security implications. ***(Developers can use external storage to access data from other apps)***.

Application communication:

- Specialised access control mechanisms
- Developers can use external storage to access data from other apps
- Developers control access to app processes
- Developers control access to app data

The Android SDK has a lot of components that allows developers the integrate their apps with each other, but also share data between apps. Many of these features must be implemented with care and rely on security permissions granted by the user during installation. The next section describes how Android protects data and apps created for corporate use and private use.

3.3.3 Privacy protection

In the previous sections, the mechanisms that allow apps to communicate with each other were described. In an environment where a mobile device is used for both personal and business purposes, corporations are concerned with the ability that the device gives a user that may jeopardise the confidentiality of the business data that may reside on the device if it is used for both business and personal use.

In the quest to enable control over Android devices in corporations, businesses often implement mobile device management (MDM) systems. There are many mobile device management systems available to corporations, and many of them enable the company to control the data that reside on the device. This should be a concern to the owner of the device because the corporation may have access to personal information stored on the device.

It is this dilemma that mobile operating systems should help solve. Organisations need control over their data and apps and employees need their private information protected.

Android enables the control and isolation of business apps by making use of three mechanisms introduced in Android over time. These mechanisms are grouped under a feature called Google for Work (Google Inc., 2016). Apps controlled under the work profile are grouped together on the Android screen as shown in Figure 8.

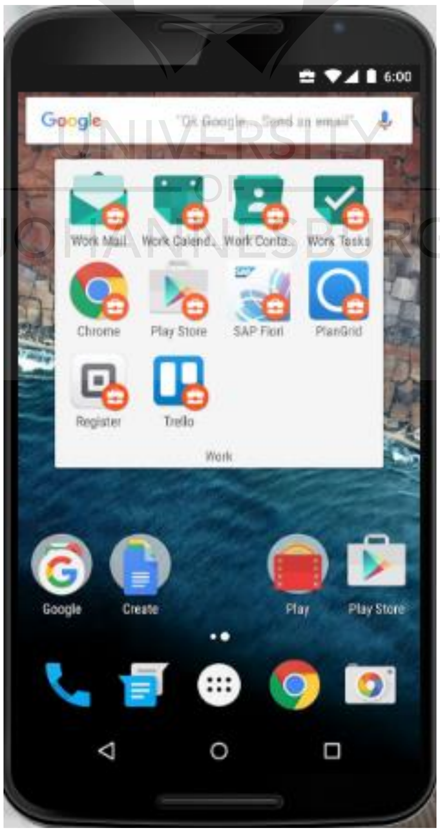


FIGURE 8: GOOGLE FOR WORK (GOOGLE INC., 2016)

The three mechanisms that enable the isolation and control of business apps and data are:

- **Multi-user accounts**. Android isolates the business apps and data from personal apps and data by leveraging the multi-user capability of the Linux operating system. It has already been mentioned that each app runs in the context of its own user-id. User-ids from Android 4.2 were modified to enable multiple actual users to use an Android device (Elenkov, 2015). Corporates can now install apps, with its associated sandbox data under a corporate controlled user, called a work profile (Google Inc., 2016). This isolates both the processes and data from each other, even if it is the same app that exist in both the device owner's account and the corporate account. ***(Multi-user accounts)***.

  Figure 9 shows an example of an email Android app that consists of both a personal and corporate profile. When the personal profile email is executed it runs under a user-id u0-a13 and when the email app is run with the work profile, the app executes under user-id u10_a13.

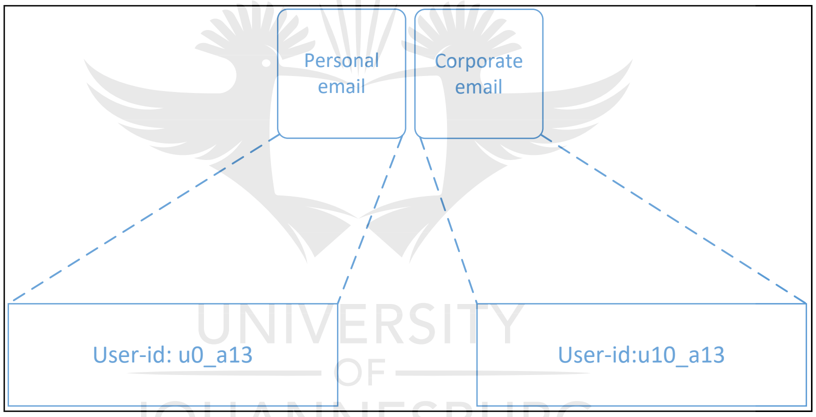


FIGURE 9: ONE ANDROID APP WITH TWO PROFILES (BY AUTHOR)

- **Device Administrators**. Android has a concept called a device administrator. A device administrator is allowed to control some device profile settings. Example: A device administrator may control the password policy on the device, enforcing complex passwords and minimum password length. From Android 5 onwards a special device administrator policy, called account wipe was introduced (Google Inc., 2016). This allowed a corporate-owned mobile device management system, to delete all the apps and data belonging to a work profile. ***(Device admin – permissions)***.
- **Per-user virtual private network (VPN)**. Android allows a corporate to not only group apps together under a work profile, but also control any network data to and from these apps to be routed through a per-app VPN. This means any traffic generated by a corporate-owned app is routed and encrypted through a corporate VPN (Elenkov, 2015). This enforces the concept that network traffic generated

from the corporate controlled apps can only transfer between the app and the corporate network. ***(Per user\per app VPN)***.

The above mechanisms enable corporates to control and manage corporate-owned apps and data, it also assures a device owner that the corporate managed MDM system only controls a subset of the security of the Android device and only control access to corporate-owned apps and data.

Privacy protection:

- Multi-user accounts
- Device admin - permissions
- Per user\per app VPN

The next section describes methods and attacks against the Android operating system that enables attackers to gain access to data that should be protected.

3.3.4 Known attacks against Android

One of the goals of this thesis is to develop a model for a more secure mobile operating environment. In order to accomplish this a better understanding of existing vulnerabilities in mobile operating systems are required. This section looks at the security aspects of the Android operating system.

The number of mobile attacks has increased dramatically over the past years. Figure 10 shows the number of reported vulnerabilities for both Android and the iOS since 2010 until June 2016 as reported by the National Vulnerability Database (NVD) (National Institute of Standards and Technology, 2016).

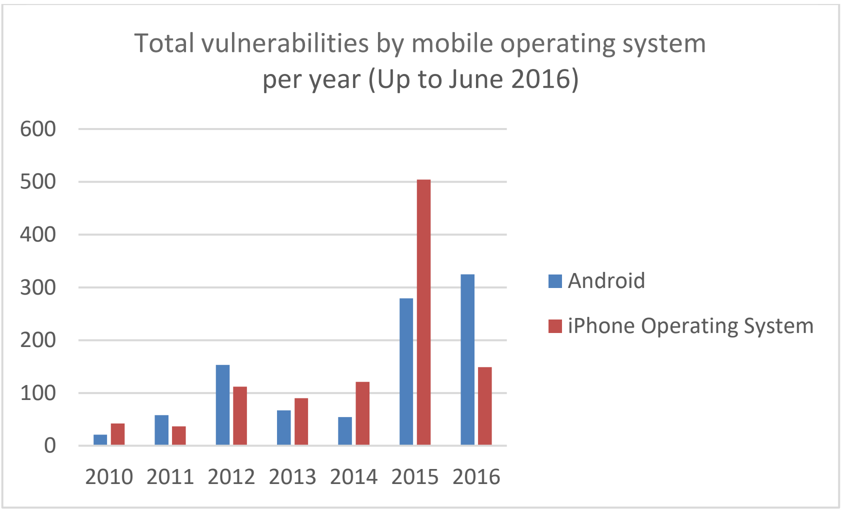


FIGURE 10: TOTAL VULNERABILITIES BY MOBILE OPERATING SYSTEM (NATIONAL INSTITUTE OF STANDARDS AND TECHNOLOGY, 2016)

An interesting phenomenon was the spike in vulnerabilities in iOS in 2015. There were more vulnerabilities listed for iOS in 2015 than there was for Android. On the other hand, 2016 showed that the total Android vulnerabilities increased above the iOS reported vulnerabilities

One method to look at attacks on Android is to group attack vectors in high-level attack surfaces (Drake et al., 2014). These attack surfaces are Remote Attack Surfaces, Local Attack Surfaces and Physical Attack Surfaces (Drake et al., 2014). Investigating these attack surfaces further allows the categorisation according to technical vulnerabilities and design vulnerabilities.

Technical vulnerabilities are vulnerabilities that exist because of a problem in the source code of the system whereas a design vulnerability is a vulnerability because of the way in which the system components interact or the way in which the system works.

Despite the technical vulnerabilities, certain design issues also create vulnerabilities. The design issues are categorised into five design issues that create vulnerabilities. The design categories are described in more detail below:

- **Section 3.3.4.1**. Android ecosystem
- **Section 3.3.4.2**. Code review and side loading
- **Section 3.3.4.3.** User dependent security approval
- **Section 3.3.4.4**. Privilege escalation
- **Section 3.3.4.5.** Rooting of devices

3.3.4.1 Android ecosystem

Google does not have full autonomy over the Android code base. The Android version running on a device is influenced and affected by four players. These four players are shown in Figure 11 and described in more detail below.

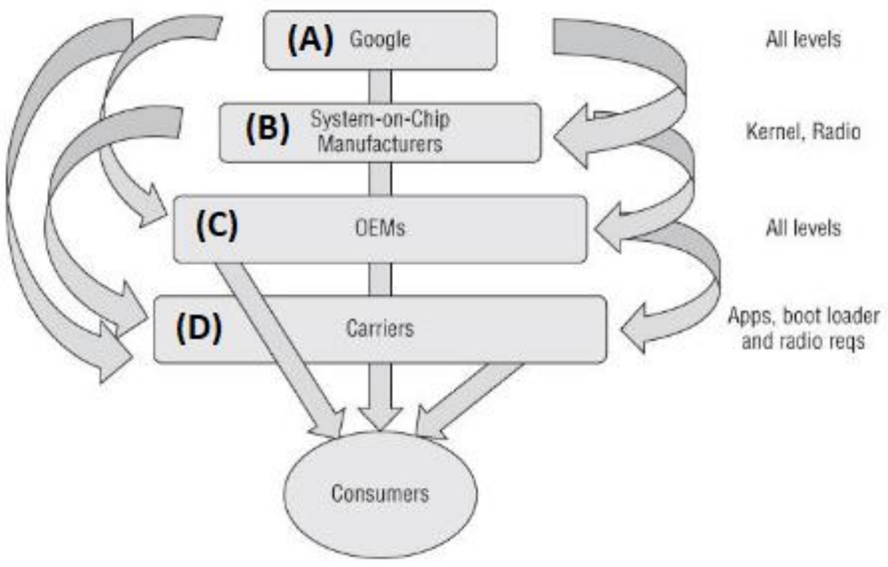


FIGURE 11: PLAYERS IN THE ANDROID CODE BASE (DRAKE ET AL., 2014)

- **Google** (A in Figure 11). Google is responsible for changing and affecting code on all the levels
- **System on-Chip (SoC)** (B in Figure 11). The System-on-Chip (SoC) manufacturers are licensed by ARM (ARM Ltd., 2016) to manufacture silicon chips with processor, memory and many other components on one piece of silicon. A list of these manufacturers is available from the ARM website[1]. The SoC manufacturers must modify the Linux kernel to ensure the Linux kernel is fully supported on their processors and have all the device drivers included to support the various components that are on the silicon (Drake et al., 2014).
- The **original equipment manufacturers** (OEM) (C in Figure 11) take the silicon from the various SoC vendors and develop their own handsets. These manufacturers modify the Android code base again to support their specific handsets. They may also modify the user interface and add more utilities into the Android operating system to give the end-user a better or more secure experience.
- **Carriers** (D in Figure 11) also modify Android. They don't just provide the infrastructure for delivering the voice or data, they sometimes also form agreements with OEMs to re-brand phones and deliver these handsets as subsidised contracts. To retain user loyalty, these handsets are sometimes locked on to the carriers' network, disabling the ability of a user to use the handset on another carrier's network. To enable all of this, the carriers also make changes to the code base of Android and are at the end of the day responsible for delivering over the air (OTA) patches to these handsets.

All of these players affect the code base of Android. This means that when a vulnerability is found in the Android operating system, a number of parties are involved to implement

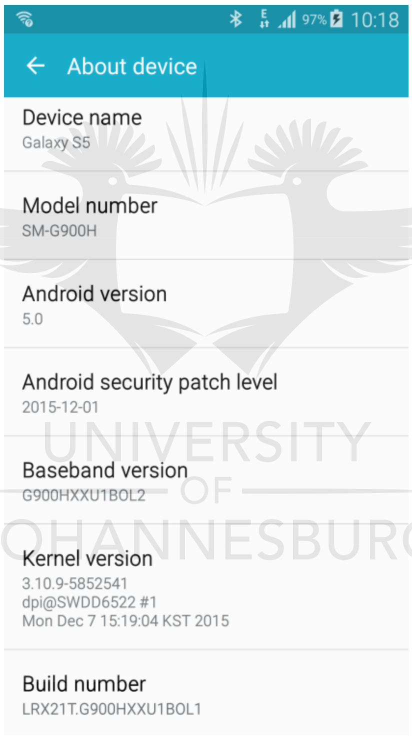
[1] http://www.arm.com/products/processors/licensees.php

and distribute the patch to Android handsets. In some cases, handsets never get patched, simply because someone in the Android ecosystem is not delivering the patch. ***(Shared security responsibility)***.

In the 2015 Android security report, Google admits that only 70.8% of handsets are eligible for monthly security patches (Google Inc., 2016). Of these 70.8% eligible handsets, the updates must still be sent to handsets before the patch is applied and this involves both OEMs and Carriers.

Figure 12 is a screenshot for Android running on a Samsung Galaxy S5 model phone taken on 18 July 2016. The Android version is supported for Android patches, but the patch level on the phone is dated December 2015, which means that any Android vulnerabilities that were patched since then have not been applied to the phone in question.

The patching of Android is thus a ***shared responsibility***.



Device name
Galaxy S5

Model number
SM-G900H

Android version
5.0

Android security patch level
2015-12-01

Baseband version
G900HXXU1BOL2

Kernel version
3.10.9-5852541
dpi@SWDD6522 #1
Mon Dec 7 15:19:04 KST 2015

Build number
LRX21T.G900HXXU1BOL1

FIGURE 12: ANDROID PATCH LEVEL (BY AUTHOR)

3.3.4.2 Code review and side loading

Google introduced Verify Apps in 2012 for new apps being deployed through the Google Play store (Google Inc., 2016). Verify Apps scan Android apps made available through the Google Play store to see if they are potentially harmful apps (PHA).

Google also searches the web for PHAs and include this in their Verify Apps cloud engine. When a user downloads an app from another source, the device can still display a warning to the user about the PHA. The unfortunate part of this is that users can still elect to ignore the warning and install the app.

The Android operating system allows users to install apps from third-party stores, which may not have the same levels of code review that Google has. The multiple sources of apps can be called an ***open app ecosystem***. ***(Open app ecosystem)***.

Android apps are packaged as files with an .apk extension. It is possible for an Android user to enable developer settings on their devices, which allows a user to install an app by just using the apk file that the user might have downloaded from any location. This is called sideloading.

Users resort to ***side loading*** when they want to gain access to apps not approved by Google, not distributed in their region or don't want to pay for the app. The danger of sideloading an app is that the app bypasses the Verify Apps mechanism on the cloud, which allows PHA to be installed on a device. ***(Side loading of apps)***.

3.3.4.3 User dependent security approval

Before an app is installed, the app displays a list of permissions that are requested by the app. Many people do not understand these Android permissions and just accept the permissions, even though there may be potentially dangerous permissions required by an app. An example of an app which requests a lot of unnecessary permissions is the Flash Keyboard app on the Google Play store (DotC United, 2016). The Flash Keyboard permissions include permissions like: Download files without notification, modify system settings and disable your screen lock. The problem with these permissions is that the app can update some functionality without it getting scrutinised by Google, it can change system settings and it can potentially create an insecure mobile device.

Prior to Android 6, users had to accept all the permissions or decline to install the app. From Android 6, apps must give users the ability to selectively allow specific permissions.

The problem with these permissions is still that it is dependent on the knowledge of the user to allow an app access to some areas on the mobile device. The Android operating system is dependent on the ***security knowledge of the user***. ***(User dependent security knowledge)***.

3.3.4.4 Privilege escalation

In an article that has been cited more than 335 times, Davi et al. (2011) demonstrate that the permission structures that apps gain can be abused when apps with fewer privileges use components from apps with higher privileges to gain access to content where the original app did not have permissions to. Figure 13 shows that an Application B has permissions in Application C, but because Application A has access to the component C_{B1} that it inherently also gains access to C_{C1} even though it does not have direct access to the components in Application C.

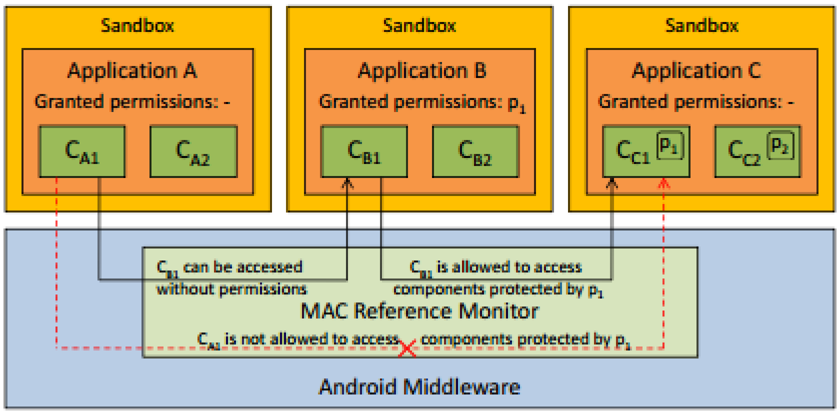


FIGURE 13: PRIVILEGE ESCALATION ATTACK IN ANDROID (DAVI ET AL., 2011)

The unfortunate part of the privilege escalation possibility is that it functions within the design parameter. The design parameters for the authentication mechanism has a serious ***design flaw***. ***(A flawed access control mechanism)***.

Since the article from Davi et al. there have been many articles relating to the mitigation of these types of attacks. One of the mitigations used to minimise these attacks is through the use of Verify Apps. Inherently though the vulnerability still exists in the Android architecture and it does not look as if this may be fixed soon.

3.3.4.5 Rooting of devices

Rooting a device means that the apps can run as root in the Linux operating system. This means the app gets permissions to break out of the sandbox and perform tasks not normally possible by other means. There are a number of apps that use root permissions to provide functionality not normally possible in the app. Examples of this functionality are functions that allow remote control of the Android device from a computer. All of these apps require root privileges to function properly. ***(Rooting of devices)***.

Attack methods and vulnerabilities:

- Shared security responsibility.
- Open app ecosystem.
- Side loading of apps.
- User dependent security knowledge.
- A flawed access control mechanism.
- Rooting of devices.

The Android operating system consists of many components that have evolved since the first version of Android. Android is an open development environment that allows for a lot of functionality, but it may be at the cost of security.

3.4 Summary of drivers

Table 3 provides a summary of the various drivers identified for the Neo model. The table depicts the various security aspects of both Apple iOS and Android. The table consists of five areas. Each of these areas correlates to specific sections of this chapter.

TABLE 3: IOS AND ANDROID SECURITY DRIVERS

Apple iOS	Android
Design methodology	
Application level isolation	Application level isolation
Isolation mechanisms	
File system boundaries	File system boundaries
Non-privileged process execution	Non-privileged process execution
Same user account	Separate user accounts
Controlled operating system interfaces	
Controlled processing	
Modern data and code execution protection	
Application communication	
Specialised access control mechanisms	Specialised access control mechanisms
Apps cannot start processes belonging to another app directly	Developers can use external storage to access data from other apps
	Developers control access to app processes
	Developers control access to app data
Privacy protection	
Operating system controlled sharing	Multi-user accounts
Per-app VPN	Device admin - permissions
App categorisation ensures controlled data sharing	Per-user\per-app VPN
Attack methods and vulnerabilities	
Injection attacks	Shared security responsibility
Web exposed components	Open app ecosystem
Use complex attacks to expose data	Side loading of apps
Jailbreaking	User dependent security knowledge
	A flawed access control mechanism
	Rooting of devices

It is clear both iOS and Android use an application level isolation model. This means that the operating systems isolate applications from each other. One of the major drivers for these mobile operating systems are to deliver a computing environment where users can install applications, without severely affecting the overall security of the operating system.

The drivers specified in Table 3 are further evaluated in Chapter 6 to determine the shape and requirements of the Neo model.

3.5 Conclusion

Since the Neo model concerns itself with a mobile operating environment, the isolation mechanisms used in existing mobile operating systems were evaluated.

Both Android and Apple iOS uses a sandbox model to isolate apps and their respective data from each other. Android has a lot more mechanisms and structures to facilitate the exchange of data between apps and allow one app to use components from within other apps. Apple iOS, on the other hand, uses a lot more stringent controls to exchange information between apps.

Android and Apple iOS have mechanisms that isolate private information from business information and gives corporate mobile device management systems the ability to control business apps and their respective data.

Both Android and Apple iOS suffers from vulnerabilities that are discovered in the code base of the operating systems, it seems however that Apple iOS is better equipped to patch the operating system than Android, which leaves many Android devices insecure when new vulnerabilities are exposed.

The isolation properties of both Android and iOS is important to ensure the protection of the operating system when an app misbehaves. Strong controls are necessary to ensure access to data and functionality of other apps does not lead to unauthorised access.

One of the biggest risks in both Android and iOS is that if one app with malicious intent is installed on the system, then it puts the whole system at risk. Users may inadvertently approve unnecessary permissions for the malicious app. The approved permissions may allow the malicious app to gather more information than what it is allowed.

The other challenge with today's mobile operating systems, is the implementation by vendors haven't fully explored and enabled a user to only use one device for both personal and business purposes. Even though the mobile hardware is getting more powerful, there hasn't been a mobile operating system implementation that has been adopted as a desktop replacement.

The isolation features and aspects of iOS and Android act as drivers for developing a model for a mobile operating environment. The Neo model uses the drivers identified in this chapter as input for decision-making purposes.

The two major mobile operating systems have both adopted a sandbox-like approach to isolate apps, as described in this chapter. The next chapter describes the isolation techniques and mechanisms in other technologies.

Introduction

- Chapter 1 Introduction

The end game

- Chapter 2 Model Overview

Literature study

- Chapter 3 Isolation properties in mobile operating systems (SCP)
- **Chapter 4 Isolation techniques in other technologies** (SCP)
- Chapter 5 An overview of peripheral security (MAP)

Bridge

- Chapter 6 The drivers for a new model

Building the Neo model

- Chapter 7 Overview of the Neo Model
- Chapter 8 The Secure Container Property
- Chapter 9 The Mutual Authentication Property
- Chapter 10 Container life cycle
- Chapter 11 Novel Implementations

Evaluation and summary

- Chapter 12 Potential implementation of Neo Model
- Chapter 13 Conclusion and future work

Chapter 4 ISOLATION TECHNIQUES IN OTHER TECHNOLOGIES (SCP)

4.1 Introduction

In the previous chapter, the isolation techniques used in the two major mobile operating systems were described. Android and iOS use different techniques to achieve isolation and isolation is used to protect apps and app data from other apps. The level of isolation implemented in Android and iOS is on the application level.

Virtual Machines provide a different level of isolation where the degree of abstraction is a lot stronger that is observed in the isolation techniques used by today's mobile operating systems.

This chapter describes how virtual machines are used in both desktop and mobile devices. The chapter first gives an overview of virtual machines and provides a reminder of how virtual machines function. The chapter then looks at two types of virtual machines that have received a lot of attention in the literature for security purposes. The two types of virtual machines that are described are:

- Hypervisor-based virtual machines and the other is called
- Container-based virtual machines.

The objective of describing and investigating these two types of virtual machines is to identify drivers that can be used as input to the Neo model. The focus area is to see how they isolate computing components, work with existing input/output peripherals and how communication between virtual machines are established if they are running on the same hardware.

When a significant driver is identified the same callout mechanism is used as in Chapter 3. These callouts are then summarised at the end of this chapter and used as input in Chapter 6.

This chapter is structured as follow:

- **Section 4.2** provides a brief overview of the different virtual machines technologies and architectures. This section acts as a quick reminder of how virtual machines function and describes the virtual machine landscape.
- **Section 4.3** concentrates on virtual machines that function using hypervisors. This section provides an overview of hypervisor-based virtual machines and describes a special implementation of a hypervisor-based virtual machine called Qubes-OS (Rutkowska, 2016).

- **Section 4.4** describes container-based virtual machines. This section describes container-based virtual machines with a specific focus on Cells (Dall et al., 2012).

4.2 Virtual Machines

The concept of virtual machines can be traced back to the original IBM VM/370. IBM used virtualisation technology to partition the mainframe into multiple systems (Creasy, 1981). This type of partitioning stayed in the realm of mainframes up to early 1990 when VMware introduced the system virtualisation on the x86 platform (Rosenblum, 2004).

To help set the scene for a discussion on virtual machines, is it important to agree on what type of virtual machine is under discussion. Figure 14 helps with the discussion and describes the taxonomy of virtual machines categories. The literature identifies two primary categories of virtual machines (Smith & Nair, 2005). These categories of virtual machines are process virtual machines and system virtual machines (The second layer from the top in Figure 14).

Process virtual machines are used to describe the runtime environment of interpretative programming languages, like the Java virtual machine or Microsoft's Common Language Infrastructure (Smith & Nair, 2005).

System virtual machines are used to describe the abstraction of hardware into separate virtual computers (Smith & Nair, 2005).

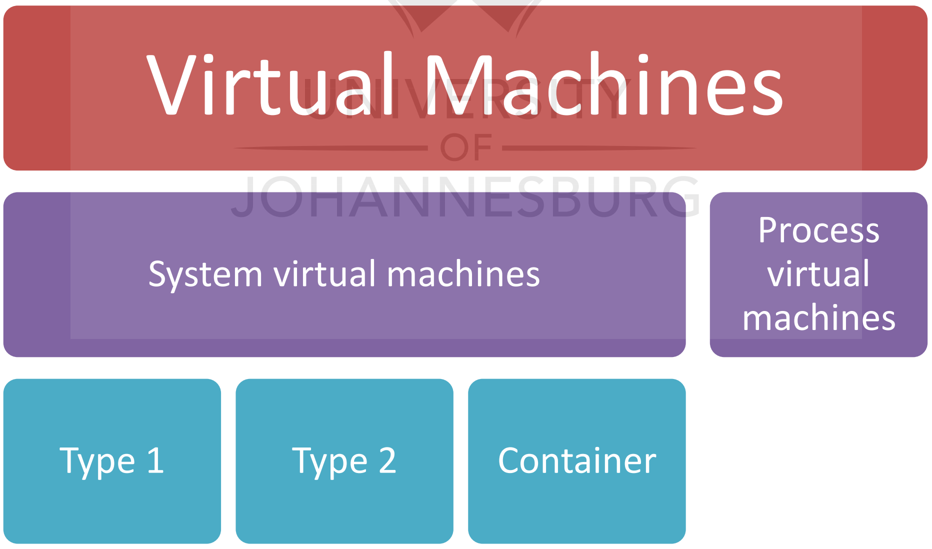


FIGURE 14: VIRTUAL MACHINE CATEGORIES (CREATED BY AUTHOR)

System virtual machines are further categorised into type 1 and type 2 hypervisors (Tanenbaum & Bos, 2015) and a third category called container-based virtual machines are further described. These system virtual machines are depicted at the lowest level in Figure 14.

Figure 15 shows a comparison between type 1, type 2 and container virtualisation. A type 1 hypervisor exists directly on the hardware, with the virtualised operating systems running on top of the hypervisor (a in Figure 15). A Type 2 hypervisor is implemented as a hypervisor that runs as an installed application on a host operating system (b in Figure 15).

The last category of virtual machines that are identified in this thesis and shown in Figure 14 is container-based virtualisation. Container virtualisation does not implement as strong isolation as what can be achieved with hypervisors (England & Manferdelli, 2006). Container virtualisation achieves virtualisation by cloning user-space instances while sharing the same kernel (Raho et al., 2015). This is shown as c in Figure 15. Container-based virtual machines do not host a guest operating system, instead, it makes a copy of some of the host operating system components into specific containers.

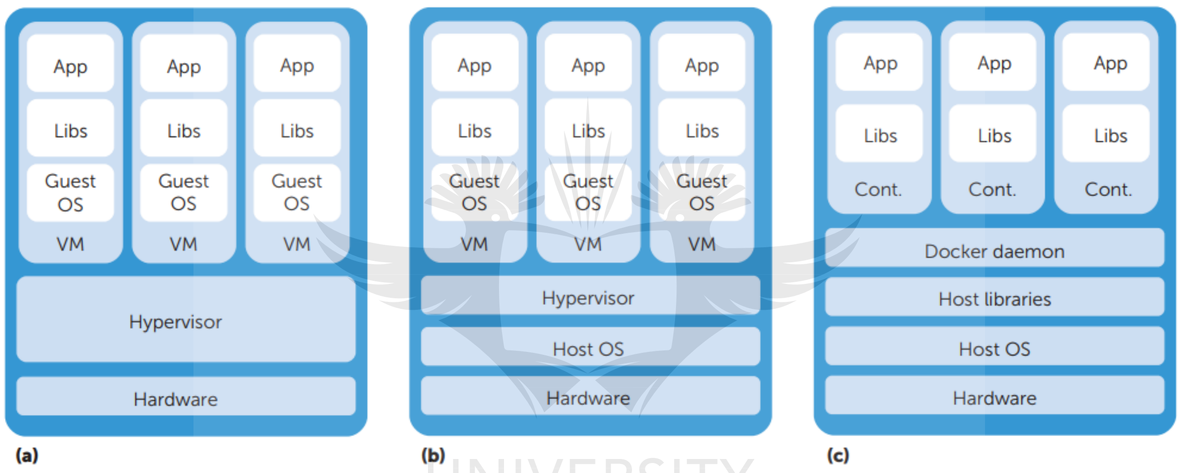


FIGURE 15: SYSTEM VIRTUALISATION TECHNIQUES (COMBE ET AL., 2016)

Each container is referred to as a virtual machine. There has been a number of operating systems implementing container-based virtual machines, some with more or fewer features than others (FreeBSD, n.d.) (Canonical Ltd., n.d.). This document focuses on the features and containers that are possible using Linux namespaces and is discussed in more detail in section 4.4 of this chapter.

This section provided a quick overview of virtual machines. Three system virtual machine implementation techniques were identified. An hypervisor based implementation is called Qubes-OS and is further described in the next section.

4.3 Hypervisor-Based Virtual Machines

A Hypervisor is a special layer in an operating system that allows a host operating system to present virtualised instances of hardware to be used by a guest operating system (Tanenbaum & Bos, 2015).

In general, there are two types of hypervisors, type 1 hypervisors and type 2 hypervisors. The easiest way to distinguish between type 1 hypervisors and type 2 hypervisors is in type 1 hypervisors, only the hypervisor software runs in the most privileged mode. Type 2 hypervisors are dependent on a host operating system managing the scheduling and resource allocation of guest virtual machines running on the host operating system (Tanenbaum & Bos, 2015).

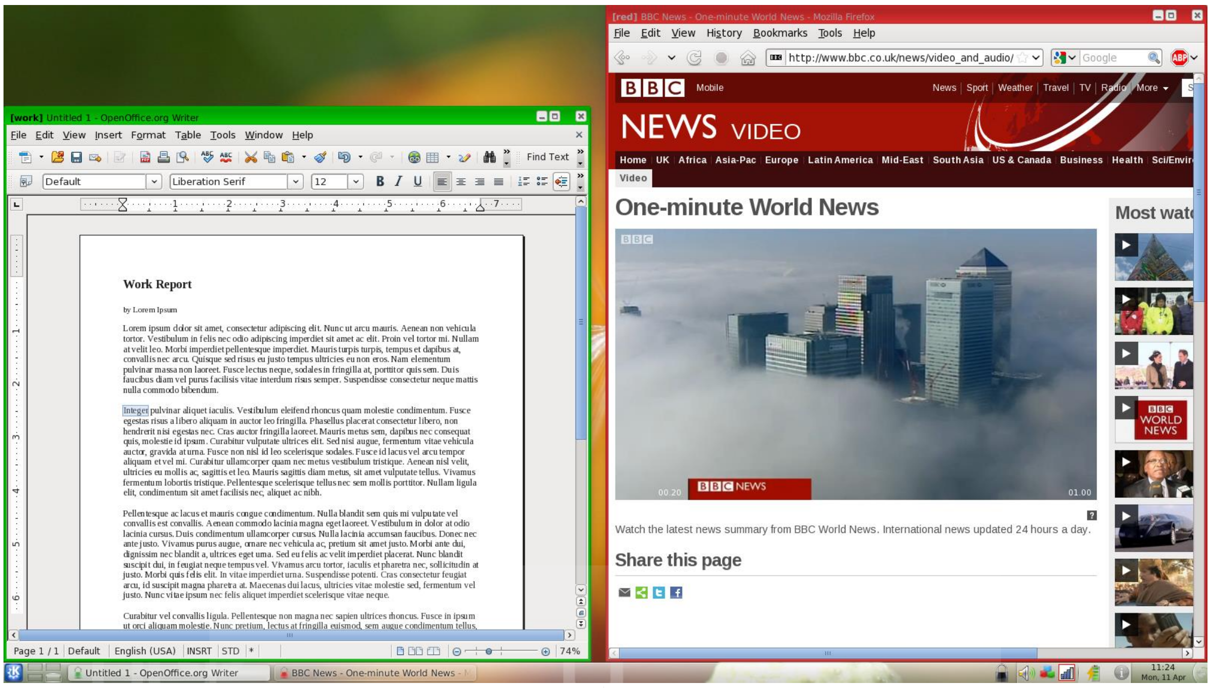
Even though the focus of this study is on mobile operating environments, commercial implementations of type 1 or even type 2 hypervisors on mobile devices have been limited. Furthermore, the focus of this section is specifically on the implementation of hypervisor virtualisation to enable a more secure computing environment.

With the above focus in mind, Qubes-OS (Rutkowska, 2016) was selected as a system, using hypervisors to ensure a more secure computing environment.

4.3.1 Qubes-OS

At the time of writing this, Qubes-OS was on version 3.1 and is freely available. Qubes-OS is a Linux-based operating system built on the Xen hypervisor (The Linux Foundation, 2013) (Rutkowska, 2016). Qubes-OS architecture is based on security through isolation. This architecture is in line with this thesis and is further explored to determine the features and architectural decisions that enforce security in Qubes-OS. ***(Hypervisor level isolation)***.

Figure 16 is a screenshot of Qubes-OS with two AppVMs running. One AppVM, with a green border(on the left), is a work AppVM used for work purposes and currently has a work report open in OpenOffice. The other AppVM, with a red border (on the right), is used only for general web browsing. The red AppVM is totally isolated from the green AppVM. Even in the event that the red AppVM may become compromised the green AppVM stays isolated.


[work] Untitled 1 - OpenOffice.org Writer
File Edit View Insert Format Table Tools Window Help
Find Text
Default
Liberation Serif
12
Page 1 / 1 Default English (USA) INSRT STD *
74%
[red] BBC News - One-minute World News - Mozilla Firefox
File Edit View History Bookmarks Tools Help
http://www.bbc.co.uk/news/video_and_audio/
Google
BBC Mobile
News Sport Weather Travel TV Radio More
NEWS VIDEO
Home UK Africa Asia-Pac Europe Latin America Mid-East South Asia US & Canada Business Health Sci/Envir
Video
Most wat
BBC WORLD NEWS
Untitled 1 - OpenOffice.org Writer
BBC News - One-minute World News - M
11:24 Mon, 11 Apr

Work Report

by Lorem Ipsum

Lorem ipsum dolor sit amet, consectetur adipiscing elit. Nunc ut arcu mauris. Aenean non vehicula tortor. Vestibulum in felis nec odio adipiscing imperdiet sit amet ac elit. Proin vel tortor mi. Nullam at velit leo. Morbi imperdiet pellentesque imperdiet. Mauris turpis turpis, tempus et dapibus at, convallis nec arcu. Quisque sed risus eu justo tempus ultricies eu non eros. Nam elementum pulvinar massa non laoreet. Fusce lectus neque, sodales in fringilla at, porttitor quis sem. Duis faucibus diam vel purus facilisis vitae interdum risus semper. Suspendisse consectetur neque mattis nulla commodo bibendum.

Integer pulvinar aliquet iaculis. Vestibulum eleifend rhoncus quam molestie condimentum. Fusce egestas risus a libero aliquam in auctor leo fringilla. Phasellus placerat consectetur libero, non hendrerit nisi egestas nec. Cras auctor fringilla laoreet. Mauris metus sem, dapibus nec consequat quis, molestie id ipsum. Curabitur vulputate ultrices elit. Sed nisi augue, fermentum vitae vehicula auctor, gravida at urna. Fusce non nisl id leo scelerisque sodales. Fusce id lacus vel arcu tempor aliquam et vel mi. Curabitur ullamcorper quam nec metus vestibulum tristique. Aenean nisl velit, ultricies eu mollis ac, sagittis et leo. Mauris sagittis diam metus, sit amet vulputate tellus. Vivamus fermentum lobortis tristique. Pellentesque scelerisque tellus nec sem mollis porttitor. Nullam ligula elit, condimentum sit amet facilisis nec, aliquet ac nibh.

Pellentesque ac lacus et mauris congue condimentum. Nulla blandit sem quis mi vulputate vel convallis est convallis. Aenean commodo lacinia magna eget laoreet. Vestibulum in dolor at odio lacinia cursus. Duis condimentum ullamcorper cursus. Nulla lacinia accumsan faucibus. Donec nec ante justo. Vivamus purus augue, ornare nec vehicula ac, pretium sit amet justo. Morbi ante dui, dignissim nec blandit a, ultrices eget urna. Sed eu felis ac velit imperdiet placerat. Nunc blandit suscipit dui, in feugiat neque tempus vel. Vivamus arcu tortor, iaculis et pharetra nec, sollicitudin at justo. Morbi quis felis elit. In vitae imperdiet urna. Suspendisse potenti. Cras consectetur feugiat arcu, id suscipit magna pharetra at. Maecenas dui lacus, ultricies vitae molestie sed, fermentum vel justo. Nunc vitae ipsum nec felis aliquet imperdiet scelerisque vitae neque.

Curabitur vel convallis ligula. Pellentesque non magna nec sapien ultrices rhoncus. Fusce in ipsum ut orci aliquam molestie. Nunc pretium, lectus at fringilla euismod, sem augue condimentum tellus,

One-minute World News

Watch the latest news summary from BBC World News. International news updated 24 hours a day.

Share this page

FIGURE 16: QUBES-OS AT WORK

This section on Qubes-OS is organised as follows:

- In **section 4.3.1.1** the design requirements of Qubes-OS is described. This is important in order to understand the problem Qubes-OS is trying to solve.
- In **section 4.3.1.2** the architecture of the system is described. This highlights important components and describes the system in a little more detail.
- Section **4.3.1.3** evaluates the six security requirements, mentioned in the Introduction of this chapter, with the security features and architecture of Cubes-OS. The outcome of this helps to establish the security credibility of Cubes-OS.

Qubes-OS was designed according to specific requirements. These requirements are discussed in the following section.

4.3.1.1 Design requirements

The words "security domain" used in this section describes different isolation environments. An application running in one security domain cannot gain access to the data or systems in another security domain.

Even though there is no publicly available list of design requirements the following security requirements are inferred using the information from the Qubes-OS website (Rutkowska, 2016):

- **Provide an environment where users can run their desktop applications in a secure and isolated environment.** Qubes-OS should be focussed on providing a secure environment for desktop users to run applications in, under various conditions.

- **Security through isolation**. Applications on Qubes-OS run in specific domains. Each domain is isolated from another domain through various architectural components, ranging from isolated storage locations, isolated network devices and separate window managers (Mansfield-Devine, 2010) (Rutkowska, 2016).
- **Allow multiple security domains**. The user can decide the number of security domains that are required. These domains can be categorised according to the requirements of the user (Rutkowska, 2016).
- **Seamless GUI integration.** The system should be easy to use for the user so that the user can easily move from an application in one domain to an application in another domain.
- **No security isolation of applications running in the same domain**. Data, corruption, exposed vulnerabilities or bugs of an application in one domain can impact any other application running in the same domain.
- **A compromised domain should not affect any other domain**. A compromised application or component in one domain should not be able to automatically compromise the integrity of another domain.
- **Secure clipboard, copy and paste between domains**. Data copied onto the clipboard should be made available to an application in another domain, using a secure process that cannot be used to launch attacks to another domain.
- **Secure file transfer between domains**. The user should have the ability to copy a file from one domain to another without it affecting the integrity of either system. The mechanism to handle the copying of files should also ensure that no other domain can see the data as it is being transferred.
- **Isolated network domains with advanced firewalling and proxy**. Network communication to and from domains should be controlled on a per domain basis and an all domains should be protected from attacks from the network.
- **Disposable virtual machines**. The user should have the ability to use a domain temporarily and then dispose of the data without having to worry about the data being available after the domain has been disposed of.

To implement the above security design requirements, the designers and authors of Qubes-OS implemented the system based on Linux virtual machines running on a Xen hypervisor.

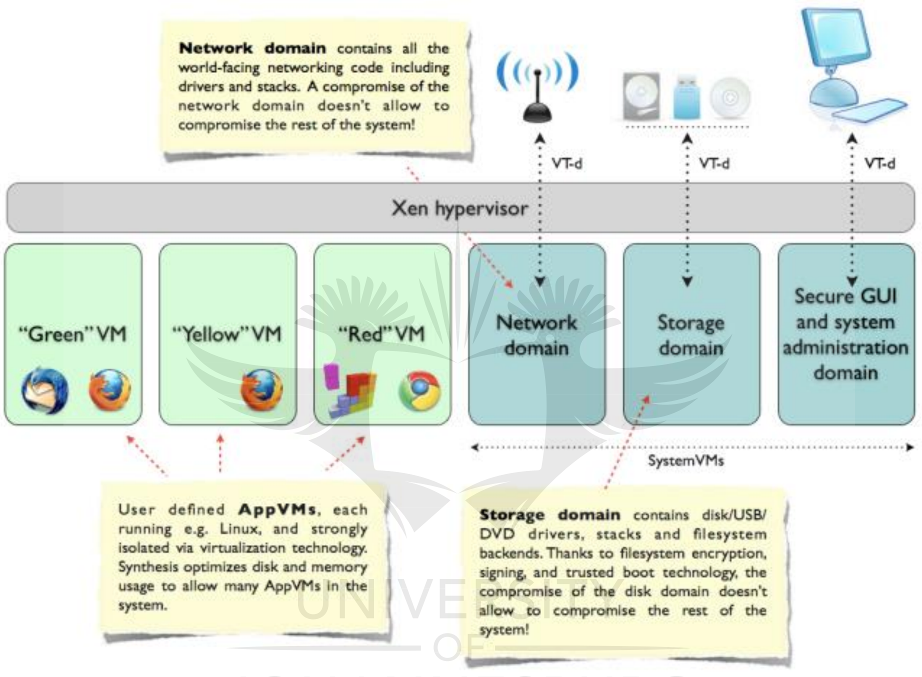
4.3.1.2 Architecture

The architecture of Qubes-OS consists of a number of components. Figure 17 shows an overview of the Qubes-OS architecture. At the top of the diagram are a few hardware devices, depicted as storage devices, a keyboard and screen and an antenna. Below the hardware is the Xen hypervisor. Below the hypervisor is a number of virtual machines. The virtual machines are categorised as either SystemVMs or not-SystemVMs. Qubes-OS isolate operating systems using hypervisor-level isolation. ***(Hypervisor-level isolation).***

Design methodology:

- Hypervisor level isolation

Applications run on virtual machines called domains. These virtual machine domains are known as AppVM domains. Each AppVM runs either a para-virtualised (The guest virtual machine is specially adapted to run on the hypervisor) Linux or a fully virtualised instance of Linux or Microsoft Windows (No changes are needed on the guest virtual machine in order for it to run on the hypervisor).


Network domain contains all the world-facing networking code including drivers and stacks. A compromise of the network domain doesn't allow to compromise the rest of the system!

User defined **AppVMs**, each running e.g. Linux, and strongly isolated via virtualization technology. Synthesis optimizes disk and memory usage to allow many AppVMs in the system.

Storage domain contains disk/USB/DVD drivers, stacks and filesystem backends. Thanks to filesystem encryption, signing, and trusted boot technology, the compromise of the disk domain doesn't allow to compromise the rest of the system!

FIGURE 17: HIGH-LEVEL QUBES-OS ARCHITECTURE (RUTKOWSKA & WOJTCZUK, 2010)

The AppVMs are based on a read-only template of a system, but with a copy on write (COW) file system when a user creates or edits a file in an AppVM. The template\COW implementation ensures that Linux based AppVMs are very small, even if there are many of them since most of the file system stays read-only and is shared between different domains based on the same template. Figure 18 shows two AppVMs, that uses a common appvm.img template as the base of the AppVM operating system. Any files created or modified are encrypted with an AppVM specific key and stored in a separate file only accessible by the AppVM. ***(Template based operating environments)***. ***(File system efficiency using copy on write mechanisms)***. ***(Encrypted virtual machine images)***.

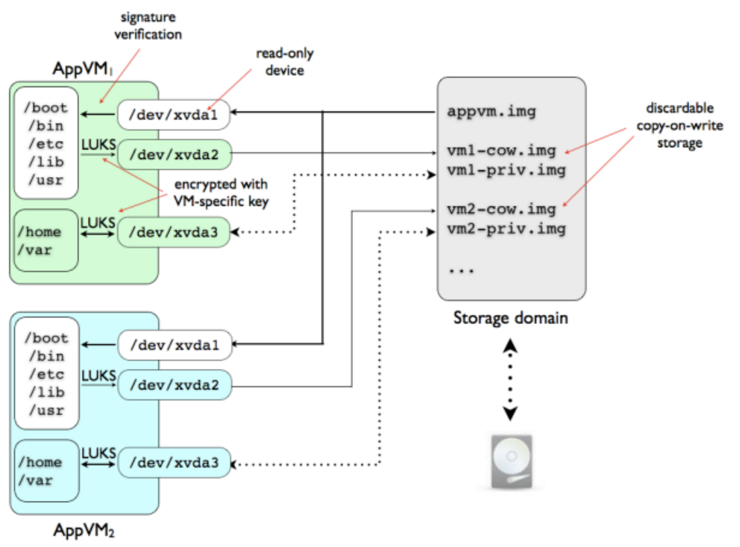


FIGURE 18: THE SECURE FILE SYSTEM SHARING MECHANISM (ROSENBLUM & GARFINKEL, 2005)

File system deployment methodology:

- Template based operating environments.
- File system efficiency using copy on write mechanisms.

Another special domain is called the Network domain. The Network domain is the domain in which the hardware drivers are installed and connected to the physical network interface cards. Connected to the Network domain is usually a special domain, called the Firewall domain, which acts as a system-wide firewall for all the AppVMs. Each AppVM is then connected to the Firewall domain. ***(Application virtual machines has no direct network access)***. ***(Strict network access control)***. ***(No direct network access to the management environment)***.

Network access security:

- Application virtual machines has no direct network access.
- Strict network access control

The Storage domain is used for USB storage devices and managing the file system of the AppVMs. The primary focus of the domain is to ensure the secure boot and file management of Dom0 and other AppVMs. ***(Removable storage has no direct access to operating environment files)***.

Operating environment confidentiality:

- Encrypted virtual machine images
- Removable storage has no direct access to operating environment files.

The secure GUI (Graphical User Interface) and system administration domain are used to manage the various user-defined AppVMs. In Xen terms, this is the Dom0 domain. It starts, stops and changes the configuration of the various AppVMs. It also hosts an AppViewer component that is used to display the GUI of a specific AppVM to the shared desktop. ***(User interface distinction between different application virtual machines)***.

Protected management environment:

- No direct network access to the management environment.
- A management environment controls application virtual machines.
- User interface distinction between different application virtual machines.

The last architectural aspects worth mentioning is the inter-VM components. These components allow AppVMs to exchange data in a secure, but limited way. The inter-VM components are: ***(Inter-domain communication occurs through controlled interfaces).***

- **RPC Framework**. The Remote Procedure Calls (RPC) framework in Qubes-OS allows RPC commands to be sent from Dom0 to another AppVM. It is not possible to use RPC between AppVMs. This framework also enforces RPC policies. The RPC policies govern the RPC activity allowed or denied per AppVM. These policies can be modified by the user of the system according to their requirements. The RPC framework allows Dom0 to start applications in the AppVM when a user selects the application from the menu (Rutkowska, 2018).
- **Secure File Copy**. Qubes-OS allows a user of one AppVM to select a file and then copy it to another file. It should be noted that the action always starts with the original file. Qubes-OS creates a shared memory device between the two AppVMs that assists with the file transfer. The destination file is always in a special folder on the destination AppVM, from where the user can move the file to a final directory. Secure File Copy ensures only the two AppVMs in question takes part in the communication and no other AppVM can access the shared memory (Rutkowska, 2018).
- **Secure Clipboard**. Secure Clipboard is a function of the secure GUI. It assists the transferring of data through a "copy" command into the clipboard, from where it will then be possible for a user to "paste" the data from the clipboard into another

AppVM. AppVMs cannot pull information from the clipboard, it has to be initiated by the user to specifically select copy and then paste (Rutkowska, 2018).

Operating environment communication:

- Inter-domain communication occurs through controlled interfaces.

There are a number of architecture components in Qubes-OS that ensures AppVM isolation and ensures system integrity. The architecture isolates the data and applications from other AppVMs so that one AppVM cannot gain access to data in another AppVM, or a compromised AppVM cannot gain access to the data or applications in another AppVM or in the Dom0 domain. The next section evaluates Qubes-OS as a secure operating system.

4.3.1.3 Security requirement evaluation

Qubes-OS is designed to be a secure operating system. The isolation aspects are evaluated using security requirements identified by Resehtova, Karhunen, Nyman and Asokan (2014). The section concludes by making some general observations regarding the applicability of Qubes-OS to solve the problem mentioned in Chapter 1 section 1.2.

Resehtova et al. (2014) identified six security requirements that must be fulfilled to strengthen the security of isolation mechanisms. The applicability of the six requirements to the Qubes-OS is described below:

- **Separation of processes.** Each AppVM runs as a separate guest operating system. Not only is each process inside the same virtual machine isolated using virtual memory, each operating system is also separated from other operating systems. It is the job of the hypervisor to isolate the various operating systems from each other. Qubes-OS ensures processes in one AppVM does not have any access to processes in another AppVM.
- **File system isolation.** The different AppVMs uses a VM template, to create the basic file system of the AppVM. The running AppVM minimises the space required for multiple operating systems, by using the concept of a copy-on-write (COW) to minimise duplication of files. Furthermore, the Dom0 filesystem is encrypted and only decrypted during operating system boot, using keys stored in the Trusted Platform Module (TPM). Once Dom0 has been decrypted and mounted, the individual AppVMs can be mounted using separate encryption keys.
- **Device isolation.** Device access from the AppVMs happens using different mechanisms. Network communication happens through virtual Ethernet devices. Screen access occurs through a dummy graphics driver, loaded per AppVM. Qubes-OS also implements the concept of a driver domain, which is a separate domain that hosts device drivers and is separate from Dom0. Qubes-OS, however,

does support Intel VT-d, which allows a virtual machine direct access to a hardware device. The advantage of Intel VT-d is that Intel VT-d allows quicker responses from hardware to the virtual machine while ensuring no other virtual machine to access the device. It also protects direct memory access (DMA) through remapping ensuring no other virtual machine can capture data transfer using DMA.

- **Interprocess communication (IPC) isolation**. Qubes-OS use specific RPC channels to transfer information between Dom0 and AppVMs. The RPC channels are controlled using strict policies defined on the Dom0 domain.
- **Network isolation**. As already mentioned. Each AppVM has a separate virtualised network interface, with its own IP address. Access to and from the real network and the AppVMs occur through a proxy and firewall VM.
- **Resource management**. Since Qubes-OS is aimed as a desktop operating system, the standard configuration allows one AppVM to misuse resources assigned to it, potentially causing other AppVMs from starving. The user of the Qubes-OS, however, is always in control and can stop this type of AppVM. It is also possible to change the default configuration of an AppVM, limiting the AppVM from utilising all resources.

Qubes-OS is a desktop operating system, designed and architected to provide a highly secure operating environment. Qubes-OS uses Linux as the basis of its Dom0 controlling domain, but allow different types of guest operating systems to be installed. Qubes-OS, however, has not been built as a mobile operating system environment. Support for small screens or mobile telephony has not been included in the operating system. Qubes-OS also does not provide special security or authorisation of I/O peripherals. Qubes-OS does not have any capability for multiple users using the system, nor does it allow owners of data to remotely manage and control operating environments.

Qubes-OS provides a very secure isolation of operating environments. It allows a user to create and group environments based on their trust level. Qubes-OS is not built or designed to be a mobile operating system. It does not support multiple users or allow companies to control their own applications or data. For this reason, Qubes-OS is not suited as a solution to the problem addressed in section 1.2.

This section described hypervisor-based virtual machines. The next section provides background and a discussion about the isolation mechanisms provided by container-based virtual machines.

4.4 Container Based Virtual Machines

Container-based virtual machines use a lightweight mechanism in the Linux operating system, to create isolated computing environments. Container-based virtual machines do not use a hypervisor at all, instead, it uses special mechanisms on a normal Linux operating system to create isolated computing environments. A number of systems make use of container based virtual machines. Some of these systems include: Docker (Docker Inc., 2018), Containerd (The containerd authors, 2018) and Kubernetes (The Kubernetes Authors, 2018)

This section is organised as follow:

- The mechanisms that enable the isolation of processes are discussed under **section 4.4.1**.
- **Section 4.4.2** describes a solution called Cells that implement Linux containers to run multiple mobile operating systems on one device.

The next section describes the mechanisms available in Linux that enable Linux to isolate processes from other processes.

4.4.1 Linux Containers

A Linux container is also known as operating system level virtualisation (Calarco & Casoni, 2013). Container virtualisation does not run multiple operating systems, instead, it creates isolated operating environments inside the existing operating system. Container virtualisation uses a Linux feature, called Namespaces (Kerrisk, 2010) and a number of other features to create isolated operating environments. ***(Operating system level isolation)***.

Design methodology:

- Operating system level isolation

A namespace creates an abstraction level that isolates a global resource from all processes except those in the same namespace. A global resource is a resource used by an operating system that is normally available to any process running on the operating system. Namespaces ensure that a clone of a global resource is made and makes it only available to the processes in the same namespace.

Seven namespaces assist in creating isolation. The seven namespaces are (Kerrisk, 2010):

- **Cgroup namespaces**, or control group namespaces, isolate and control directory structures, and processing resources from one namespace to another. Cgroup namespaces allow processing resources, such as CPU time or memory allocation to be strictly controlled.
- **IPC namespaces**, ring fence interprocess communication mechanisms. IPC mechanisms allow processes to share data with each other. The IPC namespace

ensures that data sharing between processes can only happen between processes running in the same namespace.

- **Network namespaces**. The network namespace creates a namespace specific network interface. Only processes in the same namespace can use the namespace specific network interface. The network namespace ensures network traffic is routed to the correct namespace and does not allow processes in one namespace to access network traffic destined or originating from another namespace. ***(Network traffic to and from one namespace is not accessible by another namespace)***.

Network access security:

- Network traffic to and from one namespace is not accessible by another namespace.

- **Mount namespaces**. The Linux operating system uses the concept of mounting storage devices into mount points, which is effectively just a directory. Mount namespaces group mount points together into a specific namespace. The mount namespace ensures that a mounted storage device is only available to processes in the same namespace. This ensures storage devices connected to the main Linux operating system, is not automatically available to processes running in a separate namespace. ***(Removable storage is only accessible from one namespace at a time)***.

Operating environment confidentiality:

- Removable storage is only accessible from one namespace at a time.

- **PID namespaces**. Each process running on the Linux operating system has a unique process identifier. PID namespaces allow processes running in separate namespaces to have the same process identifiers. Without PID namespaces, processes may determine PID of other processes on the same system, but with PID namespaces, processes only know of processes in the same namespace. ***(Applications within the same namespace can communicate without restrictions)***. ***(Applications in different namespaces cannot communicate with each other)***.

Operating environment communication:

- Applications within the same namespace can communicate without restrictions.
- Applications in different namespaces cannot communicate with each other.

- **User namespaces** allow user identities to be isolated between different namespaces. A process always executes in the context of a specific user identifier on a Linux system. User namespaces ensure that the same process can execute in two different namespaces, but with different user identifiers. The user namespace allows processes to run different security permissions depending on which namespace they run in. ***(Applications in different namespaces cannot communicate with each other)***.
- **UTS namespaces** allow an administrator to give each namespace a hostname different from the main Linux operating system. Processes in the one namespace will think they are running on a host with a hostname different from processes in another namespace.

A number of Linux container implementations, like Docker or Linux Containers (LXC), uses the namespace features to build truly isolated operating environments, without having to make use of hypervisor-based technologies. Their ability to make use of the namespaces allow a much more lightweight implementation. Linux containers usually produce much better performance results than hypervisor-based virtualisation (Xavier et al., 2013). The performance factor makes Linux containers more attractive to implement on mobile processors, since it does not have such a big performance impact, and as such is a lot friendlier towards battery usage. ***(Low impact resource utilisation).***

Processing performance:

- Low impact resource utilisation.

Some of the issues that still exist with Linux Containers include problems with device abstraction, hot plugging of I/O peripherals and random number generators (Reshetova et al., 2014). These issues create challenges for fully isolated mobile operating systems.

A number of researchers have taken the Linux container concept, expanded it and implemented a system that supports multiple mobile operating systems. The system is called Cells and is discussed in the next section.

4.4.2 Cells

In 2011, a virtual mobile smartphone architecture was introduced as Cells (Andrus et al., 2011). The Cells architecture allows multiple instances of an Android operating system to run on one device. Cells, later on, evolved into a commercial product called Cellrox (Cellrox Ltd., 2015).

This section describes the cells architecture and its use of isolation mechanisms. The architecture is further evaluated as a solution to the problem described in Chapter 1.

Cells use the principle of Linux namespaces to isolate separate operating environments for each of its virtual phones. The namespace isolation creates user-level operating environments that host the full Android userspace. Inside each of the Android user spaces is the full Android (What was then called) Dalvik virtual machines. Each Dalvik VM provides a sandbox environment inside each virtual phone that runs one application. The Dalvik VM is the same as in a normal Android operating system, as described in Chapter 3 section 3.3.

Figure 19 describes the Cells architecture. The architecture consists of a Linux kernel component with a number of virtual phones, interfacing with the Linux kernel. The Linux kernel is depicted as the light grey block at the bottom of the diagram. In the Linux kernel are a number of device drivers, modified device drivers and namespace implementation. The namespace implementation in Cells uses three mechanisms. The three mechanisms are (Dall et al., 2012):

- **Device driver wrapper**. A device driver wrapper creates a virtual device exposed to only the active virtual phone. The device driver wrapper multiplexes communications between the virtual phones and the actual device driver. An example of the device driver wrapper is the virtual framebuffer device that sends and receives screen information from the active virtual phone.
- **Namespace-aware device subsystem**. In Linux, input devices interact with a device drivers, which goes through a device subsystem, that eventually interacts with event handlers that notifies processes of certain events that occurred. These events can be mouse movements, keyboard input, GPS coordinates and many more. Cells modify the device subsystem so that the event handlers that notify processes of events taking place to become namespace aware. This means the event handlers may only notify processes running in specific namespaces instead of all the processes that may be listening on the system. The namespace aware subsystem ensures processes in certain virtual phones, instead of everyone, can only read input from devices.
- **Namespace-aware device drivers**. A namespace-aware device driver is a modified device driver to be namespace aware. Android consists of a number of special device drivers running in the kernel of the operating system. An example

of such a device driver is the Binder device driver. The binder device driver is responsible to ensure different Dalvik VMs can communicate with each other when required. These device drivers have been modified so that instead of them communicating with each Dalvik VM running on all the Virtual Phones, communications between Dalvik VMs are now namespace bound. This means Dalvik VMs only communicate with other Dalvik VMs in the same Virtual Phone.

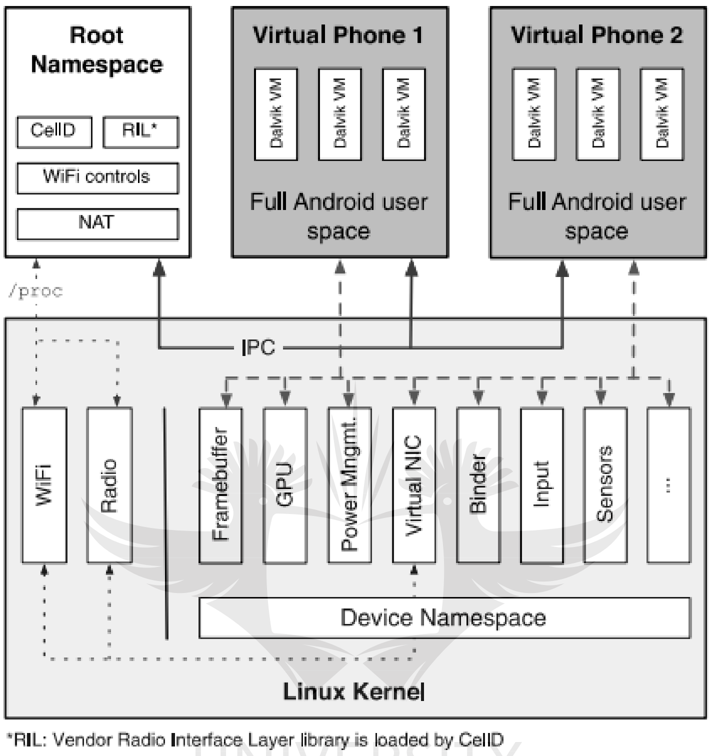


FIGURE 19: CELLS ARCHITECTURE (DALL ET AL., 2012)

The isolation model in Cells accomplished a well-defined isolation of various virtual phones. Each virtual phone is isolated from other virtual phones. Reshetova et al. (2014) comment that Cells provide mechanisms to achieve a high level of isolation. The Cells architecture evolved into a commercial product with tight integration with mobile device management that enables companies to control their own virtual phones.

The next section summarises the drivers highlighted in this chapter.

4.5 Summary of drivers

The ability of hypervisor-based virtual machines and container-based virtual machines to provide security in isolated environments is an important factor for the design of the secure isolated operating environment. The secure operating environments assist in defining the secure container property, required by the Neo model.

This section summarises the features and concepts of hypervisor-based virtual machines and container-based virtual machines that should be considered when designing the Neo model.

Table 4 summarises the callouts identified throughout this chapter. It summarises important security and design aspects of the hypervisor and container-based virtual machines.

TABLE 4: HYPERVISOR AND CONTAINER BASED SECURITY DRIVERS.

Hypervisor	Container
Design methodology	
Hypervisor level isolation	Operating system level isolation
File system deployment methodology	
Template based operating environments	
File system efficiency using copy on write mechanisms	
Operating environment confidentiality	
Encrypted virtual machine images.	
Removable storage has no direct access to the operating environment.	Removable storage is only accessible from one namespace at a time.
Network access security	
Application virtual machines have no direct network access.	Network traffic to and from one namespace is not accessible by another namespace.
Strict network access control.	
Protected management environment	
No direct network access to the management environment.	
A management environment controls application virtual machines	
User interface distinction between different application virtual machines	
Operating environment communication	
Inter-domain communication occurs through controlled interfaces.	Applications within the same namespace can communicate without restrictions.
	Applications in different namespaces cannot communicate with each other.
Processing performance	
	Low impact resource utilisation

Container-based virtual machines provide a different level of isolation than hypervisor-based isolation. Because of the higher level of isolation in container-based virtual machines, container-based virtual machines have a lower performance impact on host resources.

The management environment in hypervisor-based isolation is more isolated from the various virtual machine operating environments than container-based virtual machines.

The stronger isolation in hypervisor virtualisation can provide better security controls than in container virtualisation.

The next section comments on the solutions described in the chapter.

4.6 Conclusion

Chapter 1 describes a problem with personal and business use of mobile devices. The chapter describes the problem where a user uses multiple devices for both personal and business purposes and then may use a cloud-based solution to ensure the data is synchronised between devices.

The focus of this study is to see if a mobile operating system model can be established that minimises the reliance on multiple devices. This means that one device is used for both business and personal purposes. The challenge with using just one device is a problem of data isolation and data control. The problem however also extends to the processing capabilities of today's mobile devices and their ability to interact with various I/O peripherals.

This chapter described various isolation technologies that can be used to help solve the problem of isolating business and personal data when only one device is used. The technologies described uses virtual machines to establish isolation. This chapter distinguishes between hypervisor-based virtualisation and container-based virtualisation.

The first virtual machine solution that was described, was a hypervisor-based solution called, Qubes-OS. Qubes-OS uses Xen as a hypervisor solution. It creates a Dom0 domain, that manages the whole system, with various AppVMs providing operating system environments, each running its own operating system copy. Qubes-OS is a highly secure operating system but does not provide the user of the device with the ability to support multiple user accounts. Furthermore, Qubes-OS is designed and built for a desktop computer and even if ported to mobile hardware, may not run well on battery operated devices.

The second virtual machine solution described was Cells. The Cells architecture makes use of container-based virtualisation. Linux containers use Linux mechanisms to create lightweight isolation for Linux processes. Linux containers allow processes to be isolated from other processes so that processes in one namespace cannot access resources belonging to another namespace. Linux namespaces provide the mechanisms for creating isolated processing containers but lacks a number of components to support isolated containers for mobile devices.

Cells expand the namespace capabilities of Linux to successfully run multiple instances of an Android operating system. Apps in one virtual phone have no access to data or apps running on another virtual phone. Cells is a good example of an architecture that allows a user to use virtual phones for various purposes. The challenge with Cells, which this study tries to extend, is the ability of Cells to securely interface with different I/O peripherals and provide data owners with the tools to control access to containers where they own the data and applications in the container.

This chapter described a number of mechanisms that isolates operating environments in computing devices, using virtualisation as a basis. Some mechanisms work really well on desktop computers, while other solutions proved that isolation mechanisms might also be implemented on mobile devices.

The solutions described in this chapter rely on existing mechanisms to support the security of I/O peripherals. None of the solutions makes special mention of mutual authentication of I/O peripherals. They also do not have special authorisation mechanisms that control access between the I/O peripheral and specific container. The next chapter examines the existing mechanisms that ensure confidentiality, mutual authentication and authorisation of I/O peripherals and their computing devices.

In section 2.3 of Chapter 2, the secure container property (SCP) and the mutual authentication property (MAP) were defined as important properties of the Neo model. The drivers that assist in design the SCP of the Neo model has now been identified. The next chapter identifies drivers for the MAP.

Introduction

- Chapter 1 Introduction

The end game

- Chapter 2 Model Overview

Literature study

- Chapter 3 Isolation properties in mobile operating systems (SCP)
- Chapter 4 Isolation techniques in other technologies (SCP)
- Chapter 5 An overview of peripheral security (MAP)

Bridge

- Chapter 6 The drivers for a new model

Building the Neo model

- Chapter 7 Overview of the Neo Model
- Chapter 8 The Secure Container Property
- Chapter 9 The Mutual Authentication Property
- Chapter 10 Container life cycle
- Chapter 11 Novel Implementations

Evaluation and summary

- Chapter 12 Potential implementation of Neo Model
- Chapter 13 Conclusion and future work

Chapter 5 AN OVERVIEW OF PERIPHERAL SECURITY (MAP)

5.1 Introduction

In Chapter 1 it was discussed that people use various devices for both personal and business purposes. One of the reasons for using various devices is based on usability. While working at a desk in a company, a big screen with a physical keyboard may be preferred above a small touchscreen-based device like a smartphone. On the other hand, a small touchscreen-device may be preferable while using public transport to listen to music or audio books, or even get a little bit of reading done.

The form factor of input as well as output devices changes depending on the circumstances of the user. In the attempt to solve the problem of relying on multiple devices, a one-device solution is being explored. Using one computing device minimises the risk that exists when data needs to be transferred between devices. One computing device, however, locks the user into a specific configuration for input and output.

A solution where a user can only use one device must also address the problem of a user being bound to a specific form factor for input and output. This means the solution should allow the user to have the freedom to interact with the computing device using different form factor input and output peripherals.

This chapter evaluates the security mechanisms that exist in wired and wireless I/O peripherals. The focus of this chapter is to identify drivers that are used to help establish the design requirements that relate to the MAP for the Neo model. The same call-out mechanism used in the previous chapters are used to highlight and identify a specific driver.

This chapter focusses on the security that exists in various I/O peripherals when communicating with a computing device. The chapter is structured as follow:

- **Section 5.2** evaluates the architecture of wired peripherals and evaluates the security built into the communication of wired peripherals.
- **Section 5.3** focusses on wireless peripherals and describes the security mechanisms and security service applicable to wireless peripherals, as they exist at the time of writing.
- **Section 5.4** summarises the drivers identified in this chapter.
- **Section 5.5** provides a conclusion and makes an overall statement about the security of today's wired and wireless peripherals.

5.2 Security in wired peripherals

As a rule, the confidentiality of data or the authentication and authorisation of devices has not been a major requirement for wired peripherals. There are exceptions to this rule, however, but there are very few of them.

To provide structure, the threat landscape in wired I/O peripherals are evaluated using the security services defined by ISO 7498/2 (ISO, 1989). The five security services identified by ISO 7498/2 are:

- Identification and authentication.
- Authorisation.
- Confidentiality.
- Integrity.
- Non-repudiation.

Only three of the five security services defined by ISO 7498/2 are used in this section. The three security services used in this section are the identification and authentication service, the authorisation service and the confidentiality service.

This section is structured as follow:

- **Section 5.2.1** discusses the threats that exist when devices do not authenticate, together with some of the mechanisms available that minimise these threats.
- **Section 5.2.2** discusses some of the threats that exist when a device does not have permissions to interact with a computing device. The section also discusses the mechanisms and protocols that address these threats.
- **Section 5.2.3** describes the threats to the interception of transmitted data and some of the mechanisms available that address these threats.

5.2.1 Identification and authentication threats

Identification and authentication address the concern that a subject needs to be positively identified by a system. Positive identification is important in computer security to implement other security services, such as proper authorisation or confidentiality. If a subject cannot be identified, then the security model should either allow or disallow communication. In the case where communication is allowed without authentication, only one level of access is possible, it may not be possible to implement multiple levels of authorisation because of the lack of identification.

In wired devices, the identification threat focus is on technical threats, instead of security threats. In VGA connections, when a screen is connected to a computer, the computer should know the capabilities of the screen. If the capabilities of the screen were not known, the user has to inform the device driver of the capabilities of the screen, which could result in screens either not displaying anything or not performing to its best capabilities (Extron Electronics, 2018).

The functionality that allows for the communication of the device capabilities is also known as Extended Display Identification Data (EDID). EDID is available in many display types, which include Digital Visual Interface (DVI) and High-Definition Multimedia Interface (HDMI) standards. EDID may provide identity information in the form of a serial number, about devices, but does not have any authentication mechanism to confirm the identification information is correct.

In USB connections, USB devices have a hardware ID and instance ID. The hardware ID contains identifying information about the class of device, vendor information and revision information. More than one device can have the same hardware ID, since you may have more than one of the same device connected. The instance ID contains unique identification information of multiple instances of the same hardware that describes where they are connected to the USB bus. The instance ID may contain serial number information, but it may also only contain a unique number generated by the USB host.

When identifying information is available, the computer should trust that the identifiers have not been falsified. There is no authentication taking place in EDID or USB connections. USB Type-C, however, supports authentication. USB Type-C Authentication allows companies to set policies that approve specific devices to be used in on their computers (USB 3.0 Promoter Group, 2017). This may minimise threats where unidentified storage devices are plugged into computers, or where vendors are concerned about the charging capabilities of third-party charging devices.

USB Type-C Authentication, unfortunately, does not uniquely identify each device, instead, it identifies types of devices, from a specific manufacturer with a specific feature set. It may also identify devices that are owned and controlled by a specific company.

The individual identification and authentication of devices do not seem to be a capability that is addressed with today's hardware standards. The lack of electronic identification and authentication creates a reliance on the user to identify and authorise the device through visual means. Visual identification and authentication require users to actively do this, which may require training and the results may not be reliable. Attackers may create hardware that looks and acts exactly like a specific vendor's peripheral, instead it may contain either sub-standard hardware or may contain capabilities not known by the user. These capabilities may be hidden features that capture either user input or output.

The identification and authentication of wired peripherals are seriously lacking. Without proper identification and authentication, authorisation services and confidentiality service can easily bypass. In the few cases where identification information is available, but no authentication, very little trust can be put into the identification information provided. ***(I/O Peripherals require trustworthy identification and authentication)***.

Identification and Authentication:

- I/O Peripherals require trustworthy identification and authentication

The next section describes the threats that exist on wired peripherals when they are authorised to access anything and the problem of controlling access.

5.2.2 Authorisation threats

By default, when a user connects a peripheral to a computing device, the user implicitly authorises the device to get access to the computer. The authorisation is based on the discretion of the user of the system. This means, that even if the user does not own the data or the application, or even the computing device, the user may still authorise a device to interact with the device.

An exception to the rule is when the device requires a special device driver to be installed. In these cases, the user must have at least administrative permissions on the system to install and load the device driver.

One area where authorisation can be managed from a corporate perspective is with USB devices. A number of anti-malware software editions allow network administrators to set policies that allow the company to control which USB devices are authorised to be connected to the company's computers.

Authorisation levels of peripherals are either full access to the operating system or no access to the operating system. In the case where a user has the discretion to connect a peripheral when connected, any object on the operating system of the connected computer can use the peripheral. In the case where control is managed via system policies, it is still a case of having full access to the operating system or no access at all. ***(I/O Peripherals has either full or no access).***

The biggest threat to unauthorised peripherals usually belongs to devices that have storage capacity. Unauthorised storage devices may contain malware or it can be used to copy sensitive information from the computer.

The authorisation of wired peripherals is not addressed in most systems. Wired peripherals seem to have an all or nothing approach. The authorisation is reliant on physical actions and companies do not have the capability to authorise devices to only certain systems or certain data. ***(Operating systems lack built-in authorisation mechanisms for most wired peripherals)***.

Authorisation:

- I/O Peripherals has either full or no access.
- Operating systems lack built-in authorisation mechanisms for most wired peripherals.

The next section described the confidentiality aspects of wired peripherals.

5.2.3 Confidentiality threats

Confidentiality in peripherals may be classified according to the confidentiality of data while at rest or while being transmitted. A typical USB memory device is dependent on the file system and another mechanism to ensure data stays confidential when it has been copied to the device. Most USB memory devices do not have confidentiality capabilities at all, which is a real threat to the confidentiality of the data it contains.

Confidentiality of data as it is transmitted is also problematic for most wired peripherals. Most wired connections do not provide any type of encryption. Confidentiality threats for data while being transmitted is a reality in the cases of keyloggers or screen grabbers.

Without getting into too much detail, a hardware keylogger is a device that sits either in-line with your keyboard and computer or connects to the computer's data bus to intercept and record key presses.

Figure 20 depicts an in-line keylogger that can record typing and store the results inside the USB device in an encrypted format. The vendor of the keylogger depicted, claims that the device can store up to 1.5 million keystrokes. This specific keylogger ships with a utility application that is required to get access to the captured keystrokes (KeyCarbon LLC, 2017).

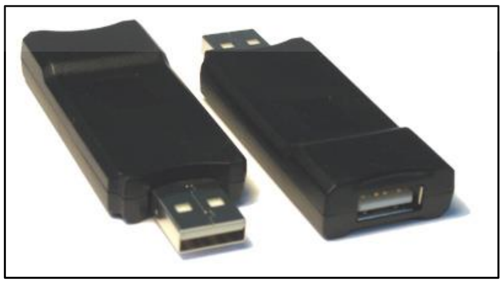


FIGURE 20: IN-LINE USB KEYLOGGER (KEYCARBON LLC, 2017)

Figure 21 depicts a keylogger that plugs into the mini-PCI bus of the computer. Using a mini-PCI to PCI bus, the card can be installed on a desktop computer. The keylogger passively records key presses and is not externally visible on the computer.

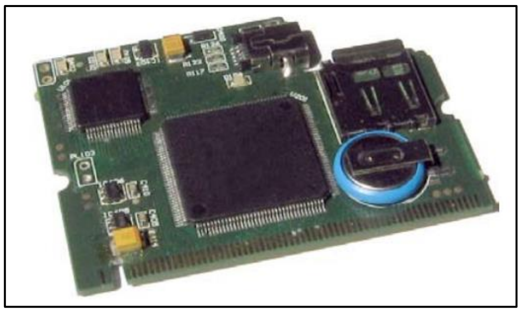


FIGURE 21: PASSIVE KEY LOGGER (KEYCARBON LLC, 2017)

The hardware keyloggers allow attackers to record potentially sensitive information typed by a user. Hardware screen grabbers are installed in-line between the computers video port and the monitor or other external display. The hardware screen grabber records video output and saves it as separate JPEG-encoded pictures.

Figure 22 shows a DVI version of a screen grabber. The one end plugs into the video port of the computer, with the USB to help power the device. The other end plugs into the monitor. The screen grabber has a powerful processor and up to 8GB of memory to store captured images (KeeLog, 2016).

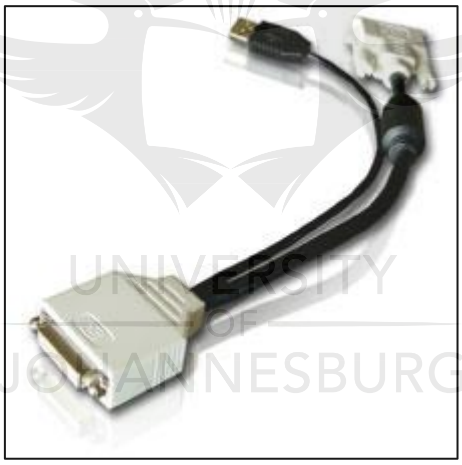


FIGURE 22: A HARDWARE SCREEN GRABBER (KEELOG, 2016)

Screen grabbers, also called frame grabbers, allow an attacker to view potentially sensitive or confidential information that is being accessed by the user. One of the biggest threats of these hardware devices is that they do not require any software to be installed on the computer. If the user does not visually identify the threat, then there is very little that can be done to detect these type of attacks. ***(Even in wired peripherals, unencrypted signals may be intercepted by attackers)***.

There were no known mechanisms found that ensure confidentiality of video signals to an output peripheral. Treat (2002) describes a device that can either be added externally to an existing keyboard or replace existing keyboard hardware that ensures keys are encrypted before it travels over the wire that connects the physical keyboard to the computer and then decrypts before the computer's BIOS interprets the key.

Confidentiality:

- Even in wired peripherals, unencrypted signals may be intercepted by attackers

Security in wired peripherals is not normally seen as a major risk factor. Wireless communication, on the other hand, receives a lot of attention and most wireless protocols address the three security services as described in the next section.

5.3 Security in wireless peripherals

I/O peripherals that communicate wirelessly with a computing device make use of either Bluetooth (Bluetooth SIG, Inc, 2017) or WiFi (IEEE, 2017) protocols to establish wireless connectivity.

Bluetooth is generally used for input devices, but may also be used for some output peripherals, like speakers or headphones. Bluetooth is a low power, but limited bandwidth technology. WiFi has a higher cost of power but has a lot more bandwidth available. Miracast (WiFi Alliance, 2013) is a WiFi standard that allows streaming of video and sound to output peripherals.

This section discusses the security services of identification and authentication, authorisation and confidentiality in wireless communication. The security service of identification and authentication is discussed first since the other two security services depend on authentication.

5.3.1 Identification and authentication

On a wireless connection, the authentication of a device ensures that devices cannot impersonate other devices. Without proper authentication, devices may impersonate other devices which may allow an attacker access to a computing device, either for input or output purposes.

The Bluetooth protocol uses a Bluetooth device address (BD_ADDR) as a unique identifier for each Bluetooth device. Authentication happens by establishing an initial shared secret between the two devices. In literature, the initial shared secret is also known as K_{init}. K_{init} is created using a PIN number and a pseudorandom number. After K_{init} has been established, a unit key (K_a) is established that is used for further authentication (Haataja et al., 2013) (Sandhya & Devi, 2012).

Up to version 2.0 of the Bluetooth standard was susceptible to a man-in-the-middle attack, where an attacker could intercept the initial identification exchange and inject their own identification. With the later versions of Bluetooth, the initial authentication became known as a secure simple pairing (SSP) and was hardened against man-in-the-middle attacks (Bluetooth SIG, Inc, 2017). ***(Bluetooth has definite identification and authentication mechanisms)***.

SSP allows two devices from establishing identity by either using an out-of-band (OoB) connection, like near field communication (NFC) or visually comparing a set of numbers on the two devices. By confirming that both sets of numbers are identical between the two devices, the user ensures that the communication is established between the two devices and is not currently being intercepted by a third party. ***(I/O peripheral authentication has a higher level of trust when using OoB authentication)***.

In a WiFi network, security is established by using WiFi protocols such as Wireless Equivalent Privacy (WEP) and Wi-Fi Protected Access (WPA and WPA2) (Khasawneh et al., 2014). Of the three security standards, WPA2 is, at the writing of this document, the most secure of these methods, with the least amount of known vulnerabilities.

WiFi security uses a slightly different model than Bluetooth. Most WiFi protocols create a wireless network that can be used by multiple devices at a time. The WiFi network can be established by many different devices, which may include a wireless access point (WAP). The Bluetooth protocol establishes connections only between two devices.

WPA2 authentication may either be established using Personal or Enterprise modes. In Personal mode, the authentication and authorisation occur using a pre-shared key (PSK). A password, together with network identifier (SSID) is used to derive a cryptographic pairwise master key (PMK). Both the device that wants to join the WiFi network and the WAP should now know the PMK. Each device on the same WiFi network using WPA-PSK has the same PMK. The PMK is used more as an authorisation mechanism since WPA-PSK does not identify WiFi clients the same way that WPA2 in Enterprise mode (Also known as WPA-802.1x) (Khasawneh et al., 2014).

WPA2-Enterprise uses a specific authentication server that authenticates each device, using either the user's identity, certificate or other means to authenticate. After the authentication step, the authentication server assists in setting up the PMK that is then used for confidentiality purposes. The PMK established in WPA2-Enterprise is different for each client.

Identification and Authentication:

- Bluetooth has definite identification and authentication mechanisms.
- I/O peripheral authentication has a higher level of trust when using OoB authentication

Bluetooth has a definite identification and authentication step built into the Bluetooth process. WPA2-PSK does not identify each device using some type of unique identifier, while WPA2-Enterprise uses a specific authentication service to authenticate each client. The lack of proper identification in WPA-PSK plays a role in the authorisation and confidentiality of devices that make use of WPA2-PSK. The next section describes the level of access established in Bluetooth and WPA2.

5.3.2 Authorisation

The authorisation security service as defined by ISO 7498/2 concerns itself with managing levels of access. This implies a client has already been authenticated and that the authentication process was successful. Authorisation in a system manages the level of access a specific client has in a system.

In a Bluetooth network, a device that has just been authenticated is added to a trusted list of devices. Any device in the trusted list may access the Bluetooth profile specific services that it supports. This means that an authenticated Bluetooth keyboard may interact with the human interface device (HID) service on the computing device. In this case, access is either granted or not granted. There is only one level of access. The device is either allowed or not allowed. This is also the same as the File Transfer Profile Bluetooth service. The Bluetooth subject has either access or not access to transfer files from the Bluetooth object.

When a Bluetooth connected device has been granted access, the device can be used by the user of the system in any application or data file where the user has permissions. This means that the Bluetooth device inherits the access level of the user of the system. ***(Authorisation levels of I/O peripherals depend on the authorisation level of the user)***.

WPA2 is slightly different. WPA2-PSK goes through the authentication process and then allow access to the network. In this case, the level of access on the network is either allowed or not allowed. The WiFi network does not manage access to systems, instead, relies on the systems themselves to manage access. With WPA2-Enterprise, the authentication server, interface with a RADIUS server.

The RADIUS server helps with the authentication process to either allow or not allow access to a network, but may also move users to a different network. Users that belong to a special group or access the network during a defined time frame, may be moved to a less sensitive network zone than other users. This means that with WPA2-Enterprise access to different networks may be established given a specific set of criteria.

Bluetooth and WPA2, in general, have either an allow or deny the level of access. WPA2-Enterprise allows access to be granted to different levels of networks, defined by an administrator, instead of just allowing or denying access to the network. ***(Fine-grained authorisation is not possible on a per I/O peripheral basis)***.

Authorisation:

- Authorisation levels of I/O peripherals depend on the authorisation level of the user.
- Fine-grained authorisation is not possible on a per I/O peripheral basis.

The next section describes how Bluetooth and WPA2 establish confidential connections between subjects and objects.

5.3.3 Confidentiality

Confidentiality is defined as ensuring an unauthorised party, even in cases where the data may have been intercepted, cannot read data. The confidentiality of data may be ensured by using encryption and decryption (ISO, 1989). This section provides an overview of the encryption of data in Bluetooth and WPA2 protocols.

Section 5.3.1 described the process that authenticates a Bluetooth device with another Bluetooth device. During the authentication process, a link key is established that is shared by both devices. The link key is used during further communication to generate an encryption key that encrypts the communication between the two devices. The encryption algorithm uses a stream cypher to encrypt and decrypt the data (Haataja et al., 2013).

Bluetooth allows two devices to select whether they would like to communicate over an encrypted channel or not. This allows cheaper Bluetooth devices to establish communication without any encryption. The communication confidentiality is dependent on the capabilities of the Bluetooth device (Haataja et al., 2013) .

WPA2 uses the pairwise master key (PMK) that is established during the authentication process. Using the PMK, both the WAP and the WPA2 client generates a pairwise transient key (PTK). The PTK is used in establishing an encrypted channel between the client and the WAP. The PTK is different for each client and is created immediately after the PMK. The PTK ensures the confidentiality between the client and the WAP. ***(Wireless signals are encrypted to and from I/O peripherals and computing devices)***.

Even though the PTK is different for each client, the PMK may not be, as in the case of WPA2-PSK. Since WPA2-PSK uses a password or passphrase to create the PMK, a brute force attack may reveal the PMK used by all the clients on the WPA2-PSK secured network. As soon as an attacker has the PMK the attacker can derive the PTK for a specific client. The attacker has to capture the traffic generated between the client and the WAP as authentication occurs to derive the PTK.

The re-authentication process can be forced onto any client by sending the client a disconnect packet. Upon receiving the disconnect packet, the client automatically re-authenticates and establishes a new PTK. The attacker, knowing the PMK and having captured the traffic between the client and WAP can also derive the PTK, which allows the attacker to decrypt any of the encrypted packets. ***(Care should be taken when establishing cryptographic keys)***.

Deriving the PTK in a WPA2-Enterprise environment is a lot more difficult since the PMK is not derived from a password, that must be used by all the clients. The PMK is also different for each client. WPA2-Enterprise ensures stronger authentication and confidentiality than WPA2-PSK. ***(Pairwise Transient Keys and Link Keys are ephemeral keys providing perfect forward secrecy)***.

A modern operating system implements Bluetooth and WPA2 encryption as part of the operating system. Applications are not responsible to ensure encrypted channels are established between the client and server network nodes. The encrypted network channel is established and maintained by the operating system and network device. ***(Encryption stops and starts at the operating system level)***.

Confidentiality:

- Wireless signals are encrypted to and from I/O peripherals and computing devices.
- Encryption stops and starts at the operating system level.
- Care should be taken when establishing cryptographic keys.
- Pairwise Transient Keys and Link Keys are ephemeral keys providing perfect forward secrecy.

Both Bluetooth and WPA2 ensures some level of confidentiality between different devices on the wireless network. The Neo model concerns itself with a mobile operating environment and as such requires confidentiality of wireless communication. Both Bluetooth and WPA2 are technologies that may be adopted into the Neo model.

5.4 Summary of drivers

Security mechanisms in wireless communication are fairly mature. Some wired peripherals do not consider security as an important requirement. A number of security aspects have been identified in both wired and wireless security. The identified security aspects act as drivers towards the design of the Neo model. The identified drivers are summarised in Table 5.

Table 5 structures the identified drivers into identification and authentication, authorisation and confidentiality. The drivers for the three security services are used in Chapter 6.

TABLE 5: I/O PERIPHERAL SECURITY DRIVERS.

Wired	Wireless
Identification and Authentication	
I/O Peripherals require trustworthy identification and authentication	Bluetooth has definite identification and authentication mechanisms.
	I/O peripheral authentication has a higher level of trust when using OoB authentication
Authorisation	
I/O Peripherals has either full or no access.	Authorisation levels of I/O peripherals depend on the authorisation level of the user.
Operating systems lack built-in authorisation mechanisms for most wired peripherals.	The fine-grained authorisation is not possible on a per I/O peripheral basis.
Confidentiality	
Even in wired peripherals, unencrypted signals may be intercepted by attackers	Wireless signals are encrypted to and from I/O peripherals and computing devices.
	Encryption stops and starts at the operating system level.
	Care should be taken when establishing cryptographic keys.
	Pairwise Transient Keys and Link Keys are ephemeral keys providing perfect forward secrecy.

5.5 Conclusion

Computing devices that allow multiple Input and Output peripherals to connect to them allow the user the freedom to use the device in various environments. Bigger screens are useful in environments where the user is far away from the screen or more than one person needs to view the screen. Certain types of data are also more suited to different screen sizes. Documents may be difficult to view and edit on very small screens. Physical keyboards provide tactile feedback and may allow certain people to type very fast, while smaller touchscreen keyboards may be more useful in mobile scenarios.

This chapter evaluated the existing security in both wired and wireless I/O peripherals. Three security services from ISO 7498/2 were used as a measuring device to measure and describe the threats and mechanisms available in the specific security service.

The identification and authentication service establishes positive identities between subjects and objects. Without proper authentication, any provided identity information cannot be trusted. Most of the information security service defined by ISO 7498/2 is dependent on successful authentication.

Sections 5.2.1 and 5.3.1 discussed the identification and authentication of both wired and wireless peripherals. USB devices and video signals may carry some form of identification in wired peripherals. Wireless communication in Bluetooth peripherals goes through an identification and authentication process. Wireless communication in WiFi networks may identify approved devices in WPA2-PSK using a shared secret between all connected devices, while WPA2-Enterprise supports individual identification and authentication.

The identification and authentication of wired peripherals are lacking and users are generally allowed to connect any I/O peripheral to their computing device. The inability to positively identify and authenticate a wired I/O peripheral from the actual computer system may cause unauthorised devices to connect and potentially damage or steal information.

The identification and authentication of wireless peripherals are a lot better than in wired connections. In general, identification and authentication is part of the communication protocol for Bluetooth, but ad-hoc and home use of WiFi connections rely on the user identifying a device and allowing the device on the WiFi network. The Bluetooth protocol is ideally suited for certain types of I/O peripherals, although it may be limited in the amount of bandwidth available, making it more suited for input devices instead than output devices.

Sections 5.2.2 and 5.3.2 discussed the authorisation of connected I/O peripherals. Wired I/O peripherals mostly get their authorisation through the action of the user. Wireless I/O peripherals are authorised when positively authenticated. For both wired and wireless I/O peripherals, access is granted to the system and in general, cannot be controlled to include or exclude access to certain data sets or applications.

Sections 5.2.3 and 5.3.3 established that confidentiality is near non-existent in wired I/O peripherals, but is built into both Bluetooth and WiFi, using WPA2 security. Depending on the implementation of WPA2, an attacker using a number of techniques can defeat confidentiality, but the biggest flaw is the shared password used by all devices in WPA2-PSK.

Section 5.4 provided a summary of the security aspects that act as drivers for the Neo model. These drivers are used as input in Chapter 6 to derive specific design requirements.

In an environment where I/O peripherals need to communicate with a computing device, security is of utmost importance. Attackers may interact, intercept or connect to a computing device without having to be physically close to the computing device. The authorisation of devices usually works on an all or nothing basis. Even on devices where applications or data does not belong to the user of the device, peripherals may interact with these devices without the owner's permission.

On computing devices that rely on the application and data isolation, I/O peripheral access must be controlled and authorised. The need for fine-grained authorisation levels may also be necessary in order for owners to ensure I/O peripherals may only be used in the specific environment, periods or situations.

The drivers that affect the design for the secure container property (SCP) and mutual authentication property (MAP) of the Neo model has now been identified. This chapter also concludes the literature study of the research. The next chapter summarises the drivers identified in Chapters 3 to 5. The drivers are consolidated and evaluated and a set of design requirements are identified in the next chapter.

Introduction

- Chapter 1 Introduction

The end game

- Chapter 2 Model Overview

Literature study

- Chapter 3 Isolation properties in mobile operating systems (SCP)
- Chapter 4 Isolation techniques in other technologies (SCP)
- Chapter 5 An overview of peripheral security (MAP)

Bridge

- Chapter 6 The drivers for a new model

Building the Neo model

- Chapter 7 Overview of the Neo Model
- Chapter 8 The Secure Container Property
- Chapter 9 The Mutual Authentication Property
- Chapter 10 Container life cycle
- Chapter 11 Novel Implementations

Evaluation and summary

- Chapter 12 Potential implementation of Neo Model
- Chapter 13 Conclusion and future work

Chapter 6 The Drivers for a New Model

6.1 Introduction

The approach, as described in Chapter 1 section 1.5, taken in this thesis is to conduct an extensive literature study. The literature study, conducted in Chapters 3 to 5 identified a number of security aspects. The security aspects focused on the isolation mechanisms used by mobile operating systems, but also other technologies, like virtualisation. The security aspects also included the security mechanisms used to ensure a certain level of security when communicating with I/O peripherals.

The security aspects identified in Chapters 3 to 5 were summarised as drivers at the end of each of the chapters. The aim of this chapter is to disseminate the drivers and derive a set of inputs when designing the Neo model. The inputs strengthen the aspects of the secure container property (SCP) and mutual authentication property (MAP).

This chapter is organised as follow.

- **Section 6.2** reviews the drivers identified in the previous chapter pertaining to the secure container property (SCP).
- **Section 6.3** reviews the drivers identified that pertains to the mutual authentication property (MAP).
- During the review process, a number of requirements are identified. The requirements are summarised in **section 6.4**.
- **Section 6.5** provides a conclusion to the identification of the requirements.

The call-out mechanism is again used in this chapter. Figure 23 describes the changed callout mechanism. Each callout consists of a design category that depicts either SCP or MAP. SCP categories are secure container properties and MAP categories are mutual authentication properties. Each callout and design requirement is numbered. Example: Figure 23 describes a SCP Design Category with number one. The category show one SCP design requirement, numbered 1.1.

SCP1. Design Category:
SCP1.1. Design requirement.

FIGURE 23: CATEGORISING DESIGN REQUIREMENTS.

Table 11, found at the back of this document under Appendix A, is constructed in such a way so that it can be folded out. Table 11 lists the final requirements identified in this chapter for the SCP. As callouts are created, a corresponding entry exists in Table 11.

6.2 Identified drivers - SCP

The Neo model defines two properties which address the objectives of this study. The first property is the secure container property. This property concerns itself with the isolation of various components in the Neo model. It specifically describes the isolation of data and applications belonging to a specific owner from data and applications belonging to another owner. This section disseminates the drivers identified in Chapter 3 and Chapter 4.

The identified drivers related to the SCP were summarised in Table 3, section 3.4 and Table 4 section 4.5. This section is structured around the driver categories identified in Table 3 and Table 4:

- **Section 6.2.1**. The architectural design methodology is decided on in this section.
- **Section 6.2.2**. The required mechanisms that must be used are discussed in this section.
- **Section 6.2.3** discusses the mechanisms that enable application communication in the Neo model.
- **Section 6.2.4** describes the mechanisms used to ensure the privacy of applications and data of multiple owners in the Neo model.
- **Section 6.2.5** evaluates the attack methods and shortcomings in existing mobile operating systems and produce a requirement for the Neo model that addresses these attacks and shortcoming.
- **Section 6.2.6** produces requirements that ensure easy and reliable file system deployments in the Neo model.
- **Section 6.2.7** highlights the requirements to ensure operating system confidentiality.
- **Section 6.2.8** identify requirements for network-level access requirements.
- **Section 6.2.9** highlights the importance and requirements of a protected management environment.
- **Section 6.2.10** identifies the requirements for some of the communication that may occur in the operating environment.
- **Section 6.2.11** addresses performance concerns of the Neo model.

6.2.1 Design methodology

Hypervisor technologies such as Qubes-OS provide a hypervisor level isolation level. Container technologies such as Cells provides operating system level isolation and existing mobile operating systems such as iOS and Android provide application-level isolation.

Figure 24 depicts the different levels of isolation identified:

- Level 0 describes hypervisor-based isolation. Hypervisor-based isolation isolates full operating systems. Compromises in the operating system level are isolated from other operating environments (Huber et al., 2016).
- Level 1 describes operating system level isolation. The operating system level isolation provides mechanisms that isolate operating system objects from each other. The mechanisms do not provide the strong level of isolation that hypervisors have but are less resource intensive than level 0 (Bui, 2015) (Huber et al., 2016).
- Level 2 describes application-level isolation. Application level isolation requires strong control over applications. Compromises in the operating system level may affect all the applications on the operating system but is a lot less resource intensive than hypervisor-based isolation (Bui, 2015).

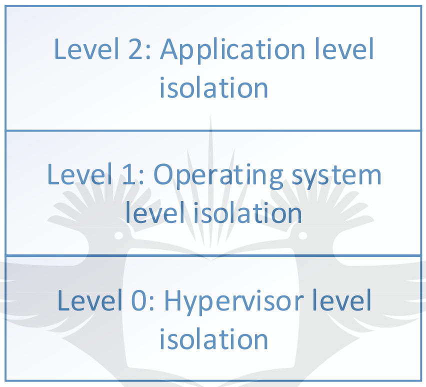

Level 2: Application level isolation
Level 1: Operating system level isolation
Level 0: Hypervisor level isolation

FIGURE 24: LEVELS OF ISOLATION (BY AUTHOR)

The Neo model describes a secure operating environment and as such require a deep level of isolation between different components. This may have a detrimental effect on the implementation of battery-powered devices. Continued development and future work on hypervisor designs may provide more battery friendly hypervisors that can be used on the mobile device.

SCP1. Design Methodology:
SCP1.1. Hypervisor level isolation.

The next aspect described is the mechanisms used that help to provide the isolation required for the SCP.

6.2.2 Isolation mechanisms

The isolation mechanisms discussed in Chapter 3 were based on application level isolation identified in Android and Apple iOS mobile operating systems. In the previous section, a design requirement for hypervisor level isolation has been made. Because of this decision, only the drivers identified in that relates to hypervisor-based isolation is applicable. No new requirement is identified for the isolation mechanism.

Isolated application environments provide strong controlled access, but the requirement for communication application is a reality. The next section describes the drivers for application communication.

6.2.3 Application communication

Different application designs require various application components to interact and share data with each other. In a hypervisor-based isolation model, applications in the same virtual machine utilise existing operating system mechanisms to communicate with each other.

Mobile operating system applications use strong communication components to share and exchange data between isolated applications. The Neo model provides an environment that allows applications that require simple communication to be used in the same containers, while applications that should be isolated from each other run in different containers.

Communication between containers is not allowed in the Neo model, except in strongly controlled cases. Strongly controlled cases include cases where the management environment may need to change the configuration in containers or where a policy allows data to be transferred from a container to external storage.

SCP2. Application communication:

SCP2.1. Applications inside containers communicate with each other using modern operating system objects.

SCP2.2. Applications in different containers cannot communicate with each other.

SCP2.3. Communication between containers are only allowed through strongly controlled mechanisms.

6.2.4 Privacy protection

Mobile operating systems provide mechanisms that allow companies to control applications and data belonging to them. The mechanisms allow the companies to control the configuration of the applications, the network security of the applications and the identification between personal and business applications and data through categorisation.

The Neo model implements strong isolation and the container mechanism provided can be used to isolation business and personal data and application. The environment should allow owners of applications and data, control over configuration, network security and access. The Neo model should allow owners to set policies enforced by the environment. The policies should be protected from unauthorised modification.

SCP3. Privacy protection:
SCP3.1. Owner based isolation of data and applications in containers.
SCP3.2. Configuration and access control using policy enforcement.

6.2.5 Attack methods and shortcomings

Between Apple iOS and Android, iOS provides an environment that allows the vendor to respond to certain vulnerabilities quicker than Android. Apple can respond better than Android because Apple does not allow the modification of the operating system environment and hardware.

Qubes-OS provide a strongly controlled management environment that is updated by the development team of Qubes-OS. The management environment provides the isolation mechanisms enforced by the system. The Neo model requires a strongly controlled management environment that may not be changed by different vendors or users. The strongly controlled management environment allows the developers and implementers to respond to identified vulnerabilities quickly.

The delivery mechanism of the management environment should ensure the integrity of the management environment, during update and after updates. The hardware should also provide a mechanism that guarantees the integrity of the management environment.

The requirement identified in this section is not called, "Attack methods and shortcoming", instead it is categorised under the protected management environment.

SCP4. Protected management environment:
SCP4.1. Unauthorized changes to the management environment is not allowed.
SCP4.2. Updating of the management environment should guarantee integrity during update and after update.
SCP4.3. Integrity of the management environment should be confirmed by the hardware

6.2.6 File system deployment methodology

Qubes-OS provides the deployment of guest operating systems through a template. The template is a file system of a guest operating system. Changes between each deployed virtual machine are accomplished using copy-on-write technology.

The use of templates allows administrators to manage the features and configuration of a deployed virtual machine. The isolated containers are still assured through the use of copy-on-write but still minimising the storage requirement of each container.

SCP5. File system deployment:
SCP5.1. Templates should be used to provide baseline container configuration.

6.2.7 Operating environment confidentiality

Qubes-OS provides confidentiality of the various virtual machines by making use of encryption at the file-system level. The Neo model assures confidentiality between containers. Each container should be encrypted using keys controlled by the owner of the container.

Removable storage poses a risk to the integrity and confidentiality of containers. Access to removable storage is strongly controlled. Removable storage may never access a container directly instead access is only allowed to a temporary container from where files can be sanitised.

SCP5 which have already been identified in section 6.2.6 are extended in this section.

SCP5. File system deployment:
SCP5.1. The use of templates provide baseline container configuration.
SCP5.2. Access to removable storage is strongly controlled.
SCP5.3. Deployed containers are encrypted on the hard disk.

6.2.8 Network access security

Qubes-OS has a dedicated firewall virtual machine where all network traffic is routed through. Linux container implementations ensure network traffic to a specific namespace cannot be accessed by another namespace instance.

The Neo model adopts the principle of controlled network access. Network traffic is not allowed to access any data and application container directly, instead indirect access mechanisms should be used. Network traffic is directed to ensure no other container has access to network traffic meant for it.

SCP6. Network access security:

SCP6.1. No direct access from inside any application and data containers to the network, instead indirect mechanisms should be used.

SCP6.2. Data and application containers can only access network traffic directed to and from it.

6.2.9 Protected management environment

A management environment is used by Qubes-OS to control the virtual machines used by the user. The management environment is also protected in the Qubes-OS implementation from network traffic by not connecting the management environment to the network directly.

The Neo model requires a management environment in its own secure container. The management environment should be protected from network attacks. Unauthorised access should not be allowed in the management environment.

SCP4 that was identified in section 6.2.5 are repeated below but expanded with one more requirement.

SCP4. Protected management environment:

SCP4.1. Unauthorized changes to the management environment is not allowed.

SCP4.2. Updating of the management environment should guarantee integrity during update and after update.

SCP4.3. The integrity of the management environment should be confirmed by the hardware

SCP4.4. Network connections to the management environment is not allowed.

6.2.10 Operating environment communication

Applications in specific Linux namespaces by default cannot access applications in other namespaces. Qubes-OS allow inter-domain communications, but only through very specific interfaces management by the management environment.

The Neo model should not allow communication between application and data containers. Applications inside the same containers are allowed through container specific interfaces.

No new requirement has been identified in this section and falls in with R2 identified in section 6.2.3.

6.2.11 Processing performance

It has already been mentioned in section 6.2.1 that mechanisms ensuring level 0 isolation is normally more resource intensive than level 2 isolation. The Neo model adopts level 0 isolation. This may affect the performance of an implementation of the Neo model. The author of this thesis chose level 0 isolation for security purposes rather than the increased performance that level 1 or 2 isolation may provide.

Processing performance, while being a driver identified in the previous chapters are not a requirement identified for the Neo model.

SCP1 to SCP6 are the major requirements identified in this section that the Neo model must adopt. These requirements specifically relate to the SCP of the Neo model. The next section identifies requirements that relate to the mutual authentication property (MAP).

6.3 Identified drivers - MAP

The mutual authentication property (MAP), as described in Chapter 2 section 2.3 concerns itself with the secure communication between the computing device and I/O peripherals. The MAP specifically concerns itself with the:

- **Mutual authentication** of the I/O peripheral and the Neo device.
- The **authorisation** of I/O peripherals to access a Neo device and container on the device.
- And lastly the **confidentiality** of any communication between the secure container and I/O peripheral.

This section evaluates the drivers identified in Chapter 5 and summarised in Table 5. From the drivers, a number of MAP requirements are identified. Section 6.3.1 to 6.3.3 evaluates the drivers under specific security service categories.

Table 12 in Appendix A can now be folded out. Table 12 lists the final MAP requirements identified in this chapter. The reader can follow along as MAP requirements are identified and added to the table.

The first of the three security services used during the process is identification and authentication.

6.3.1 Identification and Authentication

Existing wireless protocols such as Bluetooth has a well-defined identification and authentication process. The use of out-of-band (OoB) mechanisms minimise the chances of man-in-the-middle attacks.

The Neo model requires strong identification and authentication for both the Neo device to I/O peripheral and I/O peripheral to Neo device. Initial identification and authentication that helps establish a shared secret is critical to ensure future secure communication and requires OoB connections to minimise man-in-the-middle attacks.

The Neo model does not specify that existing protocol, like Bluetooth, must be used, but any implementation should consider the maturity of a specific protocol.

MAP1: Identification and authentication
MAP1.1. Mutual authentication is required for Neo device and I/O peripheral.
MAP1.2. OoB mechanisms must be used during initial identification and authentication.

6.3.2 Authorisation

Existing authorisation implementations of I/O peripherals are based on the security level of the user that uses the system. Operating systems as a generalised rule do not have built-in authorisation mechanisms for I/O peripherals, but instead, leave it up to the communication protocol to implement. Access control levels for I/O peripherals are either access or no access. Access level means that the I/O peripheral has access to the device, while no-access means that the device does not have access to the device.

The Neo model requires per-container level access control of I/O peripherals. A combination of user, I/O peripheral and container is used to determine access to a container. The authorisation of I/O peripherals should be managed by the management system of the Neo device. Policies and access control rules set by the container owners affect the access control rules defined in the management system.

MAP2: Authorisation
MAP2.1. Access control triplets must be used to determine access. The access control triplet consists of a user, I/O peripheral and container identifier.
MAP2.2. Access control is managed by the management system of the Neo model.
MAP2.3. The owners of containers set access control.

6.3.3 Confidentiality

Encryption is used in most wireless protocols between devices. The use of cryptographic keys plays an important role in ensuring encryption between nodes. The use of once-off session keys (ephemeral keys) minimises the risk of full compromise when shared secrets or long life keys are compromised. Encryption is usually established in the protocol and ensures encryption inside the protocol.

The Neo model requires encryption between Neo device and I/O peripheral. The encryption keys used during the encrypted sessions should be protected from any user access or modification.

MAP3: Confidentiality
MAP3.1.Only encrypted communication is allowed between Neo device and I/O peripheral.
MAP3.2.The management of encryption keys should occur in the management environment and not from within containers where users have access.

This section identified three overall requirements for the MAP. The three requirements set specific requirements for identification and authentication, authorisation and confidentiality. The next section summarises the requirements in table format so that it may be referenced in future sections of the thesis.

6.4 Architectural model requirements

Section 6.2 and section 6.3 consolidated the drivers identified during the literature study. During the consolidation process, a number of requirements were identified. The requirements are categorised under requirements for the SCP and the MAP. This section summarises the requirements into two tables. Table 11 list the requirements for the SCP. Table 12 list the requirements for the MAP.

There are seven categories of the SCP requirements. Table 11 list the seven categories under SCP1 to SCP6. SCP1 describes the isolation level that is adopted by the Neo model. SCP2 provides important information regarding how applications communicate. SCP3 discuss specific requirements that address the separation of company and private information. SCP4 highlights the management environment and the protection of the management environment in the Neo model. SCP5 touches on mechanisms that assist in the deployment of container images, as well as the confidentiality of deployed containers. SCP6 describes requirements for network security.

Table 12 summarises the requirements for the MAP. The requirements are grouped into three categories. MAP1 describes the requirements for identification and authentication. It highlights the importance of mutual authentication and an initial OoB identification and authentication. MAP2 describes the access control services, grouped into a category called authorisation. The access control requirements describe access control between the user, container and I/O peripheral. MAP3 concerns itself with the requirements that ensure confidentiality during I/O peripheral communication.

6.5 Conclusion

During the literature study, a number of drivers that must be taken into account when designing the Neo model were identified. This section disseminated the drivers into a set of requirements.

Section 6.2 evaluated the drivers identified while studying container and isolation mechanisms. 12 Category drivers were identified. The 12 driver-categories were disseminated into six SCP requirement categories. Each SCP requirement category highlighted one or more requirements that the Neo model should address when designing the Neo model.

Section 6.3 disseminated the drivers identified while studying the security aspects of I/O peripherals. The drivers were reduced and a set of three MAP requirement-categories were identified. The MAP requirements categories were based on the three of the five ISO 7498/2 security services. The three MAP requirement categories were requirements for identification and authentication, authorisation and confidentiality.

The Neo model describes a mobile computing environment that ensures two important security properties. The SCP and MAP are the basis of the Neo model and the requirements identified in this chapter are used in the next section of this study where the architectural model of the Neo model is designed.

To provide a better understanding of the Neo model, the next chapter provides an overview of the Neo model. The overview gives a high-level overview of the model which is then further described in the rest of this document.

Introduction

- Chapter 1 Introduction

The end game

- Chapter 2 Model Overview

Literature study

- Chapter 3 Isolation properties in mobile operating systems (SCP)
- Chapter 4 Isolation techniques in other technologies (SCP)
- Chapter 5 An overview of peripheral security (MAP)

Bridge

- Chapter 6 The drivers for a new model

Building the Neo model

- **Chapter 7 Overview of the Neo Model**
- Chapter 8 The Secure Container Property
- Chapter 9 The Mutual Authentication Property
- Chapter 10 Container life cycle
- Chapter 11 Novel Implementations

Evaluation and summary

- Chapter 12 Potential implementation of Neo Model
- Chapter 13 Conclusion and future work

Chapter 7 OVERVIEW OF THE NEO MODEL

7.1 Introduction

Chapter 1 highlighted the problems that arise when users use multiple mobile devices and the need to exchange data between the various mobile devices. Chapter 1 defined a primary and secondary objective. The primary objective is to develop a model for a mobile operating environment. This model takes into account that data and applications belong to various owners, but used by one user. The model prescribes that a user only use one device for both personal and business purposes, thereby minimising the dependency on potential insecure cloud storage. Because of the one-device requirement, users need to have the option to use different I/O peripherals with this one device to ensure successful human-computer interaction, based on their requirement.

The aim of this chapter is to introduce the Neo model. The Neo model describes how multiple I/O peripherals can communicate with one device and how the device ensures isolation of programs and data belonging to different owners. The chapter guides the reader through a high-level overview of the various components that make up the Neo model.

The chapter is organised as follow:

- **Section 7.2** of this chapter provides a high-level overview of the Neo model. The general principles and properties are described in section 7.2.
- **Section 7.3** introduces the user to the Neo device and the management system on the Neo device.
- **Section 7.4** describes the architecture behind the various I/O peripherals. It further highlights how the system ensures access control to the various containers in the Neo device.
- **Section 7.5** describes the components and processes required to ensure the Neo device can communicate in a corporate environment.
- **Section 7.6** describes how the Neo model ensures device interconnectivity in cases where data can be shared between different devices.
- **Section 7.7** concludes this chapter by summarising the features, the existing research outcomes from this chapter.

A mechanism is employed that helps keep track of the SCP and MAP requirements identified in Chapter 6 section 6.4. The two tables, Table 11 and Table 12 can again be folded out in Appendix A: Special foldouts. In this part of the thesis, whenever an SCP or MAP requirement is addressed, a special callout with the relevant requirement number, as displayed in Figure 25, will appear in the paragraph of the text.

FIGURE 25: REQUIREMENT CALLOUT (BY AUTHOR)

7.2 A birds-eye view of the model

The Neo model describes a hypothetical black box that does not have any I/O peripherals connected to it directly but allows multiple I/O peripherals to interface with it wirelessly. This hypothetical black box is called the Neo device. The Neo Model has two important properties (du Toit & Ellefsen, 2015):

- **The secure container property** (SCP). Data and applications are isolated from other data and applications according to ownership. The isolation of data and applications are called the secure container property (SCP).
- **The mutual authentication property** (MAP). Access and secure connectivity of the I/O peripherals and Neo device are ensured through the Mutual Authentication Property (MAP)

Figure 26 shows the SCP and MAP in relation to the Neo device and the I/O peripherals. The SCP is depicted as a number of containers inside the Neo device. The containers isolate data and applications based on ownership. The MAP ensures I/O peripherals are authenticated with the Neo device and that the Neo device is authenticated with an I/O peripheral. The MAP further ensures that communication between the Neo device and I/O peripheral is confidential.

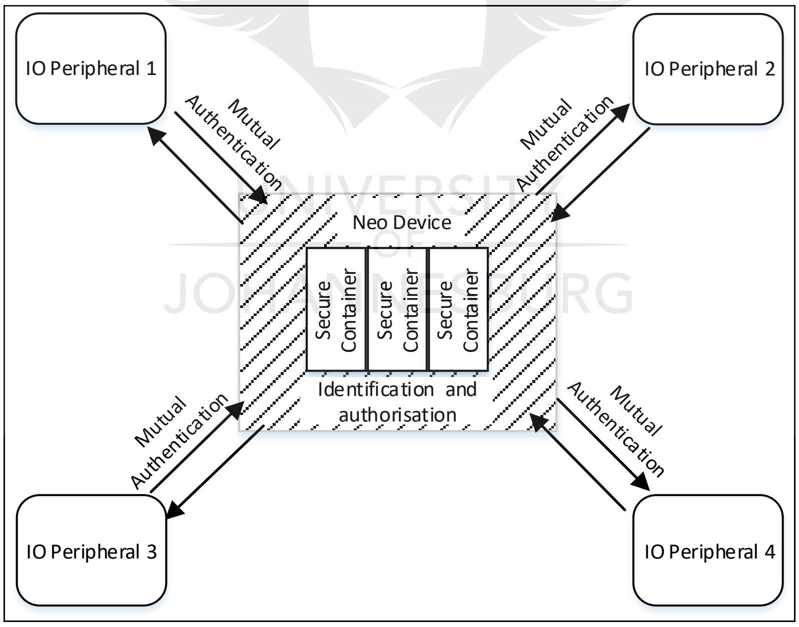


FIGURE 26: A BASIC OVERVIEW OF THE NEO MODEL (DU TOIT & ELLEFSEN, 2015)

The Neo model does not prescribe the implementation of the SCP and MAP. Instead, it highlights the importance of these two properties as an integral concept, in order to address the problem and hypothesis defined in Chapter 1 section 1.3. The SCP ensures that the Neo device can be used securely for both personal and business purposes. The MAP provides the ability to use the Neo device with various I/O peripheral with different form factors.

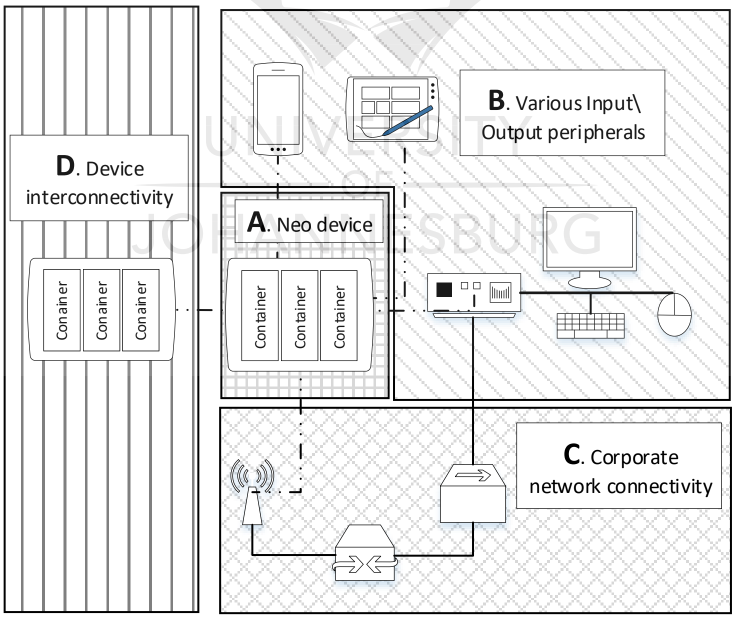
The SCP and MAP are both important properties of the Neo model and are described in more detail in Chapter 8 and Chapter 9, but the model further describes a number of components that are required in order to address a number of concerns.

7.2.1 The landscape of the Neo model

The Neo model also describes the landscape in which a user can use a mobile device for personal and business purposes. The landscape consists of a number of areas. This section describes, in high-level, each of the areas and how the different areas interact with each other.

Figure 27 portrays the landscape in which the Neo model operates. The middle of the diagram depicts the Neo device that connects to various other components. The landscape consists of:

A.) The Neo device. The Neo device is the hypothetical mobile device that can be used for both personal and business purposes.
B.) Various I/O peripherals. Various I/O peripherals can connect to the Neo device to support the use in different areas and situations.
C.) Corporate connectivity. A corporate environment requires special usage and management of the Neo device.
D.) Device interconnectivity. Neo devices can also interact directly with each other.



B. Various Input\ Output peripherals
D. Device interconnectivity
A. Neo device
Conainer
Conainer
Conainer
Container
Container
Container
C. Corporate network connectivity

FIGURE 27: THE NEO MODEL LANDSCAPE (DU TOIT & ELLEFSEN, 2015)

This chapter describes A – D in Figure 27 in more detail. The device around which the model cannot exist is the Neo device.

7.3 The Neo device (A in Figure 27)

The Neo device is a hypothetical mobile device. The nature of the Neo device is that the device is not dependent on any built-in I/O peripheral. Instead, the MAP addresses any I/O peripheral connectivity. The Neo device is also mobile, which means that it is dependent on battery power when used in a mobile capacity. The Neo device, however, can be used in a docking station that enables the device to be powered by an external power source. The power source also charges the battery, which enables the Neo device to be mobile.

This section is organised into:

- **Section 7.3.1** that describes the physical properties of the Neo device and
- **Section 7.3.2** that describes the services that run on the Neo device.

7.3.1 Physical properties

The physical properties of the Neo device are different from existing mobile devices. The problem statement in Chapter 1 describes a scenario where users use multiple mobile devices for business and personal use. The requirement for multiple devices is multi-faceted, but it includes a desire to interact with the mobile device with different form factors and physical properties. This statement is in essence also the secondary objective of this thesis.

To address the secondary objective, the Neo Model defines a device without any physical limitation or size specification for the input and output hardware. Any I/O hardware is removed from the core Neo device and is defined as interchangeable I/O peripheral devices.

Physically the Neo device can be visualised with very few ports. The device has space for an out-of-band (OoB) wire to be connected that is required when authorising wireless I/O peripherals, as alluded to in section 7.2, by the device owner. The device must also have space for a docking station connector or charging cable. Apart from this, an indicator may provide status information about the device.

The Neo device should have the same potential processing capabilities of a desktop computer. The processing capabilities ensure that the device can be successfully used as a desktop computer replacement, which is a requirement for addressing the hypothesis in Chapter 1 section 1.3 and addresses the primary objective of the thesis. The managing of the processing and the power requirements for the processor is beyond the scope of the thesis.

The Neo device has the same capabilities of today's mobile phones. This means that users can use the Neo device for phone calls, but it can also include other components like a wireless network interface card, a global positioning system (GPS), temperature sensors or accelerator sensors.

The Neo model does not prohibit the packaging of the Neo device with seemingly built-in peripherals like a touch screen or digital camera. The model allows this type of packaging to occur, however, the touch screen or digital camera will still access the core Neo device through the identification and authorisation service described in the next section of this chapter.

7.3.2 Management services

Two important services run on the Neo device and are depicted as (A) and (D) in Figure 28. These two services enable the Neo device's side of the MAP and the SCP. The two services are:

MAP2.2

- A **management system** (A). The management system is a special system container that manages device-wide policies and settings. Some of these settings are described in more detail later in this chapter. The management system stores and provides access to authorisation settings. The management system is also responsible for provisioning and controlling secure containers on the Neo device.
- **Identification and authorisation service** (D). The identification and authorisation service provides a number of services. It is firstly responsible for identifying and authenticating I/O peripherals and users to the Neo device, but also identifies and authenticates network connections to containers, where applicable. The service also authorises authenticated objects access to containers. The service also ensures that all communications between the Neo device and any I/O peripheral are confidential.

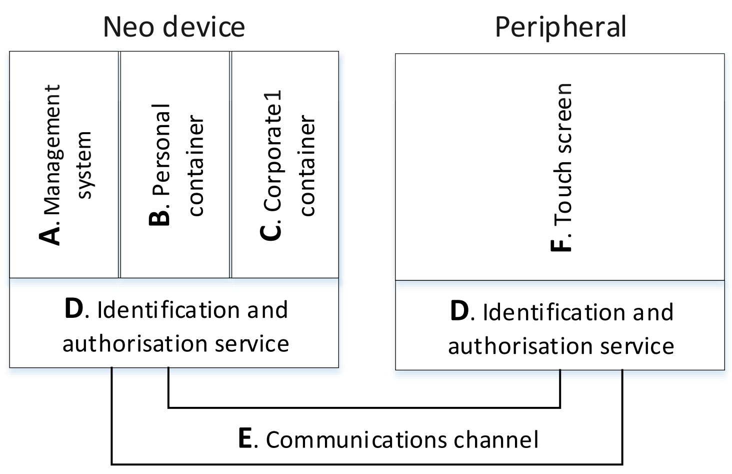

Neo device
Peripheral
A. Management system
B. Personal container
C. Corporate1 container
F. Touch screen
D. Identification and authorisation service
D. Identification and authorisation service
E. Communications channel

FIGURE 28: NEO DEVICE COMPONENTS (DU TOIT & ELLEFSEN, 2015)

Figure 28 describes a Neo device on the left-hand side, with a number of services and secure containers, and an I/O peripheral on the right-hand side, communicating with each other. This figure is used in the following explanation:

When the Neo device is initially switched on, it goes through initial configuration steps:

1. It requires that the owner of the device connect an I/O peripheral (which should contain at least a screen and some type of input mechanism) to the Neo device using the OoB cable and ports. The simplest I/O peripheral that fulfils this requirement is a touch screen, say T. It may also be possible that the Neo device is sold and packaged with a default separate touch screen (T) that allows the user to at least interact with the Neo device (F in Figure 28). The Neo device creates an identity for the I/O peripheral (T) and gives the I/O peripheral (T) access to interface with the Management System on the Neo device.

MAP1.2

2. The user is then prompted to create a user identity with some type of authentication mechanism. The identity and authentication objects are stored in the Management System (A in Figure 28).
3. The Management System automatically provisions an encrypted Personal Container (B in Figure 28) that is only accessible by the initial I/O peripheral and newly created user identity.
4. The Identification and authorisation (D in Figure 28) system ensure any access to the Management System and Personal Container only occurs with authenticated I/O peripheral and user identities.
5. The Identification and authorisation system also ensures all communication between the Neo device and I/O peripheral is automatically encrypted (E in Figure 28).

MAP3.1

6. The initial provision and access to other containers, like a corporate container (C in Figure 28) are handled by the Identification and authorisation system together with the Management System.

The Neo device exhibits specific properties that ensure personal and business applications and data are securely isolated, and access to the isolated data is controlled. The Neo device, however, allows many I/O peripherals to connect to it. These I/O peripherals are described in more detail in the next section.

7.4 Various Input/Output peripherals (B in Figure 27)

The primary objective of the thesis is to design the Neo model to ensure that a user can use the device for both personal and business purposes while minimising the requirement for data to be exchanged between multiple computing devices. In order to support this objective, multiple form-factor I/O peripherals should be able to connect to the Neo device.

It has already been argued that users may prefer multiple mobile devices for their ergonomic nature. The Neo model does not prescribe specific size or type of I/O peripherals. The Neo model prescribes I/O peripherals with basic computing capabilities that authenticate and authorise connections to and from Neo devices. This means that there is mutual authentication between a Neo device and an I/O peripheral.

Mutual authentication is required because I/O peripheral owners get the assurance that their I/O peripherals are connecting to only authorised Neo devices: When the Neo device is used in a corporate environment, the Neo device can be owned by the user, but the potentially wireless I/O peripheral that stays in the user's office, belongs to the business.

The business needs assurance that when a user types sensitive information on an I/O peripheral that the data is only transmitted to an approved Neo device and container. Without this mutual authentication, it may be possible for an attacker to hijack the connectivity from an I/O peripheral and capture login details or other sensitive information typed on the I/O peripheral on a rogue Neo device.

MAP1.1

The above requirement makes it difficult for an I/O peripheral that exists today to be used with a Neo device. The I/O peripherals need their own identification and authorisation service to ensure mutual authentication. Chapter 9 elaborates on this principle.

The Neo model does not limit the type of I/O peripherals, as long as it can interface with the Neo device wirelessly or through the docking port. This means that I/O peripherals can include I/O peripherals such as:

- Touchscreens. These peripherals allow the user to interact with the Neo device for both input and output.
- Docking stations with their own mouse, keyboard and monitors can be used in cases where users like to sit at a desk or kiosk-type utilities.
- Wireless big screens or projectors for entertainment purposes or presentations.
- Wireless speakers or earphones for audio or private audio.
- Printers to ensure that users can output data to paper.
- Microphones that ensure users can make phone calls or record information on the Neo device.

The Neo model ensures that different people can use the Neo device for different purposes using various I/O peripherals with different form factors. This versatility ensures that users can use the Neo device for both personal and business purposes, minimising the requirement for exchanging personal or business data between different devices.

The next section describes how the Neo model fits into a corporate environment, ensuring control by the business over its own data without compromising personal user information and data.

7.5 Corporate connectivity (C in Figure 27)

The Neo model ensures that the Neo device can be used for both personal and business purposes. Interfacing the Neo device into a corporate network or system requires special attention since the device potentially does not belong to the corporate, but needs access to corporate applications and data.

Corporate connectivity from the Neo device does not only include aspects such as how the physical or wireless connection should be achieved, but it also addresses aspects such as:

- Identification and authentication of Neo devices by the corporation.
- Identification and authentication of users by the corporation.
- Authorisation of users and devices on the corporate network and data.
- Confidentiality of corporate data on Neo devices.
- Confidentiality of corporate data being transmitted between Neo devices and the corporate network.

The identification and authentication, as well as authorisation aspects, are described in more detail in Chapter 9 with the confidentiality aspects discussed in more detail in Chapter 8 and Chapter 9.

From a high-level perspective, Neo devices connect to a corporate network, either through a docking station or through a special gateway device, called a Gateway Controller. Figure 29 summarises the services provided by the Gateway Controller. It shows the Neo device on the left-hand side connecting through some sort of wireless interface to the Gateway Controller that manages access to the company network and resources. The Gateway Controller acts as a guard that provide the following services, as depicted in Figure 29:

A.) **Identifies and authenticates** Neo devices.
B.) After authentication, the Gateway Controller **authorises** access to certain network resources, described as network zones and network services.
C.) The Gateway Controller and the Neo device automatically **establishes encrypted channels** between the Gateway Controller and the approved secure container in the Neo device.

D.) The Gateway Controller challenges the user for authenticating a **corporate user identity** as soon as the device connects to the Gateway Controller.

E.) The Gateway Controller also provides an interface to the corporate **mobile device management** (MDM) system that enforces **policies**, on the Neo device. More details on how this is accomplished are described in Chapter 8.

SCP3.2

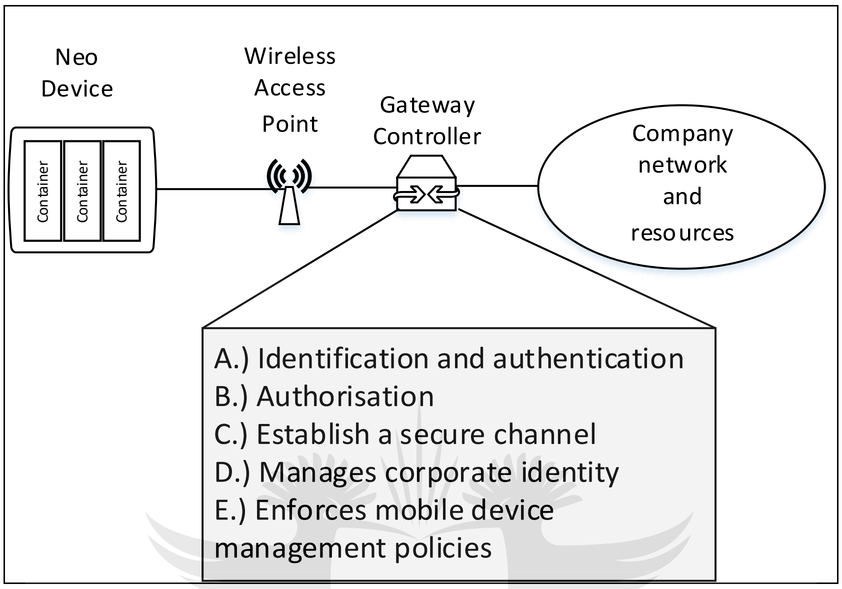

Neo Device
Container
Container
Container
Wireless Access Point
Gateway Controller
Company network and resources
A.) Identification and authentication
B.) Authorisation
C.) Establish a secure channel
D.) Manages corporate identity
E.) Enforces mobile device management policies

FIGURE 29: SERVICES PROVIDED BY THE GATEWAY CONTROLLER (BY AUTHOR)

The Neo model ensures that the Neo device can be safely used for both personal and business purposes. The Gateway Controller acts as an interface to the corporate network and ensures that the corporate can control use aspects of the Neo device when the corporate container is activated on the Neo device.

Apart from the fact that corporates can fully control their data and applications on the Neo device, the Neo device also allows users the freedom to connect and interface their personal container data and applications with other Neo devices.

7.6 Device interconnectivity (D in Figure 27)

The Neo model allows users to share data or applications in their personal container with other Neo device owners. This feature allows users to share and connect various applications with other users. This may include, but is not limited to, the sharing of photos, documents or games.

The sharing of personal data with other people frequently occurs in mobile device environments. The requirement to share personal data does not only happen between one person and one other person but frequently users need to share information between groups of people.

The Neo model allows users of personal containers to share data or connect applications to not only one person, but also multiple people. This allows users to share a collection of pictures to a number of family members for example or allow multiple users to play the same game.

As soon as a user decides to share or connect their application or data with another user on a Neo device the following process is involved. The Neo device uses temporary containers to store applications or data when it interfaces with other devices. Figure 30 shows the end-state of this process from a high-level, with User A's Neo device depicted on the left-hand side and User B's Neo device on the right-hand side:

SCP6.1

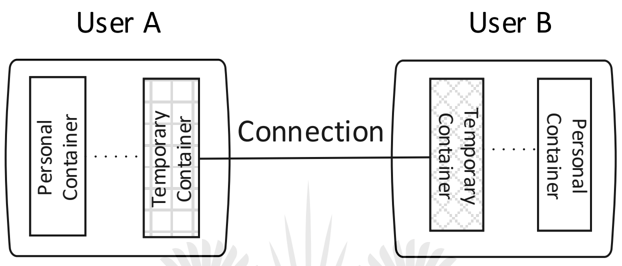


FIGURE 30: SHARING OF DATA OR APPLICATIONS (BY AUTHOR)

A.) User A selects the data or application from the personal container and selects the share-activity.

B.) The Neo device interrogates the data or application to determine the data interchange mechanism that is required in order for the sharing to succeed. Example: Sharing a game may require a specific service from the game application to be exposed, whereas sharing of pictures require the data files of the pictures to be exposed.

C.) User A's Neo device creates a semi-temporary secure container and creates a copy of the data or application into the semi-temporary secure container. Depicted on the left-hand side of Figure 30.

D.) The I/O peripheral of User A's Neo device and User A's user account is automatically allowed access to the newly created secure container.

E.) User A authorises User B access to the temporary container by physically touching the two devices together.

F.) User B's Neo device gets a sharing notification with information about the data or application that is being shared and interrogates User B's Neo device to ensure the user has an appropriate application that can interface or open the shared data.

G.) User B is prompted to allow data to be received or interfaced and waits for User B's approval. As soon as it is approved, User B's Neo device also creates a temporary container and creates a copy of the appropriate application into the temporary container.

H.) The I/O peripheral of User B's Neo device and User B's user account is automatically granted access to the semi-temporary container.
I.) User A's temporary container is allowed access to User B's temporary container and vice-versa.

This whole process is managed by the Management System introduced in Section 7.3.2 and interfaces with the identification and authorisation service which manages the identification, authentication, authorisation and confidentiality of the communications. These aspects are described in more detail in Chapter 8 and Chapter 9.

The reason why this section describes these temporary containers as semi-temporary is that users can decide whether the sharing should be a once off or a more permanent sharing of information. In cases where the sharing is only once off, the temporary container is removed from the system as soon as the connection to the other Neo device is broken. In cases where the sharing is set up for a longer period, the temporary container stays active and allow connections as long as the user decides to keep sharing the data or application.

The Neo model allows users to share data and applications with other users. The sharing of data or applications occur in their own secure temporary containers. Access to the temporary containers is based on the sharing preferences of the user.

7.7 Conclusion

This chapter introduced the Neo model and the highlighted the secure container property (SCP) and the mutual authentication property (MAP) that underpins most of the model's design and landscape. The chapter introduced the SCP as an aspect that ensures data and applications are grouped together and isolated from other data and applications based on their ownership. The MAP decrees that any connection by an I/O peripheral to the Neo device must be identified and authenticated on both the I/O peripheral and the Neo device.

Section 7.2 provided a high-level overview of the Neo model. In this section, the SCP and MAP were briefly introduced. The section further described the changes to the computing landscape that the Neo model introduced.

Section 7.3 defined the Neo device that forms the core of the Neo model. The section described the Neo device as a device without any built-in I/O peripherals to allow different form-factor I/O peripherals to connect to it. The section further described the software services that ensure the safe interaction and management of various computing environments and I/O peripherals.

Section 7.4 provided an overview of the I/O peripherals that interfaces with the Neo model. Container owners control I/O peripherals and Neo device mutually authenticates each other and ensure access to the different containers on the Neo device.

Sections 7.5 and 7.6 introduced the idea that the Neo device needs to connect to corporate networks, but also to other Neo devices. Connectivity to corporate networks is required for users to use the device for business purposes. Connectivity to other Neo devices is required when users would like to share personal data with other users and want to make use of built-in Neo device functionality instead of cloud or application level security.

The next chapter describes the SCP in detail and sees how the SCP is enforced in the model but still allowing connectivity to corporate environments and other devices.

Introduction

- Chapter 1 Introduction

The end game

- Chapter 2 Model Overview

Literature study

- Chapter 3 Isolation properties in mobile operating systems (SCP)
- Chapter 4 Isolation techniques in other technologies (SCP)
- Chapter 5 An overview of peripheral security (MAP)

Bridge

- Chapter 6 The drivers for a new model

Building the Neo model

- Chapter 7 Overview of the Neo Model
- **Chapter 8 The Secure Container Property**
- Chapter 9 The Mutual Authentication Property
- Chapter 10 Container life cycle
- Chapter 11 Novel Implementations

Evaluation and summary

- Chapter 12 Potential implementation of Neo Model
- Chapter 13 Conclusion and future work

Chapter 8 THE SECURE CONTAINER PROPERTY - DESIGN

8.1 Introduction

The Neo model defines two important properties. These properties, described in Chapter 7, are the secure container property (SCP) and the mutual authentication property (MAP). The SCP is an important property of the Neo model that addresses the problem highlighted in Chapter 1. In fact, the SCP fulfils the primary objective of this thesis that states that the model should isolate and protect the data and applications based on ownership and owner-defined policies.

The term provisioning is used in this section no describe the creation and configuration of specific containers.

This chapter describes the SCP in more detail:

- A definition of the secure container is provided in **section 8.2**.
- After the definition of the secure container, the various classes of containers are described in **section 8.3**.
- **Section 8.4** describes how the containers and the information in the containers are kept confidential.
- **Section 8.5** describes why and how new containers are provisioned.
- **Section 8.6** describes how a data or application owner can implement policy-based controls on the secure containers.

The two tables, Table 11 and Table 12 in Appendix A, can again be used to correlate with the various callouts in this chapter.

The component in the model that directly addresses the SCP, is the various secure containers found inside the Neo device. The next section defines the principle of the secure container in more detail.

8.2 The secure container – defined

The SCP is one of the two properties that define the Neo model, originally defined in section 2.3.1 page 21. To understand the role of the secure container the principle of owner-based data and applications must be understood. The secure containers keep data and applications isolated based on their ownership.

SCP2.2

When a user buys a smartphone or tablet computer, the user expects to have full control over the applications that can be installed on the device and establishes ownership over the data generated or managed by these applications. This data is mostly private in nature and users are understandably hesitant to allow other people access to this data.

Users may choose to share some of their private data. The decision to share the data with other users stays with the owner of the device. This private data is owned by the user.

In the case of a business. Data generated by the business, such as business email, documents or line-of-business applications belong to the business. In this case, the business has a stake in this data and constitute part of its knowledge capital. The business applications themselves can also be owned or the business owns the licenses for the specific applications. Businesses need to have full control over how these applications are used and who has access to the data.

All applications and data are isolated based on their ownership in the Neo model. Data and applications belonging to the same owner are grouped together and isolated from data and applications belonging to another owner. The level of isolation is very strong. This means that data or applications from one owner cannot access data or applications belonging to another owner.

SCP3.1

The strong isolation of data and applications is defined as a secure container. Each container hosts both the data and the applications belonging to a specific owner. Each container has also a set of operating system interfaces for these applications with a set of abstracted peripherals such as:

- Network interface
- Input device interfaces, like keyboards, mouse.
- A framebuffer interface for screen output.
- A graphical processing unit.
- Sensor devices, like GPS, accelerometers, thermometers etc.
- Radio interface unit (For cellular telephony purposes).

A formal definition for the secure container can now be defined as:

An owner controlled computing environment, with abstracted peripherals, strongly isolated from the rest of the computing environment that hosts applications and data, based on ownership.

The above data isolation and device abstraction mean that each secure container is a mini operating system with its own apps and data. The Neo model requires strong isolation mechanisms that can be defined by hypervisor level isolation. The next section shows that there may be different types of secure containers, depending on what they are used for in the Neo model.

SCP1.1

8.3 Container classification

Now that the secure container is defined, by virtue of what they are used for, these containers do not only store user or business data and applications. The Neo model identifies three classes of secure containers, depicted in Figure 31 and described below:

A.) **Data and application secure containers**. These containers host data and applications based on their ownership (**Section 8.3.1.**).

B.) **A management system container**. The management container manages the overall working of the Neo device and ensures secure containers are available when needed (**Section 8.3.2.)**.

C.) **Temporary secure containers**. These containers host temporary data and applications for sharing purposes (**Section 8.3.3.)**.

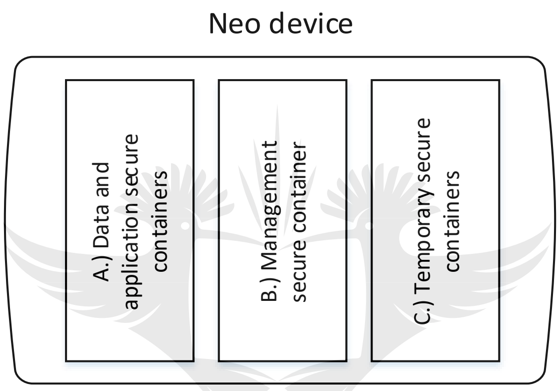

Neo device
A.) Data and application secure containers
B.) Management secure container
C.) Temporary secure containers

FIGURE 31: CLASSES OF SECURE CONTAINERS (BY AUTHOR)

Each of these container classifications is described in more detail in the following sections.

8.3.1 Data and application secure containers

In a computing environment, a specific identity or affiliation can install applications. Data generated by these applications belong to the identity or affiliation. The identity or affiliation is the owner of the applications and data. These data and applications are hosted in a secure container belonging to the specific owner (A in Figure 31).

The first time the owner switches on the Neo device, a personal identity for the owner of the Neo device is created, with a default personal container. This means the Neo device always has at least one secure container that belongs to the personal identity of the device's owner. It is also possible that the Neo device can have multiple users. Each user has its own secure container on the Neo device. This means the Neo device may have secure containers for identities managed by itself.

When the Neo device is used for business purposes, a secure container is created for the business identity of the user. This business identity and the secure container have the business as the owner. This means that the business has full control over who has access to the data and applications stored in the container. Furthermore, the business can also create policies that influence the behaviour of the container. Section 8.6 of this chapter, describes the policies created by the business.

The provisioning of a business container does not happen at the discretion of the device owner, instead, the business connects the Neo device to the business network, through a provisioning process. During the provisioning process, the business' secure container is created, together with the business user identity, on the Neo device. Section 8.5 of this chapter describes this process in more detail.

A business can also create other containers that can only be used in special circumstances. Sometimes businesses require different levels of data security. This means data and applications with a high level of sensitivity or confidentiality can be controlled and managed in its own secure container. Chapter 11 of this document describes some of these use-cases in more detail.

SCP2.1

More than one application is available inside a data and application secure container. Communication of applications inside the same container occurs through normal operating system interfaces.

Data and application containers host data and applications based on their ownership. There is, however, a special container in the Neo model, that exists on each Neo device, that does not have the user as the owner of the information stored in the container. This container is known as the management system container.

8.3.2 The management system container

The management system container (B in Figure 31), plays a vital role in the management of the Neo device and the secure containers. The management container is owned by the system. No user has direct access to the management container. Users interact with the management container through well-defined and controlled interfaces. The management container provides the following services:

A.) **Stores all user and business identities and authorisation secrets.** The management container store and maintain the various owner identities as well as their authorisation secrets. This concept is the same as the user accounts database that exists on modern operating systems. The user accounts database stores the user's identity as well as the authentication secret for each identity. The secret can be some form of a password, but can also include other authentication mechanisms like biometric secrets.

B.) **Stores all I/O peripheral identities and peripheral authorisation secrets**. Any authorised I/O peripheral's identity is stored in the management system container. Before any I/O peripheral can connect to the Neo device, the I/O peripheral must identify itself with the Neo device and provide a shared secret to authenticate the I/O peripheral. The identities and shared secrets for the different I/O peripherals are also stored in the management container. Chapter 9 section 9.4 describes the identification and authentication in more detail.

C.) **Stores the encryption and decryption keys for the various secure containers**. All the secure containers are automatically encrypted on the device. Each container has its own set of keys that enable the encryption and decryption of the container. The storage of these keys can be either in the Neo device or on a Gateway Controller. In cases where the keys are store on the Neo device, the management container stores these keys. Section 8.4 of this chapter describes how these keys are used to enable confidentiality of the secure containers on the storage media (This is also referred to as data encryption at rest).

MAP3.2

D.) **Stores the business-defined policies that apply to the different containers**. When the Neo device is provisioned for business purposes, the business creates policy definitions that control the Neo device and containers on the Neo device in certain circumstances. These policies are supplied by the business and stored in the management container. Section 8.6 provides more detail of the policies and container control.

E.) **Stores the access control rules for the different containers.** Connections from I/O peripherals to secure containers are controlled by the Identification and Authorisation service. The access control rules used by the identification and authorisation service are stored and maintained in the management container. The access control rules are described in more detail in Chapter 9 section 9.5.

F.) **Hosts the identification and authorisation service.** The management system is the operating system that runs the identification and authorisation service for various functions introduced in Chapter 7 section 7.3.2 and discussed in more detail in Chapter 9.

The management system container contains critical operating system services and data. An integrity code for each of the critical operating system services is stored in a ***service list file***. The service list file is encrypted with the Neo device's private key (Section 8.4.1 elaborates on the different keys used in the Neo model) to ensure it cannot be modified from within the management system container.

During the device start-up, the Neo device verifies the integrity of the critical operating system services, by hashing each critical operating system service and verifying the resultant hash with the hash stored in the service list file. Only after the integrity of the critical operating systems services has been verified, is the critical operating system services started.

SCP4.3

SCP4.1

To minimise the attack surface of the management system container, direct connections to and from any I/O peripherals or outside network connections are not allowed. Updates to information in the management system container are managed through a temporary container, described in more detail in the next section. Any network connections or network services run in a temporary secure container that gets created during device startup and destroyed during device shutdown. These temporary containers act as a protective buffer for the management system container, but also for other containers.

8.3.3 Temporary secure containers

A temporary secure container is a container that is not stored permanently on the Neo device. These containers are normally created and destroyed depending on their role. In some circumstances, temporary containers can be saved for future use. These temporary containers can be viewed as demilitarised network zones, that helps protect the secure containers from network attacks.

Temporary containers are used with the management system container and with data and application containers.

8.3.3.1 Management system temporary container

SCP4.4

The management system temporary container protects the management system container from outside networks. The management system temporary container also sanitises input from input peripherals meant for the management system container.

When a business provisions a corporate container, the business must send data for the container across the network to the Neo device. During this provisioning process, the business data and code structure of the business container is sent to the management system temporary container. The business data and code structure are directory information, extra services and application configuration beyond the container framework as described in section 8.5.

As soon as all the data has been sent to the temporary container, the management system ensures that the business data and code structure adheres to the accepted format of how the containers are implemented. Once the management system has verified the format, the management system moves the newly created container onto the Neo device.

SCP4.2

The temporary container also gets the identifiers, secrets, initial access control rules and policies from the Gateway Controller. As soon as these objects have been created in the temporary container, the management system does a sanity check on these objects and move them for permanent storage into the management system container.

In cases where the Neo device has to communicate with devices like a Gateway Controller or other Neo devices, objects like identity information and existing I/O peripheral information are stored in the temporary container from where it is communicated to a receiving Gateway Controller or Neo device.

The management system temporary container is always active when the device starts but gets destroyed when the device is switched off. The data and application temporary containers are used

8.3.3.2 Data and application temporary container

Temporary containers for data and application security containers has two configurations. The first configuration is applicable to all data and application security containers. The second configuration is only applicable when data or applications are shared between Neo devices.

8.3.3.2.1 Configuration one. Firewalling temporary containers.

Configuration one temporary containers apply to all data and application security containers. In this case, the temporary container acts as a firewall virtual machine. Configuration one temporary containers ensure that any network traffic directed to a data and application secure container must first pass through a firewall, with firewall rules updated by the identity and authentication service.

SCP6.2

The configuration one temporary container also ensures that the data and application containers are not directly connected to potentially untrusted networks. Configuration one temporary containers only provide indirect access to network traffic. Interfacing with I/O peripherals is still managed by the management system container.

SCP6.1

Configuration one temporary containers may be implemented in one to one configuration or a one to many configurations. The one-to-one configuration ensures that there is at least one configuration one temporary container for each data and application container. The one-to-many configuration ensures that multiple data and application containers can share the same firewalling temporary container.

The one-to-many configuration requires fewer system resources. The firewall rules and routing that occurs in the configuration one temporary firewall ensures that network traffic is automatically routed to and from the correct data and application firewall.

8.3.3.2.2 Configuration two. Data sharing temporary containers.

Configuration two temporary containers are used when data or applications are shared between Neo devices or when external storage is connected to the Neo device. A temporary container is created when a user decides to share information or applications from the personal container to another user's Neo device or to and from external storage.

Only the owner of a container has the rights to share data. This means that data from a corporate container cannot be shared with other container owners, or external storage without the corporate allowing this action.

When an owner decides to share data with another user, the management system provisions a temporary container for the set of data that the user wants to share with another user. The container that stores the actual data, called the master container, interfaces with the temporary container and copies the data to the temporary container.

The interface only exists between a master container and its equivalent temporary container. These interfaces do not exist between other containers. The interface is also only initiated from the master container and never from the temporary container. This means that processes running inside the temporary container have no access back to the master container.

The owner of the data controls access to the temporary container, thereby controlling access to the shared data. A Neo device with one master container can also have more than one temporary container. This allows the same data to be shared with multiple parties in different temporary containers. A synchronisation service in the master container is responsible to ensure that data stays synchronised between the master container and the other temporary containers.

The synchronisation service is beyond the scope of this thesis since there are already many synchronisation solutions and models that enable data and application synchronisation (Bernstein & Goodman, 1981; Corbett et al., 2013; Chang et al., 2008).

The sharing mechanism does not allow external storage to directly interface with data and application containers. When files are copied from external storage to the data and application container, the temporary container allows the user to interrogate the files first before approving the files to synchronise back into the data and application container.

SCP5.2

All data and application containers use temporary containers, but also when users want to share data with other Neo device owners from their personal containers. The management system also uses a temporary secure container as a DMZ to ensure the management system container is never directly exposed to a network. The next section describes how data and applications inside the container are kept confidential while at rest.

8.4 Container confidentiality

Data owners that do not own the Neo device has a valid concern for the safety and confidentiality of their data when it is used on a device not owned by them. This is specifically a concern for businesses when they allow employees to use company data and applications on devices not owned by them.

To minimise the risk of unauthorised access to data the Neo model forces the Neo device to encrypt all containers on the Neo device. Access to the secure containers is controlled by the management system and the identification and authorisation service.

There are many solutions available today that explains how encrypted files systems can be mounted and is outside the scope of this document. What is important to note from a Neo model perspective is to understand which keys are used for encryption and decryption, as well as how these keys are kept secure for the various secure containers.

The following sections describe the security keys in more detail:

- **Section 8.4.1** identifies the keys used by the Neo model for secure container encryption and decryption.
- **Section 8.4.2** describes how the keys are stored to minimise unauthorised access to the keys.
- **Section 8.4.3** discusses the key lifecycle, from creating to destroying keys.

8.4.1 Encryption and decryption keys

The Neo model makes use of both symmetric key encryption and asymmetric key encryption. Symmetric keys and algorithms are used to encrypt and decrypt any container on the Neo device, while the asymmetric keys and algorithms are used to secure the symmetric keys.

This means, for each container on the Neo model a symmetric key is created, that encrypts and decrypts the container. A public key is used to encrypt the symmetric key and a private key to decrypt the symmetric key. Each container has its own unique public-private key-pair. The creation of the keys are described in more detail in section 8.4.3.

Figure 32 describes a personal container and a corporate container:

- The personal container is encrypted and decrypted using symmetric key 1.
- The corporate container is encrypted and decrypted using symmetric key 2.
- To keep symmetric key 1 secure, symmetric key 1 is encrypted using public key 1.
- When access to symmetric key 1 is needed, private key 1 is used to decrypt symmetric key 1.

- Symmetric key 2 is used to encrypt and decrypt the corporate container and is in its turn encrypted and decrypted using public key 2 and private key 2.
- In some instances, private key 2 can also be encrypted using a company public key and decrypted using a company private key.
- The company private key is never stored on the Neo device.

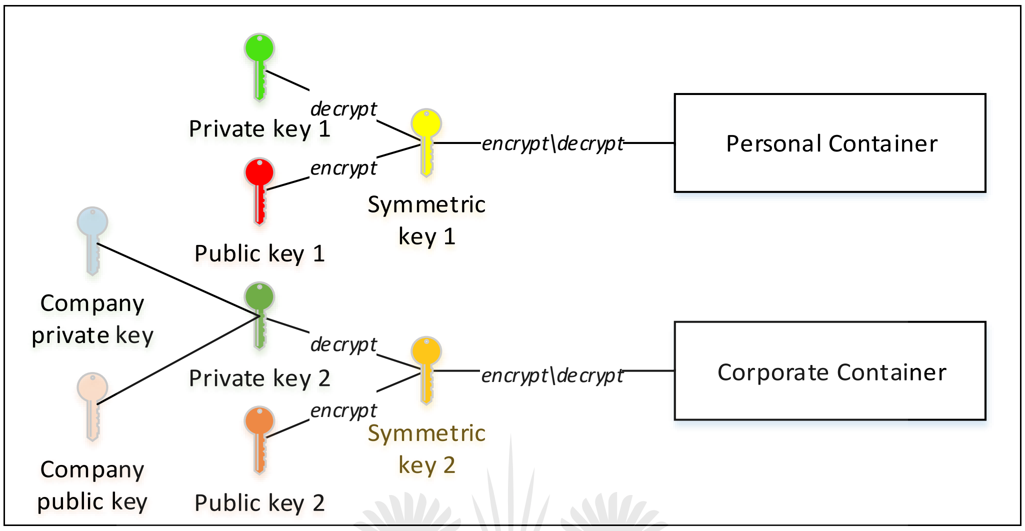


FIGURE 32: KEYS USED FOR CONTAINER ENCRYPTION

Apart from each container having its own set of keys, the Neo device also contains a set of public and private keys that are used for various purposes. A summary of the categories of keys used by the Neo model for at-rest cryptography are:

- A public-private key-pair for the Neo device.
- A set of keys for each container. The set of keys include a pair of public-private keys and a symmetric key.
- A set of asymmetric keys used by a company.

To ensure container confidentiality, the Neo model defines a number of keys that are used to encrypt and decrypt the various secure containers. The Neo device also has a set of keys that are used for accessing some of these key sets and is explained in more detail in the next section.

SCP5.3

8.4.2 Key storage

The various keys used by the Neo device is stored in one of two areas. The two areas that can store keys is in the Trusted Platform Module (TPM) on the Neo device and the management system container. The keys managed by the TPM are used for device start-up and to decrypt the management system container. The TPM managed keys are the public-private key-pair used by the Neo device for storage and integrity purposes.

All the other secure container keys are stored in the management system container. When a specific container starts up the Identity and Authentication service requests that the management system mount the encrypted container given the keys stored in the management system container.

Figure 33 shows the logical storage locations of the various keys mentioned in 8.4.1. The public and private keys used to encrypt and decrypt the management system symmetric key are managed by the TPM (A in Figure 33). The physical storage of these keys are most likely implemented as an encrypted key storage located on the storage device of the Neo device, that is unlocked by the storage root key (SRK) inside the TPM (Arthur & Challener, 2015). Section B of Figure 33 is a representation of the management system container and the various keys that can be stored inside the management system container. Section C in Figure 33 is a representation of the company private key that is not stored on the Neo device, but is only used in cases where the policy of the container disallows offline use.

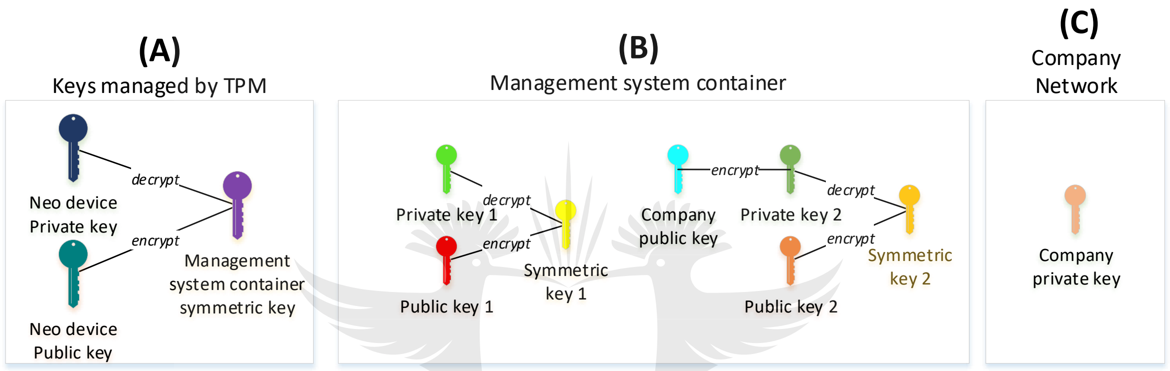


FIGURE 33: KEY STORAGE LOCATIONS (BY AUTHOR)

In some instances, some keys are not stored on the Neo device. In the case of a corporate container, where the business only allows the corporate container to become active while connected to the Gateway Controller, a special company private key is used for decrypting the private key of the container. The container private key (depicted as Private key 2), is encrypted using the company public that is also stored in the management system container. The Neo device requests the private key of the container to be decrypted by the Gateway Controller using the company private key. The company private key never leaves the confines of the company and the encrypted private key on the Neo device stays in an encrypted format in the management system container.

This means that the corporate container cannot start or work while the Gateway Controller is not available. If the connection to the Gateway Controller is lost while using the corporate container, the management system dismounts the corporate container and encrypts the symmetric key with the container public key. The identification, authentication and confidentiality of communication sessions are described in more detail in Chapter 9.

8.4.3 Key lifecycle

The challenge with encryption and decryption keys are that they may be compromised. In cases where keys are compromised, new keys may be required. This section describes when and how keys are created and what happens when keys have to change.

Any of the symmetric keys used directly in the encrypting and decrypting of containers are created during container provisioning. This means the symmetric key used to encrypt and decrypt the management system container is created by the TPM when the device is started for the first time.

The Neo device generates the symmetric keys used for the other containers during the provisioning of the specific container. This means the symmetric key used to encrypt the initial private container for the owner of the Neo device is generated when the user creates the initial user identity that then provisions the private container. The Neo device creates symmetric keys for temporary containers or corporate containers during the provisioning phase of new containers.

The symmetric keys for the secure containers are destroyed when the secure containers are removed from the system, either by the Neo device owner or remotely by the corporate management system.

Initial asymmetric keys are created during container provisioning. The Neo device itself periodically changes the asymmetric keys used to encrypt and decrypt the symmetric keys. When a new key-pair is generated, the corresponding symmetric key is decrypted using the old private key and re-encrypted using the new public key. In cases of corporate online policies, the newly created private key is encrypted using the company public key.

The owner of a container can also force a change in asymmetric keys. Owners may want to update asymmetric keys in cases where keys have been compromised. In cases where a corporate private key is lost, the corporate must first provide the old private key to decrypt the corporate container before a new asymmetric key pair can be issued to encrypt and decrypt the symmetric key.

The secure containers in the Neo model ensure that no other container can get access to the data and applications inside the container. Each container is encrypted while at rest on the Neo device. This provides a level of assurance to the data owner that the data and applications hosted inside the secure container can only be used inside an approved Neo device. In the next section, the hosting environment of each container is described and how the containers are provisioned.

8.5 Container provisioning

Secure containers can be quite dynamic in nature. Temporary containers are created and destroyed as necessary. Businesses create and enable containers on Neo devices when a

Neo device is provisioned on the business network. Because of this dynamic nature, containers should be easy to create and to destroy.

Not only should containers be easy to create, they should also adhere to a certain standard to ensure they comply with the rules as described by the Neo model. To assist with the creation of secure containers, the Neo device manages and updates a set framework for a secure container that can be easily copied and modified as necessary.

The secure container framework is effectively a pre-defined directory structure with the minimum services required by a secure container. The framework also includes pre-defined abstracted or virtualised devices that can easily be enabled or disabled as required.

The implementation of this framework is outside the scope of this document. The Neo model defines the basic requirements of this framework as the following components:

1. A set directory structure.
2. The minimum set of services such as:
    a. A synchronisation service. The synchronisation service is used between master and temporary container to synchronise data and applications.
    b. A virtual private network service. The virtual private network service is used in corporate containers to enable secure network communications from inside the container to the end-point on the corporate network.
    c. Application provisioning service. The application provisioning service interfaces with the applicable app-store to either manually allow the owner to install an application or automatically install an application.
3. Abstracted or virtualised devices. Each container should already have the abstracted representations of devices created in them that only needs to be linked to the physical devices when the container is provisioned for use.

The framework is stored as a template on the Neo device that is copied as soon as a new container is required. The copied template is then modified by the management system to adhere to the requirements of the requestor. This means the requestor may request a number of applications to be available in the container. In these cases, it is the requestor's responsibility to supply the applications to be installed inside the container during container provisioning.

SCP5.1

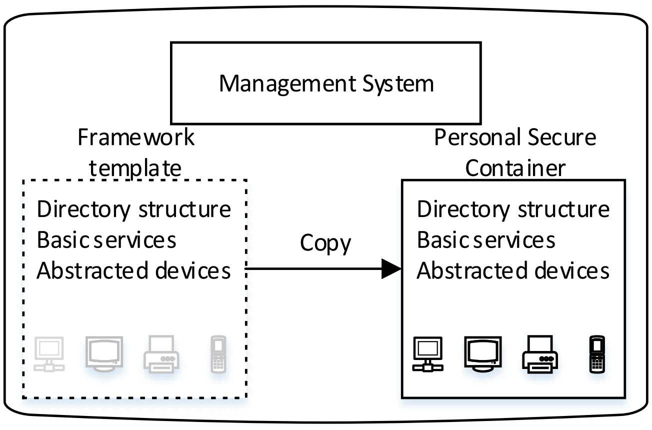


FIGURE 34: CONTAINER PROVISIONING (BY AUTHOR)

Example: Figure 34 refers. The first time the Neo device is switched on, the management system prompts the user to create a new identity. The user completes the required information to create a user identity on the Neo device. The management system creates a copy of the framework template and reconfigures the copied template as the user's personal secure container. The management system enables the abstracted devices inside the newly created secure container and links them to the approved I/O peripherals, given the appropriate access control rules. The application provisioning service did not get any specific instructions and automatically links to a vendor approved application store, allowing the user to install applications as required.

SCP2.3

The provisioning process occurs every time the Neo device requires a new container. This can be when a new user is created on the Neo device, but also when the Neo device is provisioned on a business network. Corporate data owners require control over the data and applications created in the business container that was provisioned. The control aspects of the secure containers are described in the next section.

8.6 Container control

In the Neo model, data and applications are isolated in secure containers, where the data and applications all belong to the same owner. One problem that arises from this scenario is that the device owner, may not be the owner of the secure container. How can the secure container owner be assured that the data and applications are only used according to business rules and policies?

In order to address the above concern, the Neo model allows container owners to define a number of policy rules that would apply on the secure container, but in some instances, it can also control certain aspects of the Neo device itself. This section describes the policy rules and describes how they will affect the container and device. The policy rules are defined and implemented as a set of container policies.

SCP3.2

Container policies can only be defined by businesses and are provided by the Gateway Controller to the Neo device. The container policies are stored in the management system container. The management system container also evaluates the policies and enforce the policies on the Neo device and relevant data and application containers.

SCP2.3

Device owners cannot set policies. The Gateway Controller defines access control rules and container policies. The access control rules are described in more detail in Chapter 9.

Policy rules are categorised into six categories (du Toit & Ellefsen, 2015):

1. **Exclusive use**. The exclusive use policy states that the secure container should be the only container active while a specific rule evaluates to true. The rule objects are described in more detail in 8.6.1. What this policy allows a company to do is to ensure that only the corporate container is active while the user is on the business premises or while the user is in a specific area of the business premises. This ensures that the user cannot capture potential business information using the personal container or a business container belonging to another company.
2. **Device Features**. Device feature policies enable or disable certain device features inside the container. Example: The business can choose to disable the camera feature from the business container. This means that the user will not be able to use the camera for business purposes.
3. **Approved applications**. The approved application category lists the applications that are allowed to run while the ruleset is active. One or more than one application can be assigned as approved applications. If the application is not in this list, then the application is not allowed to run. The approved applications act as a whitelisting of applications.
4. **Denied applications**. The denied applications list the applications specifically denied to run while the ruleset is running. The denied applications always override the approved applications. This means that an application is only allowed to run if it is in the allowed applications category and not in the denied applications category. The denied applications, act as a blacklisting of applications.
5. **Application policies**. Through the application provisioning service, the business controls which applications are available and required in the corporate container. Furthermore, the configuration of the applications is controlled by these policies. Applications have to be written to support application policies. The application policies allow businesses to pre-configure applications without user intervention. Example: The email application can be remotely configured to the addresses and information of the email server, without user intervention.
6. **Offline use**. The offline use policy enables users to use the corporate container while not directly connected to a Gateway Controller. This allows employees to

use their applications while travelling without any network connectivity. The offline use policy only verifies connectivity to the Gateway Controller. This means that a user can still use a corporate container online, even if the user is not physically at the office.

7. **Location activation.** The location activation category may store zero or more zone identities or GPS boundaries. When the Neo device detects that it is inside this location, it automatically enables the secure containers, to whom the policy applies.

Policy rules defined for exclusive use expose a number of policy objects around which a rule-set can be defined. The next section describes the rule objects in more detail.

8.6.1 Policy rule objects

Policy rules for exclusive use can make use of a number of objects that defines a specific scenario. The objects can be extended for future use, but there is a close resemblance to these objects and the Remote Authentication Dial-In User Service (RADIUS) defined objects:

1. **Date and time range**. The date and time object defines a date and time range. This means that a rule can define the date and time and ensure exclusive use is only available during a specified date or time range.
2. **Neo device identity**. A specific rule object can include the Neo device identity or group of identities. This ensures rules are defined for groups of Neo devices.
3. **Corporate user identity**. The corporate user identity is the user identity that has access to the corporate container. The rules can control policy behaviour according to who uses the secure container.
4. **Zone identity**. A zone is a specific area, defined by the network administrator on the corporate network. A zone identity is X.509 certificate published for a specific zone. When the Neo device is in a specific IP address range the Neo device can determine the zone identity as described in Figure 35. The Neo device requests the zone provider's IP address from the IP address provider, which can be the corporate DHCP server. The IP address provider returns a special DHCP scope option that includes the IP address of the zone directory. The zone directory is a server on the corporate network that supplies X.509 defined certificates confirming zone identities. The X.509 certificate's subject contains the IP subnet and the corporate certificate authority signs the X.509 certificate. The zone directory returns the zone identity of the requestors IP scope. The Neo device verifies the zone identity from the supplied X.509 certificate and applies the policy rule if the policy rule uses zone identities.
5. **Application identities**. When an application gets created for the Neo device, the application gets two unique identifiers. One unique identifier is assigned to the major version of the application. Another unique identifier is assigned to the

minor version of the application. The administrator can then approve applications based on either major or minor version of the application. In addition, a "catch-all" unique identifier is assigned to all applications. It is also possible for an administrator to create groups of applications and assign the group identifier. The default container that is initially created in the Neo device for the device owner allows all applications to run.

6. **GPS boundaries**. A GPS boundary, is a circular linked list of GPS coordinates, together with at least one CPS coordinate indicating which area is the inside of the GPS boundary. The circular linked list, together with the one GPS coordinate defines an area, creating a boundary.

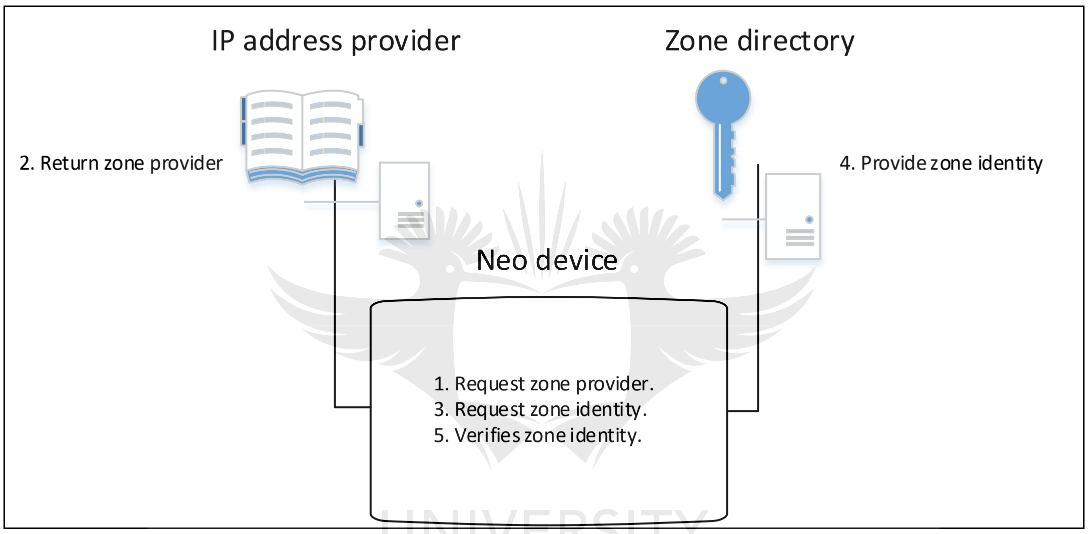
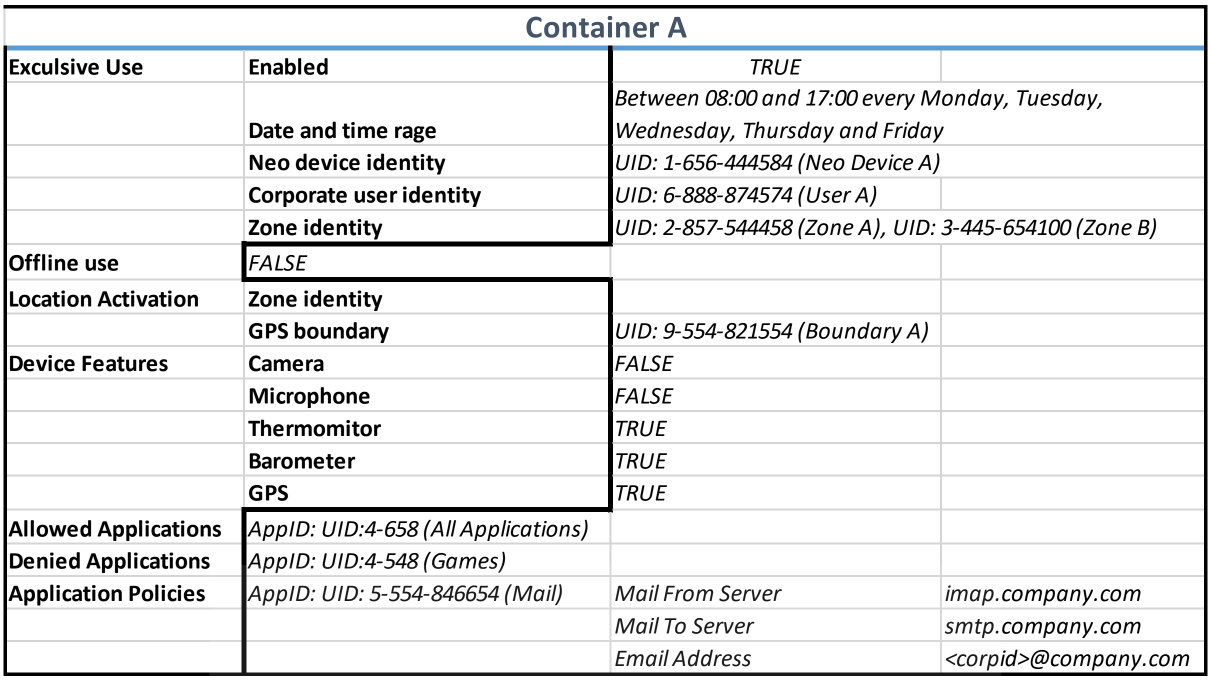


FIGURE 35: ZONE IDENTIFICATION AND VERIFICATION (BY AUTHOR)

Figure 36 is an example of a number of policy rules defined by a fictional company on their business container, called Container A. The rules that are defined in the example are the rules for exclusive use, offline use, location activation, some device features and some application policies. The example demonstrates that Container A has exclusive use of the Neo device. Furthermore, it states that the container can only be started during the week from Monday to Friday between 08:00 and 17:00. The Neo device identity states that this exclusive use rule applies to Neo Device A when User A uses it on zones A and B.



Container A			
Exculsive Use	**Enabled**	*TRUE*	
	Date and time rage	*Between 08:00 and 17:00 every Monday, Tuesday, Wednesday, Thursday and Friday*	
	Neo device identity	*UID: 1-656-444584 (Neo Device A)*	
	Corporate user identity	*UID: 6-888-874574 (User A)*	
	Zone identity	*UID: 2-857-544458 (Zone A), UID: 3-445-654100 (Zone B)*	
Offline use	*FALSE*		
Location Activation	**Zone identity**		
	GPS boundary	*UID: 9-554-821554 (Boundary A)*	
Device Features	**Camera**	*FALSE*	
	Microphone	*FALSE*	
	Thermomitor	*TRUE*	
	Barometer	*TRUE*	
	GPS	*TRUE*	
Allowed Applications	*AppID: UID:4-658 (All Applications)*		
Denied Applications	*AppID: UID:4-548 (Games)*		
Application Policies	*AppID: UID: 5-554-846654 (Mail)*	*Mail From Server*	*imap.company.com*
		Mail To Server	*smtp.company.com*
		Email Address	*<corpid>@company.com*

FIGURE 36: EXAMPLE OF POLICY RULES AND POLICY OBJECTS (BY AUTHOR)

The policy rules further state that the container may only be used while actively connected to a Gateway Controller. This means that the user cannot use it when not connected to any type of network.

The Location Activation policy states that the container automatically activates when the device detects itself within an Administrator defined GPS boundary, called Boundary A, which is the GPS boundary of the company offices.

The Device Features policy describes that the container automatically disables the abstracted camera and microphone devices, thus not allowing the user to record anything from any microphone connected to any I/O peripheral or taking pictures from any camera connected to any I/O peripheral.

The allowed applications that may be used are all the applications, except the administrator-defined group of applications, called Games.

The example concludes with an application policy that defines configuration settings for the email application that exists in the corporate container. It automatically sets the outgoing and incoming email server addresses and configures the email address given a specific syntax.

The policy mechanism enforced by the management system on the Neo device ensures that corporate data owners have control over when, where and how their data is used.

8.7 Conclusion

The SCP and the MAP fulfil the primary objective of the thesis. The primary objective of this study states that a model should be defined that isolate and protect the data and applications based on ownership.

The components that enable the SCP is summarised in Figure 37 and were discussed and described in this chapter.

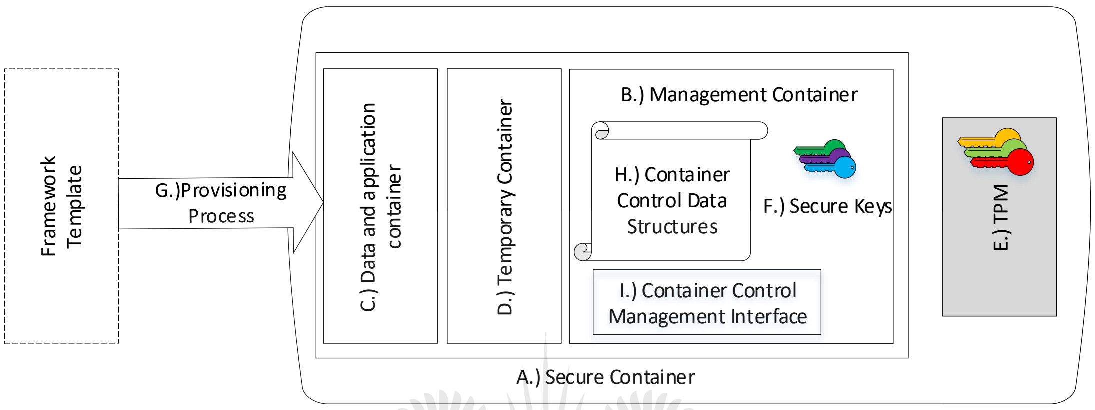


FIGURE 37: THE COMPONENTS ENABLING THE SCP (BY AUTHOR)

Section 8.2 provided a formal definition for the SCP. It defined an owner-controlled computing environment, with abstracted devices, strongly isolated from the rest of the computing environment that hosts applications and data, based on ownership. The SCP provides a definition visualised as (A) in Figure 37.

Section 8.3 highlighted that there are three different types of containers in the Neo model. User data and applications are hosted in data and application secure containers (C in Figure 37). These containers belong to a specific owner that has full control over the container. The Neo device manages access to the user data and application containers from the management system container (B in Figure 37). Temporary containers (D in Figure 37) are used to protect the management system container from network access, but also provides a mechanism for users to share information with other Neo devices, without direct access to the original containers.

Section 8.4 described how containers are encrypted and decrypted using symmetric keys. The symmetric keys are in turn protected using asymmetric keys. The management system container (F in Figure 37) and TPM (E in Figure 37) control access to the asymmetric keys and symmetric keys. The section further describes how the different keys are created and replaced when necessary.

Section 8.5 described how containers are provisioned (G in Figure 37) and section 8.6 described how businesses can control certain aspects of containers such as when they can be used and what configuration settings are automatically applied inside the container (H and I in Figure 37).

This chapter described the secure container property of the Neo model, but the reader may still not have a clear understanding of some of the control aspects of the Neo model, specifically when interfacing with I/O peripherals and other network environments. The next chapter describes the second property of the Neo model. The second property is the Mutual Authentication Property (MAP). The next chapter also describes access control and confidentiality over the network to and from the Neo device and containers.

Chapter 9 THE MUTUAL AUTHENTICATION PROPERTY - DESIGN

9.1 Introduction

The previous chapter described one of the two properties of the Neo model. The two properties, specified in Chapter 7 are the secure container property (SCP) and the mutual authentication property (MAP). These two properties are part of the primary and secondary goals defined in Chapter 1.

The SCP helps to fulfil the primary goal, which is to create a model that describes a computing environment that can be used for both personal and business purposes. The model ensures that personal and business data are strongly isolated, but still allow the data owner full control over how, where and when it can be used.

The MAP and the secondary goal evolved from the requirement of using only one device for both personal and business purposes. If the user is allowed to use only one device, how will the user interact with the device in various scenarios? When the user is in the office, the user may want to use a physical keyboard, monitor and mouse, while when the user commutes, the user may want to interact with the device using a smartphone or tablet-sized touchscreen.

The MAP ensures that all I/O peripherals and Neo devices identify and authenticate with each other. The MAP also ensures that connectivity between the I/O peripherals and the Neo device is confidential. The MAP directly fulfils the secondary objective that requires the model to be expanded to allow multiple I/O peripherals to be used with the Neo device.

The aim of this chapter is to describe the MAP. This chapter is structured in two parts. The first part describes the MAP and the communication process. The second part is structured to approach the security provided by the MAP with three of the five security services in ISO 7498-2 (ISO, 1989).

The first part is structured as follow:

- **Section 9.2** provides a formal definition of the MAP. In the definition of the MAP, the reader will see that the principles of the MAP are extended to not only include I/O peripherals, but also other network communications.
- The process of establishing and setting up the communication channel between devices are described in **Section 9.3**.

ISO 7498-2 is an accepted standard that describes five security services and a number of security mechanisms and how these security services map across the open systems interconnect (OSI) reference model (ISO, 1994). Only three of the five security services are used in this section. The three security services used are the identification and authentication, authorisation and confidentiality services. The two security services not used is the integrity and non-repudiation services.

The second part of the chapter is structured to describe the three security services:

- **Section 9.4. Identification and authentication.** This section starts by describing how the devices identify themselves and then continues describing how the authentication occurs between the Neo device and other devices, like I/O peripherals and Gateway Controllers. This section also covers how users identify and authenticate themselves. This includes both user identities managed by the Neo device, but also user identities where the identity authority lies outside the Neo device.
- **Section 9.5. Authorisation.** I/O peripherals must be authorised to have a certain level of access to not only a Neo device but also, more importantly, a specific container. Section 9.5 describes how the Neo device manages the access control. It also describes how access is controlled by a business when a Neo device connects to a company network through a Gateway Controller.
- **Section 9.6. Confidentiality**. Most communication with the Neo device occurs through a wireless connection. Section 9.6 describes how data is kept confidential while being transferred between Neo device and I/O peripheral, but also Neo devices and other networked components, like the Gateway Controller and other Neo devices.

Table 11 and Table 12 can again be used to cross reference any SCP and MAP callouts.

Before any of the MAP features are described a definition of the MAP is required. The next section provides a formal definition of the MAP.

9.2 The Mutual authentication property defined

The Neo device does not have any built-in I/O peripherals. Any interaction with the Neo device occurs through external I/O peripherals. These I/O peripherals connect wirelessly or through a docking station. Before an I/O peripheral can connect to the Neo device, the I/O peripheral is first authenticated and authorised. To ensure that the I/O peripheral is connecting to the correct Neo device, the Neo device is also authenticated on the I/O peripheral. The authentication of both the Neo device and I/O peripheral is known as mutual authentication.

The MAP property, however, describes more than just the mutual authentication. The property also addresses the authorisation of devices and confidentiality of communication. The property ensures that an I/O peripheral cannot be used with a secure container without the container owner's consent. The consent to use an I/O peripheral is part of the authorisation.

Container owners also require assurance that any interaction of I/O peripherals on their data must be kept confidential. The confidentiality aspect extends to not only I/O peripherals but also any network connectivity of the Neo device and the business network.

The **Mutual Authentication Property** (MAP) (First introduced in Chapter 7 section 7.2) is defined as: Ensuring the mutual authentication, authorisation and confidentiality between the Neo device and I/O peripherals and other relevant network devices.

The MAP is managed by the management system and identification and authorisation service on the Neo device. Figure 38, first seen as Figure 28 in Chapter 7, shows the identification and authorisation service (D) that interacts with the management system (A) to enforce the authentication of the Neo device (on the left-hand side) and the I/O peripheral (on the right-hand side). The identification and authorisation service authorises the touch screen (F) to have access to either the personal container (B) or corporate container (C). The identification and authorisation service also ensures that the communication between the I/O peripheral and the Neo device occurs through a confidential communication channel (E).

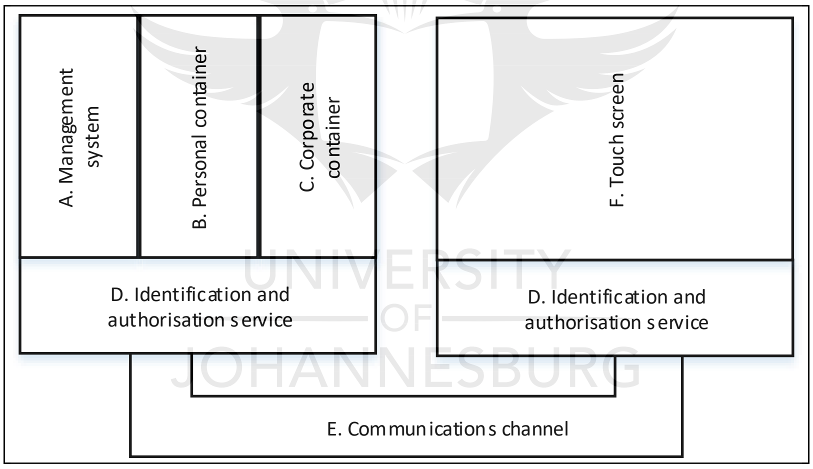

A. Management system
B. Personal container
C. Corporate container
F. Touch screen
D. Identification and authorisation service
D. Identification and authorisation service
E. Communications channel

FIGURE 38: THE NEO DEVICE COMPONENTS (DU TOIT & ELLEFSEN, 2015).

The three security services of ISO 7498/2 used to describe the security features of the MAP is applicable during the communication of various components in the Neo model. The various components include communication between the Neo device and I/O peripherals, but also communication between I/O peripherals and specific containers inside the Neo device. Before any of the security services are discussed an understanding of the communication process are required.

9.3 Communication Process

Since ISO 7498/2 is used to describe security services across the OSI reference model, the communication process of the Neo model must first be defined. An understanding of the communication process provides an understanding of the various components involved in the communication and also describes how the state of the security changes during the process.

The communication process in the Neo model describes how two devices are configured to communicate with each other. Figure 39 displays the three main phases that occur between devices from where they are first connected with each other up to where a secure channel is established between them. The three phases are:

- **Phase 1**: As described in Figure 39 there is an initial identification and authorisation phase. During this phase, the devices are identified and the devices are authorised to access each other and one of the devices are authorised to access a specific secure container.
- **Phase 2**: The second phase occurs after the initial phase. Subsequent connections between the two devices are authenticated and the authorised access of the devices are confirmed.
- **Phase 3**: The last phase is where a secure channel is established between devices, or between device and container, to ensure confidential data interchange.

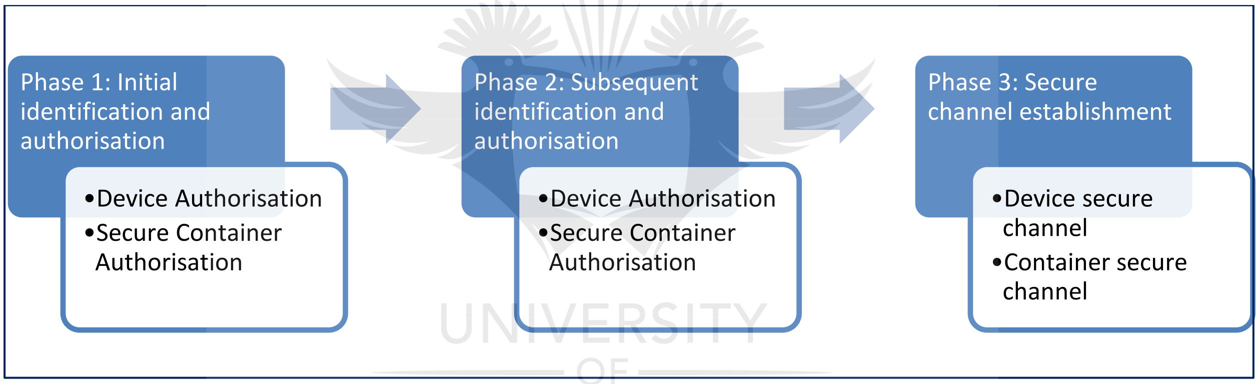

Phase 1: Initial identification and authorisation
•Device Authorisation
•Secure Container Authorisation
Phase 2: Subsequent identification and authorisation
•Device Authorisation
•Secure Container Authorisation
Phase 3: Secure channel establishment
•Device secure channel
•Container secure channel

FIGURE 39: OVERVIEW OF THE COMMUNICATION PROCESS (BY AUTHOR)

These three phases are described in more detail in the rest of this chapter, starting with the identification and authentication processes that occur in both phase 1 and phase 2.

9.4 Identification and authentication

The identification and authentication security service state that an entity must have the ability to identify itself using some type of unique identifier. Once the entity is identified, the entity is challenged to prove its identity through one or more of the acceptable categories of authentication (ISO, 1989). The generally accepted categories are:

- Something the entity knows. This can be something like a password.
- Something the entity has. This can be something like a token.
- Something the entity is. This describes the biometric aspects of the entity.

The Neo model does not prescribe which category or combination of categories must be used for authentication. The rest of section 9.4 is structured as follow:

- **Section 9.4.1** describes how Neo devices identify themselves.
- **Section 9.4.2** describes how I/O peripherals establish identity and then how authentication occurs on both the Neo device and I/O peripheral.
- **Section 9.4.3** moves away from the device and peripheral identification and describes how users are identified and authenticated, taking into account users can be either managed by the Neo device or outside the Neo device.

9.4.1 Device identification

The MAP requires that both parties authenticate themselves in the start of the communication process. This means that the Neo device should have the ability to authenticate with an I/O peripheral, just as an I/O peripheral should authenticate with the Neo device. Authentication of the Neo device implies that the Neo device has an identity.

The use of cryptographic keys as described in Chapter 8 section 8.4 for encrypting and decrypting of secure containers. Each Neo device generates a public-private key pair. Each Neo device also generates a unique identifier that identifies the device. The unique identifier with the public-private key-pair is used in the identification and authentication process between the Neo device and other devices.

I/O peripherals in the Neo model has some level of computing capability. Each I/O peripheral also has an identification and authorisation service that manages the identification, authorisation and confidentiality of devices and communication. I/O peripherals also have a public-private key-pair and each has a unique identifier.

This same concept also occurs on Gateway Controllers, with each Gateway Controller having a unique identifier and its own set of public-private keys. Gateway Controllers were introduced in Chapter 7 and are used when a Neo device communicate with corporate networks and corporate resources.

When two devices in the Neo model connects for the first time, the two devices exchange device identification information. The two devices can be any combination of Neo device, I/O peripheral and Gateway Controller. The devices go through an authentication process to verify their identities.

9.4.2 Device authentication

The three categories of devices that can authenticate each other are:

- **Neo devices**. The Neo devices are the computing equipment belonging to the user. These devices are the primary computing devices for users in the Neo model.
- **I/O peripherals**. The I/O peripherals are computing peripherals that produce output generated by the Neo devices or act as input devices to Neo devices, allowing users or sensors to interact with the Neo device.

- **Gateway Controllers.** Gateway Controllers act as guardians between Neo devices and business networks.

Authentication of devices occurs either initially when it is the first time devices want to communicate with each other, or subsequently after initial authentication. There is a slight difference in initial authentication and subsequent authentication.

9.4.2.1 Initial authentication

The goal of the initial authentication (Phase 1 in Figure 39) is to exchange identities and create a shared secret that can be used for future communication. This means that after the initial authentication occurred, the Neo device has a copy of the identity of the other device and the other device has a copy of the identity of the Neo device. Both devices also have a shared secret that is unique to them and no other two devices.

When two devices are initially configured to communicate with each other, the devices must use the out-of-band (OoB) connector on the Neo device or use some technology like near field communication (NFC). The reason for OoB or NFC is to minimise the man-in-the-middle risks associated with the Bluetooth protocol (Haataja et al., 2013).

MAP1.2

In the case of a Gateway Controller, the company authorises the Neo device by either authorising the Neo device from the company docking station or through a special authorisation node, which may have an NFC reader.

The act of the user either connecting the Neo device to a peripheral or physically touching the Neo device to another device assumes that the owner of the Neo device approves the initial authentication. In the case of a peripheral, the act of physically connecting the peripheral or touching via NFC approves the action from the peripheral owner's perspective.

In the case of a business, the user is physically allowed to dock the Neo device in a docking station, assuming approval from the business. When no docking station is available, the company provides an NFC point, where the user has to touch the Neo device. After the user docked the Neo device or touches the NFC point the identity of the Neo device is sent to an administration system on the company network. The administration system interfaces with the Gateway Controller to establish the shared secret.

The physical interaction between the two devices creates a shared secret that is used between the Neo device and the other device. The identity of the two devices is also exchanged to be used for further authorisation. The implementation of creating the shared secret is beyond the scope of this document since there are many established protocols, such as Bluetooth and WPA2, that does this well (Padgette, 2017).

9.4.2.2 Subsequent authentication

MAP1.1

After establishing the initial authentication, subsequent authentication occurs without any human intervention. The previously saved shared secret and identity information of the two devices are used to authenticate the devices with each other. The goal of subsequent authentication (Phase 2 in Figure 39) is to confirm the identities of the two devices and authenticate each other. This provides the two devices with a shared secret that can be used to establish a secure connection. The secure connection is described in Section 9.6.

9.4.3 User identification and authentication

As with any modern operating system, the Neo device requires any user to log onto the device. The initial login uses a user identity that is stored and managed by the Neo device. All user identities and user secrets are stored in the management system container by the identification and authorisation service.

Depending on the I/O peripheral used by the user the authentication supports more than just passwords, but can also support biometrics or token-based authentication. When the device is initially switched on, the device requires the user to connect an I/O peripheral to the device. As soon as the I/O peripheral is connected the Neo device prompts the user to create a user identity with a secret that can be used during the login process. The identification and authentication service authenticates any user accounts stored on the Neo device.

In the case of the Neo device connecting to a business network, the Gateway Controller prompts the user for an authorised business identity. The authentication of the business identity is managed via the Gateway Controller, using techniques similar to RADIUS authentication (Kizza, 2015). A copy of the business identity is stored in the management system and linked to the current logged in Neo user identity. Depending on the policy from the business the business user's challenge is also cached as part of the business identity stored in the management system. This allows a user of the Neo device to automatically use the cached credentials of the business user to gain access to corporate containers, without having to specify a second set of credentials.

The caching of business credentials can be controlled by policy elements on the Neo device. The device and business container policy must be in line with company policy, since it may be possible to indirectly authenticate a user using an authentication mechanism that may be different from what the company policy dictates.

Device authentication occurs in two phases when establishing a connection with another device. User authentication occurs before a user is allowed to use a Neo device, or before a business container becomes active. The next section describes how I/O peripherals and users are authorised to access different containers in the Neo model.

9.5 Authorisation

Authorisation is the process of controlling a level of access to resources. The Neo model supports a discretionary access control model, with the owner of the secure container controlling the access levels to the data and applications. This means that a user identity managed by the Neo device has full control over its own personal container, and a business owner has full control over the access levels of a relevant corporate container.

Section 9.4 described the identification and authentication of both devices (which can include I/O peripherals or Gateway Controllers) and user accounts in the Neo model. Once a device or user has been successfully authenticated, the authorisation mechanism controls the access level of I/O peripherals and user identities.

The next section describes how the initial authorisation is configured for I/O peripherals and how subsequent requests are checked by the identification and authorisation service.

9.5.1 I/O peripheral authorisation

During the initial authentication process, described in 9.4.2.1, the I/O peripheral does not have access to any of the secure containers currently present on the Neo device. By default, the act of authenticating an I/O peripheral to a Neo device authorises the I/O peripheral access to the Neo device. The container owner must define access rules to the owner's secure containers.

MAP2.3

The Neo model uses the principle of access rules to control access to and from certain devices on the Neo model ecosystem. It is important to note that an access rule does not only control access to specific objects but also to ensure that objects connect only to certain subjects. This means that access rules are just like firewall rules that control connections to and from a specific entity.

Figure 40 describes two access control rules, called **Access Rule 1** and **Access Rule 2**, are configured on the Neo device. The Neo model identifies a number of objects and subjects that can exist in the Neo model ecosystem. Objects include items like a Neo device (depicted as ***Neo Device*** in Figure 40) or specific secure containers (depicted as ***Container A*** and ***Container B*** in Figure 40). Objects are generally referred to as the components being accessed by subjects.

Some of the subjects in the Neo model ecosystem can be a specific user account (depicted as ***User Account*** in Figure 40) and a specific I/O peripheral (depicted as ***I/O Peripheral*** in Figure 40). A combination of subjects and objects are arranged in access control rules that specify which subjects have access to which objects.

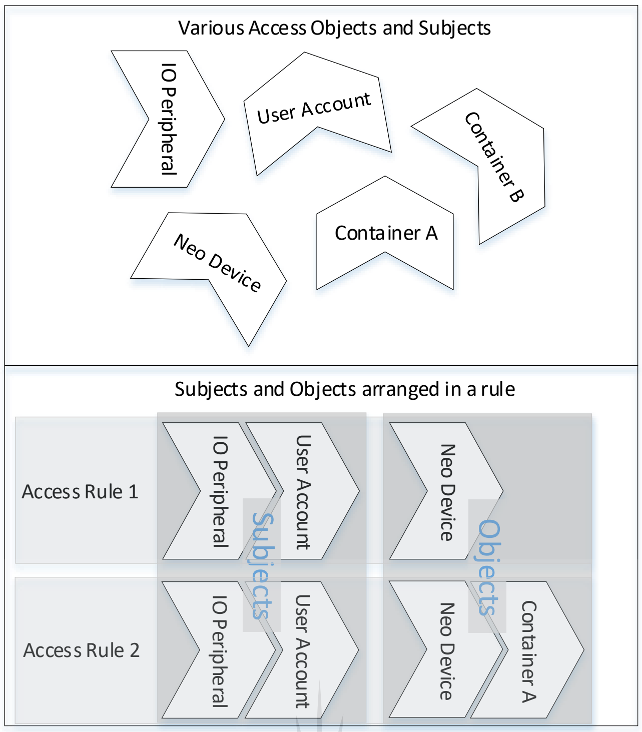


FIGURE 40: NEO DEVICE ACCESS CONTROL RULES (DU TOIT & ELLEFSEN, 2015)

Figure 40 shows ***Access Rule 1***, which is created during the initial authorisation process of ***I/O Peripheral***. ***Access Rule 1*** shows that ***User Account*** and ***I/O Peripheral*** has access to the ***Neo Device***. As soon as the owner of ***Container A*** authorises the ***I/O Peripheral*** access to the container, ***Access Rule 2*** is created. ***Access Rule 2*** describes ***User Account*** using ***I/O peripheral***, has access to the ***Container A*** running on the ***Neo Device.***

Access rules that for I/O peripherals can be described as an access rule triplet. The triplet contains the user account, I/O peripheral and a group of objects. ***Access Rule 1*** and ***Access Rule 2*** are examples of access rules for I/O peripherals.

MAP2.1

The process that enforces the access control rules is called the identification and authorisation service (D in Figure 39). The identification and authorisation service utilise the access control rules that are stored in the management system container.

Just as I/O peripheral identifiers and user identities are verified, before access is granted to a secure container, Gateway Controllers also verify Neo device identifiers and user identities, when a Neo device connects to a business network.

9.5.2 Network authorisation

Network authorisation expands on the principles established in the previous section. Network components are defined and described as either subjects or objects. The network components include items such as IP subnets, server objects and even specific UDP or TCP ports.

Figure 41 is an example of some of the network objects that can exist. On the left-hand side are six different network objects. Starting at the top is an IP subnet that corresponds to the corporate network. Below that is a database server, an email server and a line-of-business server. One of the objects is also a generic Internet object, which groups all networks that exclude the corporate network. Network objects can also include TCP port specific objects.

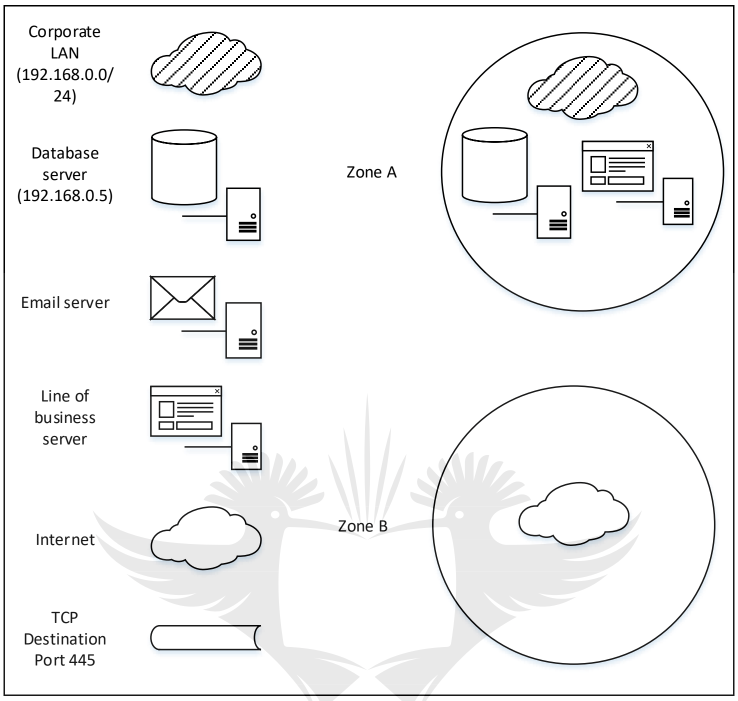


FIGURE 41: NETWORK OBJECTS AND ZONES (DU TOIT & ELLEFSEN, 2015)

The objects are grouped together by the business' network administrator into zone objects. Zone A in Figure 41 describes a group that allows access to the corporate network and specifically the database and email servers. Zone B describes access to the Internet. The network administrator can then set up an access rule on the Gateway Controller of the company that forces communication from a specific secure container running on a specific Neo device to connect to objects in a specific zone.

Figure 42 expands on the access rules described in section 9.5.1. Apart from the normal objects and subjects, like **I/O peripheral**, **Neo device**, **User Account** and **Container A**, two other objects are also defined. These objects are **Zone** and **Gateway Controller**. Figure 42 consists of three sections. At the top, it defines a number of subjects and objects. In the middle are the access rules that can be found on the Neo device. At the bottom is an access rule defined on a Gateway Controller.

Access Rule 1 of the Neo device is a copy of the access rule defined in the previous section that ensures that a **User Account**, using an **I/O peripheral** has access to **Container A** on **Neo Device**.

Access Rule 2 of the Neo device defines that any network traffic from **Zone** routed through the **Gateway Controller** has access to the **Container A** on the **Neo device**.

Access Rule 1 on the **Gateway Controller** specifies that **User Account** using **I/O Peripheral** on **Neo Device** using **Container A** has access to **Gateway Controller** and then has access to objects on **Zone**.

The access rules described in Figure 42 ensures that a user has the ability to access corporate network through a Gateway Controller. Furthermore, it ensures that only traffic between the corporate container and the specific network resources are allowed.

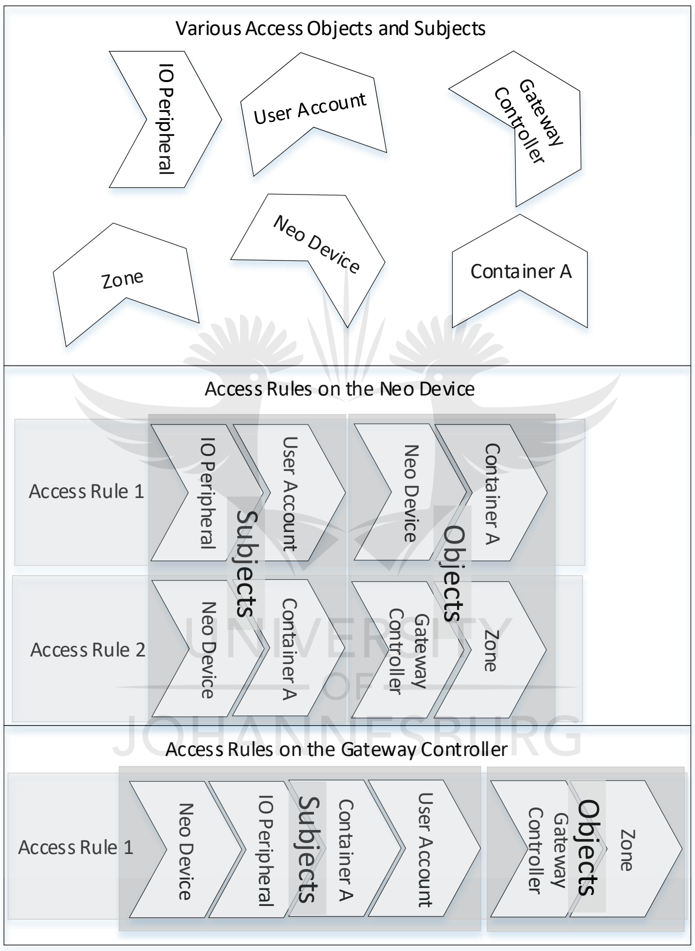


FIGURE 42: ACCESS RULES FOR NETWORK SUBJECTS AND OBJECTS (DU TOIT & ELLEFSEN, 2015)

A business network consists of many objects. Some of the objects may include things like authentication servers, mail servers, application servers and even certain IP subnets. In a Neo model environment, the Neo device can either connect to the business network using a docking station or a WiFi connection. The challenge faced by the Neo device is to detect when a connection should be seen as a general Internet type of connection or a connection established for business purposes.

In the case of WiFi connections, the Neo device categorises a connection as either public or private. In the case of a wired connection through a docking station, the Neo device categorises the connection as private. Figure 43 shows a public connection on the left-hand side and a private connection on the right-hand side. Public connections do not require an encrypted channel to a Gateway Controller and connect directly to Internet resources. The Neo device uses public connections to connect to Internet resources. Private connections require an end-to-end connection from the Neo device to a Gateway Controller. The end-to-end connection is described in more detail in section 9.6.2.

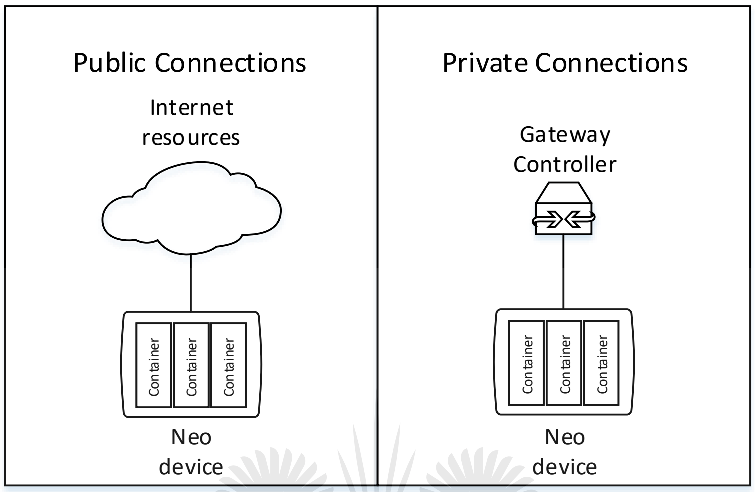


FIGURE 43: PUBLIC AND PRIVATE CONNECTIONS (BY AUTHOR)

The access control rules that manage container objects always contain either an Internet object or a Gateway Controller object. This means that an access control rule, to and from a secure container, can connect only to Internet resources or to a specific Gateway Controller. **Access Rule 2** on the Neo device in Figure 44 describes an access control rule in the case where a business container can only connect to a corporate network. **Access Rule 1** in Figure 44 describes a private **Container B** that cannot access the business network but can exchange information with resources on the Internet.

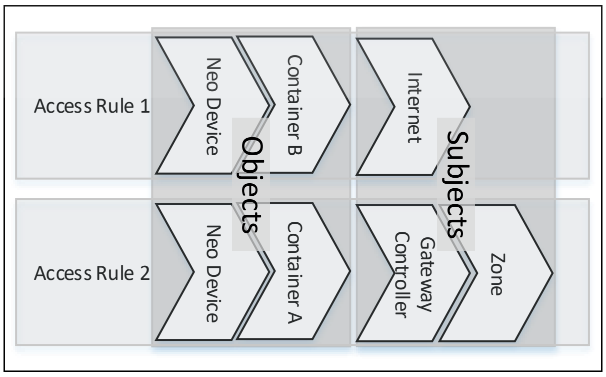


FIGURE 44: THE INTERNET AND BUSINESS ACCESS (BY AUTHOR)

In the case of a business, the access control rules are defined by the business and created on the Neo device during the time when the corporate container is provisioned on the device. Similar to the access control rules created on the Neo device, access control rules are also created on the Gateway Controller. The objects defined on the Gateway

Controller, however, includes network-specific objects. The network specific objects are grouped into zones.

All of the above can be summarised as follow. The Neo device can have an access rule that enforces communication to and from a specific secure container to a Gateway Controller. The Gateway Controller has an access rule that enforces communication from a secure container running on a Neo device. The access rules are similar to firewall rules and ensure specific network connectivity.

Even though all the diagrams used in this document describes the identification and authorisation service as a separate service on the Neo device, the service is actually running as a service inside the management container.

MAP2.2

The identification and authorisation service on the Neo device ensures traffic to and from a specific container can only connect to objects in a predefined rule set. The rules are known as access rules and stored in the management system. The identification and authorisation service also ensures that the traffic between nodes is confidential.

9.6 Communication confidentiality

Confidentiality in the Neo model is described as scrambling the data in such a way so that unauthorised systems cannot understand the data. This section describes how the Neo model ensures communication channels between Neo devices and I/O peripherals or Neo devices and Gateway Controllers are confidential.

Network communication between the Neo device and other components are encrypted. The focus of this section is not on which encryption protocol is to be used, but instead focusses on the key management.

- **Section 9.6.1** describes the management of keys used for communication between Neo device and I/O peripherals.
- **Section 9.6.2** focusses on the communication encryption and decryption between Neo device and the corporate network.

9.6.1 I/O peripheral confidentiality

Section 9.4.2.1 described how a Neo device and I/O peripheral authenticate each other. The result of the authentication is a shared secret that exists between the I/O peripheral and Neo device. The shared secret is stored in the Management Container and is only accessible by the identification and authorisation services.

MAP3.2

After the authentication process, the Neo device and I/O peripheral uses the shared secret, with their unique identifiers and session clocks, to negotiate session keys that are only used while the I/O peripheral and Neo device is connected. The session key is used as a symmetric key with a symmetric key algorithm to ensure data confidentiality between the Neo device and the I/O peripheral.

MAP3.1

The use of session key, unique identifiers and shared secret is a documented approach used by Bluetooth to negotiate session keys (Padgette, 2017).

Encryption is established between Neo device and I/O peripheral and not between secure container and I/O peripheral. This means that I/O data flow to and from the secure container to the cypher on the Neo device in an unencrypted format. At the cypher, the I/O data is encrypted and sent to the I/O peripheral where the data is decrypted using the same session key where it is then consumed by the I/O peripheral.

The reverse also happens. Data is generated by the I/O peripheral, where it is encrypted using the session key. At the Neo device, the data is decrypted and forwarded to the abstracted I/O peripheral in the secure container.

The reason why the encryption is not managed from within the secure container is that the secure container owner can potentially contaminate keys stored inside a secure container. Keys and the management of keys are stored and managed outside the influence sphere of the users of the system. An owner of a secure container has full permissions of the data stored in a secure container. Keys stored in a secure container can be exposed to an owner, which in turn can expose it to other users.

The unencrypted channel that exists between the secure container and Neo device, may allow another container to intercept I/O peripheral communication between the secure container and Neo device. This may lead to a scenario where a secure container can become infected with malware that misbehaves and then tries to intercept communication on the abstracted I/O peripherals managed by the identification and authorisation service.

One solution to solve the above problem is to ensure exclusive access. The identification and authorisation service monitors the containers requiring interaction. As soon as a container requires interaction, a session between the Neo device and approved I/O peripheral is established or resumed. As soon as the container loses interaction focus, the identification and authorisation service stops the communication with the I/O peripheral. This means that a container has exclusive access to an I/O peripheral ensuring no other container can intercept unapproved I/O peripheral data.

The exclusive access principle poses a challenge with applications in secure containers that require access from peripherals non-interactively. Example: Tracking software in a secure container needs to track the position of an I/O peripheral mounted inside a vehicle. As soon as the secure container hosting the tracking software loses focus, the tracking software stops receiving signal information from the I/O peripheral.

Another solution is to categorise I/O peripherals into interactive or non-interactive peripherals. Interactive I/O peripherals, create exclusive access channels between the Neo device and the interactive secure container. Non-interactive I/O peripherals allow multiple secure containers to access the data generated by these peripherals. Interactive peripheral examples include keyboards, mousses, displays, microphones and cameras. Non-interactive peripherals can be GPS signals, temperature and barometric information and motion sensors.

Please note that network interfaces and storage is not managed like I/O peripherals, as these devices are fully abstracted inside the secure containers, with their own identifiers ensuring communication between the abstracted representation and the physical representation in the Neo device is properly routed.

I/O peripheral communication sounds simple but poses a number of challenges when trying to implement. Some of these challenges include screen latency and levels of abstraction. These challenges are discussed in more detail in Chapter 11.

Confidentiality is not only a requirement between I/O peripheral and Neo device but also the Neo device and other network endpoints. The network end-points can be inside a corporate environment, other Neo devices or Internet-based end-points. Corporate based communication confidentiality are discussed in the next section.

9.6.2 Corporate communication confidentiality

Business owners are assured of network confidentiality when a Neo device connects to a corporate environment. Sections 9.5.1 and 9.5.2 already described the concept of a Gateway Controller and private connections. In the case of private connections, the Neo device enforces an end-to-end channel between the Neo device and the Gateway Controller. The enforcement of this end-to-end channel is further enhanced by the fact that the abstracted network interface controller that resides inside the secure container, automatically sets up a virtual private network (VPN) between the secure container and the Gateway Controller.

It is important to note that the VPN connection occurs from within the secure container. The VPN configuration is automatically configured during the provisioning process of the secure container. The provisioning service, with the virtual private network service, as described in Chapter 8 section 8.5.

In cases where the company has more than one Gateway Controller, the company administrator may allow the Neo device to connect to more than one Gateway Controller. In these cases, multiple VPN configurations are defined in the secure container as well as multiple Gateway Controller access control objects on the Neo device. Multiple Gateway Controllers may be necessary to allow users to work from within the company network and from outside the company network.

Figure 45 describes a situation where a company has two Gateway Controllers. On the left-hand side is the corporate network with a network zone object. The internal Gateway Controller positioned just below the network zone, controls access for Neo devices that connect either using the corporate WiFi network or the wired network. The external Gateway Controller that is positioned between the corporate network and the Internet controls access to the Corporate network by Neo devices entering the network from the Internet. In both these cases, traffic between the corporate secure container and the Gateway Controller uses a virtual private network (VPN) connection.

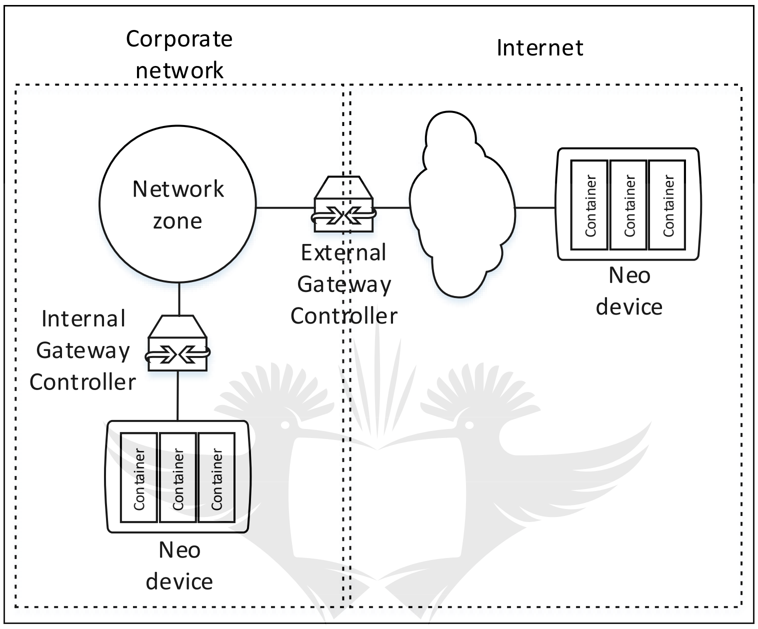


FIGURE 45: MULTIPLE GATEWAY CONTROLLERS (BY AUTHOR)

In cases where more than one VPN configuration is defined, the secure container polls each of the defined Gateway Controllers to see which one is available before initialising the applicable VPN configuration.

The VPN configuration is saved and managed from within the corporate secure container. This means that companies will be able to add or modify VPN settings when the container is connected to the corporate network. VPN settings and access control are all managed by the policies described in Chapter 8 section 8.6.

It is also possible for a company to allow traffic to and from personal containers stored on Neo devices to access Internet resources. The connections are categorised by the Gateway Controller as unauthorised connections. Unauthorised connections have a forced connection to a Gateway Controller, from where they only have access to resources on the Internet. The question that arises is how does the Neo device detect that it is inside a corporate network? This problem is addressed in network technologies like network access control (NAC) and 802.1x that can detect network connectivity from unauthorised containers and route them automatically to the Internet and falls outside the scope of this document (Institute of Electrical and Electronics Engineers, Inc, 2012).

The confidentiality component is established and managed in the MAP of the Neo model. Confidentiality is assured between Neo device and I/O peripherals, but also between containers inside the Neo device and business networks. The next section provides a summary of some of the components or activities that occur during the identification, authentication, authorisation and channel confidentiality of the mutual authentication property (MAP).

9.7 Summary of components and services

Sections 9.4 to 9.6 described a number of components, services and interactions that enable the MAP of the Neo model. Table 6 provides a summary of the various components and services that enables the mutual authentication property (MAP). The columns are organised according to three of the five security services described by ISO 7498/2, which in this case is: Identification and authentication, authorisation and channel confidentiality. The rows depict the interaction between the different components in the Neo model. This includes the various permutations of communication between users, Neo devices, I/O peripherals, containers and Gateway Controllers.

TABLE 6: SUMMARY OF COMPONENTS AND SERVICES THAT ENABLES MAP

	Initial Identification and Authentication	Initial Authorisation	Subsequent Identification and Authentication	Subsequent Authorisation	Channel Confidentiality
User to Neo Device	User account either created during initial start-up or created by Neo device owner.	Access Rules (Identification and Authorisation service)	User account and authentication mechanism	Access Rules (Identification and Authorisation service)	N/A
User to Container	User account either created during initial start-up or created by Neo device owner or linked to business identity	Access Rules (Identification and Authorisation service)	User account and authentication mechanism. Cached credentials	Access Rules (Identification and Authorisation service)	N/A
Neo Device to Neo Device	Out-of-band (NFC)	Established through initial identification and authentication	Shared secret	Access Rules (Identification and Authorisation service)	Session-based encryption
I/O Peripheral to Neo Device	Out-of-band (NFC)	Established through initial identification and authentication	Shared secret	Access Rules (Identification and Authorisation service)	Session-based encryption
Neo Device to Gateway Controller	Out-of-band (NFC or docking station)	Established through initial identification and authentication	Shared secret	Gateway Controller Zones	Session-based encryption
Container to Gateway Controller	Created by Administrator	Established through initial identification and authentication	Shared secret	Access Rules (Identification and Authorisation service)	VPN based encryption

Some aspects are not applicable. This is especially true when describing the establishment and upkeep of a secure channel between the user and the Neo device and between the user and the secure container. Users interact with the Neo device or secure containers through an I/O peripheral. The same is not true for a user authenticating or getting access to a device or container. In these instances, the user is a specific security subject that must first be authenticated and then authorised.

Table 6 further describes that the initial identification and authentication process differ from subsequent identification and authentication sessions. The Neo device owner initially creates users or link a business account to the account on the Neo device. Identities for devices, whether they are I/O peripherals, other Neo devices or Gateway Controllers are established using some out-of-band (OoB) mechanism. Subsequent identification and authentication occur using the established accounts during the initial identification and authentication.

Authorisation happens implicitly during initial identification and generates access rules that are used during subsequent authorisation processes.

Initial identification and authentication enable the mechanism that allows Neo devices, secure containers and Gateway Controllers to establish session based encryption. Session-based encryption allows devices or containers to communicate securely.

Even though there are many more subtle processes and activities that happen during identification, authentication, authorisation and channel confidentiality, this section tried to summarise the most salient aspects.

9.8 Conclusion

The MAP is one of the two properties that define the core of the Neo model. The MAP assures three of the five security services defined in ISO 7498/2. The three services addressed by the MAP includes the identification and authentication of I/O peripherals, users and Neo devices. It addresses the authorisation of I/O peripherals, other Neo devices and users to secure containers. Lastly, it describes data confidentiality during transit between Neo devices and I/O peripherals and Neo devices and corporate environments.

The MAP is an important part of the Neo model and addresses the secondary goal of the thesis that allows a user to use multiple I/O peripherals with the Neo device for both business and personal use.

Section 9.2 provided a formal definition of the MAP. The formal definition states that the MAP provides mutual authentication, authorisation and confidentiality of the Neo device and I/O peripherals and other relevant network devices. Section 9.2 also described the various components in the Neo model affected by the MAP.

Section 9.3 provided an overview of the communication process of the Neo device and other components. The communication process defined three phases of communication that occurs between Neo device and other network components. Phase 1 defined an initial identification and authorisation. Phase 2 describes recurring identification and authorisation that ensures subsequent security checks after initial communication. Phase 3 defined the phase where the Neo device and network device communicates over an encrypted channel.

Section 9.4 provided a discussion on the identification and authentication that occurs between Neo device, I/O peripherals, users, Gateway Controllers and other Neo devices. The section described identification and authentication as two separate processes that occur and described how users are identified and authenticated, even when using corporate user identities.

Section 9.5 discussed how the access of users, I/O peripherals and Gateway Controllers are managed and applied to secure containers. It describes the use of access rules defined by the container owner and used by the identification and authorisation service to ensure access control.

Section 9.6 discussed how communication between Neo device, I/O peripheral and other components, like Gateway Controllers are managed to ensure data confidentiality during transit. The use and management of encryption keys were discussed, showing that in some instances data is encrypted between devices, and in other instances that data is encrypted from within the secure container to the corporate gateway through a VPN.

Chapter 8 and Chapter 9 described the fundamental properties of the Neo model, namely the SCP and the MAP. Chapter 10 describe how these concepts and components are used in the life cycle of a container.

Introduction

- Chapter 1 Introduction

The end game

- Chapter 2 Model Overview

Literature study

- Chapter 3 Isolation properties in mobile operating systems (SCP)
- Chapter 4 Isolation techniques in other technologies (SCP)
- Chapter 5 An overview of peripheral security (MAP)

Bridge

- Chapter 6 The drivers for a new model

Building the Neo model

- Chapter 7 Overview of the Neo Model
- Chapter 8 The Secure Container Property
- Chapter 9 The Mutual Authentication Property
- **Chapter 10 Container life cycle**
- Chapter 11 Novel Implementations

Evaluation and summary

- Chapter 12 Potential implementation of Neo Model
- Chapter 13 Conclusion and future work

Chapter 10 CONTAINER LIFE CYCLE

10.1 Introduction

Chapters 7, 8 and 9 describe the Neo model and the two properties that underpin the Neo model, namely the secure container property and the mutual authentication property. The SCP and MAP address the primary and secondary objectives of this thesis.

The primary objective requires the creation of a model that describes a mobile operating system that can be used for both business and personal purposes. The secondary objective states that the model requires the ability to interface the mobile computing device with different I/O peripherals.

The aim of this chapter is to provide clarity on how the various components of the Neo model work together in a secure container's life cycle. The aim of the chapter is to enforce a deeper understanding of how the various components interact given certain scenarios. Secure containers go through different processes in their life cycle. There are different activities that cause a secure container to switch from one process to another.

This chapter is structured as follow:

- **Section 10.2** describes the three processes of a secure container and summarises the different activities that change the state of a container from one state to the next.
- **Section 10.3** describes the interaction of the different components in the Neo model when a container is provisioned.
- **Section 10.4** describes the activities that cause a container to move from a provisioned process into a run process.
- **Section 10.5** describes the activities that cause a container to be de-provisioned.

10.2 Lifecycle states and activities

The Neo model identifies three processes that influence the life cycle of a data and application secure container. This chapter focusses on the processes that directly influence the life cycle of a secure container, even though there are many other processes in the Neo model.

This section is structured in such a way as to describe the three processes (depicted in Figure 46) that affect the life cycle of a secure container:

- **Section 10.3. The provisioning process**. The provisioning process is responsible to ensure a secure container is created and a user and I/O peripheral has permissions to interact with the new secure container. The activities that start the provisioning process are:

- **Section 10.3.1**. The initial device start-up. The initial device start-up ensures that there is at least one secure container created. The initial start-up also ensures that a user is given permission to access the secure container.
- **Section 10.3.2**. An interactive user creation. When a device owner creates a new user on the Neo device, the provisioning process ensures that either the new user is assigned to an existing container, or a new secure container is created for the user.
- **Section 10.3.3**. Business provisioning. When a device is enrolled in a corporate environment a secure container for the corporate applications are created and a user is given permission to access the newly created secure container.

- **Section 10.4. The run process**. The run process is responsible to activate one or more secure containers. Secure containers can be started with the following activities:
  - **Section 10.4.1.** Interactive user login. When a user logs into the Neo device, the secure containers where the user has access to, is started. Interacting with started secure containers are only allowed using approved I/O peripherals.
  - **Section 10.4.2.** Remote initiated. Containers can also be started remotely when a Neo device gets specific startup instructions from a business management system.
- **Section 10.5. The de-provisioning process**. Secure containers are destroyed after two activities:
  - **Section 10.5.1**. Interactive removal. A device owner can interactively select a secure container and remove it from a Neo device.
  - **Section 10.5.2.** Business de-provisioning. Businesses can decide to remove secure containers from Neo devices. This process can be remotely initiated.

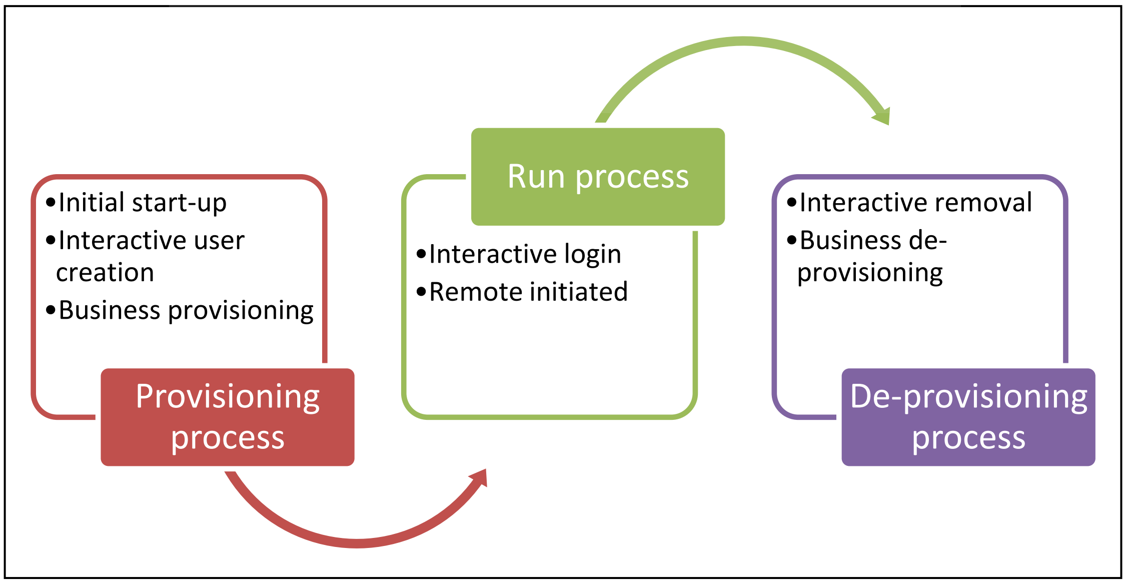


FIGURE 46: THE CONTAINER LIFECYCLE PROCESS (BY AUTHOR)

The provisioning process is the process that is responsible for creating the secure containers and discussed in the next section.

10.3 Provisioning process

The provisioning process ensures that a new secure container is initiated. New secure containers are created during initial start-up, when a user creates a new user and when a business provisions a secure container.

Each of these activities uses different interactions and different components. The first action described is the creation of a secure container during the initial start-up of the Neo device.

10.3.1 Initial start-up

When the Neo device is started for the first time, there are no users created on the Neo device. There are also no secure containers on the device, apart from the management container. The management container is an internal container and used by the identification and authorisation service and was first introduced in Chapter 7.

The initial start-up action has six general steps (Summarised in Figure 47) that ensure the creation of a secure container. Steps 1 and 2 are unique to the initial start-up process. Steps 3 to 6 differ in their complexity depending on whether the device is used for personal or business purposes.

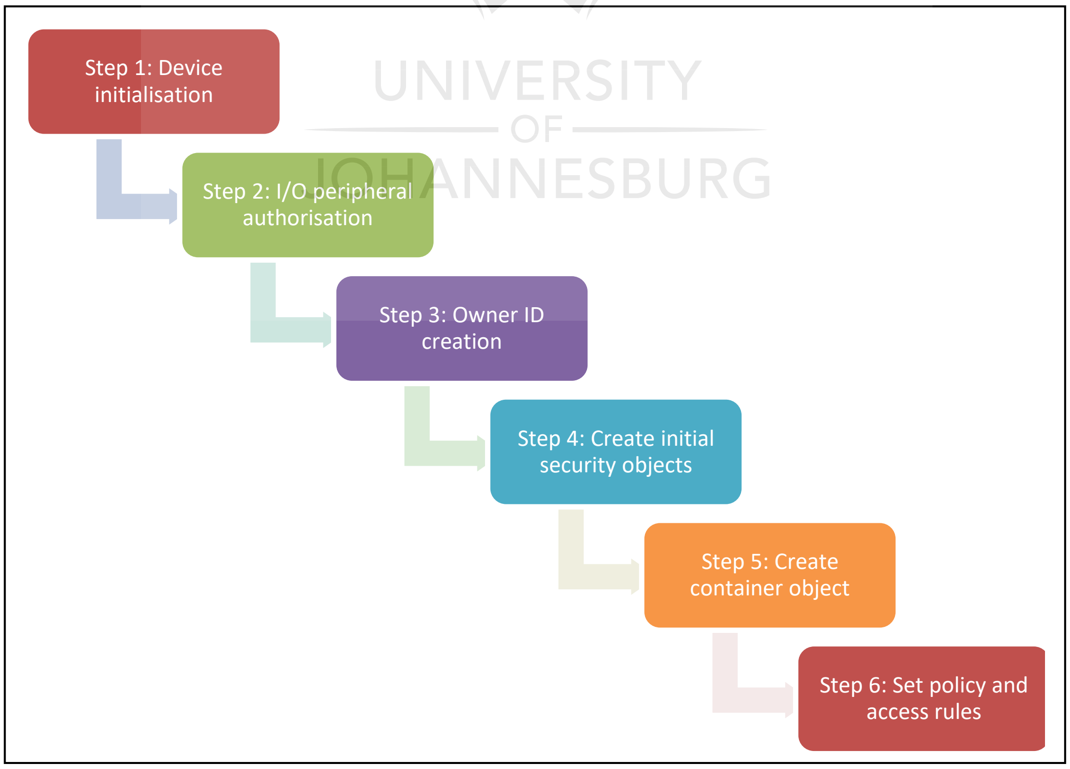

Step 1: Device initialisation
Step 2: I/O peripheral authorisation
Step 3: Owner ID creation
Step 4: Create initial security objects
Step 5: Create container object
Step 6: Set policy and access rules

FIGURE 47: GENERAL START-UP STEPS (BY AUTHOR)

10.3.1.1 Step 1: Device initialisation

The outcome of step 1 is to ensure that the Neo device has a unique identifier, the management container is encrypted using a unique symmetric key, and the integrity of the services inside the management container is established.

When the Neo device is started the Neo device interrogates the management container and see if there are any existing containers on the device. If there are no secure containers on the device, the Neo device assumes it is the first time the device is switched on.

Even if this is the first time the device is started, the management container is encrypted using a symmetric key that in turn is encrypted using a public key that is factory generated. The symmetric key is unlocked using an initial factory created a private key. This, in turn, allows the management container to be mounted.

After this initial mounting, the Neo device, through the use of TPM, generates a unique identifier and a new asymmetric key-pair. The Neo device generates a new service list file and signs it with the private key of the Neo device. A new symmetric key is also created and the management container is encrypted with the newly created symmetric key. The symmetric key is now protected using the new asymmetric key-pair.

SCP4.3

Any subsequent start-up of the Neo device decrypts the new symmetric key that is used to decrypt the management container. After the management container is mounted, the integrity of the services inside the management container is verified by using the service list file that was signed during the initial start-up.

SCP4.2

The result of step 1 has now been achieved because the Neo device has a unique identifier and the management container is protected using a set of unique and newly created keys. The integrity of the services inside the management container has also been verified.

10.3.1.2 Step 2: I/O peripheral authorisation

After the management container is secured, the outcome of step 2 is to ensure that the owner of the Neo device can interact with it. This is achieved by authorising an initial I/O peripheral\s. It should be realised that the I/O peripheral authorisation can authorise a combination I/O peripheral like a touch screen, or it can authorise separate I/O peripherals like an external keyboard, screen or mouse connected through a docking station.

The authorisation requires the I/O peripherals to connect to the Neo device either through an out-of-band (OoB) connection, or near field communication (NFC), but not through the default wireless I/O interface.

MAP1.2

Normal I/O peripheral communication occurs through wireless communications like Bluetooth or WiFi.

Double authorisation can occur between the Neo device and I/O peripheral, by something such as a button on the Neo device and another on the I/O peripheral. This, however, is not a requirement and can be implemented using a different mechanism. The act of connecting the devices together with OoB connector already implies a positive intent by the device owner. During the authorisation, the I/O peripheral and Neo device establish a shared secret that is used for future communication. The shared secret is used to encrypt communication between I/O peripheral and Neo device.

MAP1.1

MAP3.1

As soon as it is possible for an owner to interact with the Neo device, steps 3-6 diverges slightly depending on whether the device belongs to a consumer or business. Section 10.3.2 describes the steps taken when a typical consumer owns the device. Section 10.3.3 describes the steps taken when a business owns the device.

10.3.2 Interactive user creation

New user accounts can be created interactively either by the device owner or during the initial provisioning process. During the initial start-up, the Neo device prompts the user with two options. The two options are:

- **Create a new user**. When the device is owned by a consumer user, the user may choose this option to create a new user, so that the device owner can start using the device immediately.
- **Provision for a business**. When the device is owned by a company, the company admin can select this option. The provisioning process creates a business container for a business user. This option is discussed in more detail in section 10.3.3.

The rest of this section is structured to highlight steps 3 to 6 as described in 10.3.1.

10.3.2.1 Step 3: Owner ID creation

As soon as the device owner chooses to create a new user, the device generates a unique identity for the user that is hidden from the user but used internally by the identification and authorisation service. The unique identifier is assigned to a new user object that also contains other information, such as login name, email address, etc. The device owner supplies this information manually. The Neo model does not prescribe which user properties must be supplied.

10.3.2.2 Step 4: Create initial security objects

After the user identity is created, the provisioning process creates the initial set of security objects associated with the user account. This includes an asymmetric key-pair and a symmetric key. The symmetric key is encrypted using the public key of the asymmetric key-pair and decrypted using the private key.

All the keys generated in this step is stored in the management container and is only available after the device has started and in turn decrypted the management container. The device owner does not have direct access to security keys and is managed internally by the identification and authorisation service.

MAP3.2

10.3.2.3 Step 5: Create container object

After the symmetric key is created for the user, the provisioning process makes a copy of the container template. The newly copied container is encrypted using the symmetric key created in Step 4.

SCP5.1

10.3.2.4 Step 6: Set policy and access rules

Before the new container is mounted, the newly created user account and newly added I/O peripheral is automatically given permissions to the container. When the device owner interactively adds other user accounts, the owner must also select an initial I/O peripheral that can be used by the newly created user account. The access rules for the I/O peripherals and the newly created container is updated accordingly. Once the container owner is logged into the container, the container owner can select other approved I/O peripherals and give them access to the container.

MAP2.3

The identification and authorisation service also apply any device policies to the newly created user accounts and containers. During the initial start-up no policies exist that can influence the container, but after the device has been used for a while, specifically after it may also have been provisioned into one or more company networks, a number of policies may apply.

The interactive user creation process concludes with a newly created user account, with a newly created container where the user has access to. If this is the first user on the device, the user is also registered as the device owner that has permissions to create new user accounts and allow extra I/O peripherals and must approve any business provisioning processes.

10.3.3 Business provisioning

When the Neo device is owned by a business, the steps 3 to 6, as described in10.3.1, differs from the steps described in section 10.3.2. The steps, as they apply for business provisioning are described in sections 10.3.3.1 to 10.3.3.4.

The goal of business provisioning is to first create a container owned by the business, secondly to allow an employee access to the newly created container and thirdly to control the container through access rules and policies. This goal is mostly achieved by the business using a mobile device management system that interacts with the Gateway Controllers already installed on the business network.

The mobile device management system provides an interface for the network administrator to administer the various Neo devices and Gateway Controllers. It provides the interface to approve new Neo devices, assign business users to Neo devices, create access rules and define policies that are applied to Neo devices.

10.3.3.1 Step 3: Owner ID creation

When the Neo device is provisioned for a business the company becomes the owner of the relevant corporate containers that are created on the device. To enable this, each business requires a business identity that is assigned the owner of the corporate container.

During the installation and configuration of the mobile device management system, a unique identifier is created for the business. This identity is also assigned an asymmetric key-pair that is used to encrypt and protect symmetric keys stored on the Neo devices.

This means there is always two identities of importance on a Neo device when it is used for business purposes. The one identity is the business identity; the second identity is the business user identity of the user that is granted permission to access the corporate container. The business user identity is either created during the provisioning process on the Neo device or gets linked to an existing user identity already on the Neo device.

When a Neo device goes through the business provisioning process, the Neo device detects the mobile device management (MDM) system. Once the MDM has been detected, the Neo device establishes a secure connection with the MDM.

During this initial detection method, Gateway Controllers, allow network traffic from un-identified Neo devices, to access only the MDM or if required by the company, the Internet. This enables the provisioning process to progress without approval.

After the Neo device's identity has been created on the MDM, the network administrator either approves the Neo device or denies access. The network administrator also assigns a business user identity to the Neo device after the device has been approved. In the case of approval, the provisioning process continues.

The MDM creates a copy of the business identity on the Neo device. During the initial start-up, a new user identity is also created on the Neo device. The new user identity is created with properties from the corporate network of the business user that will use the Neo device. If the business provisioning process happens in cases where the Neo device belongs to a consumer, no other business user identity is created, instead, the business user identity is linked to an existing user identity on the Neo device. The detail of this link is described in Chapter 9 section 9.4.3.

10.3.3.2 Step 4: Create initial security objects

As soon as the business identity is created on the Neo device, a unique symmetric key is also created on the Neo device. Apart from the unique symmetric key, a unique asymmetric key-pair is also created. The symmetric key gets encrypted with the public key of this unique key pair and can be decrypted using the private key. Depending on the policy defined by the MDM the unique private key, is in turn encrypted using the public key of the business. In cases where the policy does not allow a business container to be used while offline, the Neo device must first request the Gateway Controller to decrypt the unique private key using the private key of the business.

In the case of an initial start-up, the business identity created on the Neo device is also assigned the device owner. This means that the network administrator must manage connections of any new I/O peripherals. This does not mean that the network administrator must approve any new I/O peripherals, but the network administrator can decide to delegate this to one of the user identities on the Neo device.

10.3.3.3 Step 5: Create container object

As soon as the business identity is created, and the unique symmetric key is created, the Neo device creates a new container using the container template that exists in the management container. The container is encrypted using the symmetric key belonging to the business identity.

SCP5.3

In some cases, the business can choose to create more than one secure container, each with its own set of applications, policies, symmetric and asymmetric keys and approved I/O peripherals. In cases where more than one container must be created, the MDM sends additional instructions to the Neo device to create the other containers. Each container has their own symmetric key, which in turn is protected by the asymmetric key-pair created with the container.

10.3.3.4 Step 6: Set initial policy and access rules

MAP2.1

After the container has been created and encrypted, an access rule is automatically created that ensures the business identity is the owner of the newly created container. Initial access rules are also created to ensure the business user identity and approved I/O peripherals have access to the container.

SCP3.2

The identification and authorisation service on the Neo device interfaces with the MDM and ensures that policies defined on the MDM are copied and applied to the Neo device and container.

As soon as the policies are available on the Neo device, the application provisioning process in the newly created container starts and provisions the installation and configuration of any applications required by the policy.

The MDM also interfaces with the Gateway Controllers and create or modify access rules or access rule objects to ensure that the Neo device and corporate container has access to approved network resources.

The business provisioning process ensures that a corporate container is created on the Neo device. It ensures that the business identity has owner permissions on the corporate container and in the case of an initial start-up becomes the owner of the Neo device. The configuration and settings are all managed and approved using an MDM. The network administrator has control over exactly where, when and how the corporate container can be used on the Neo device.

The provisioning process affects the Neo device and containers during initial start-up, but also afterwards, when new users are created or provisioned on a business network. The next section describes the components and steps that take place when a Neo device is started and an identity logs in.

10.4 Run process

The run process is involved when a Neo device is powered on. The goal of the run process is to ensure secure containers are started when required. A container does not start without a user either logging into the Neo device, or remotely initiated by a company. This section describes both of the activities that cause containers to start.

The term started, is used in this context to describe the unlocking of the symmetric key that must be used to mount the encrypted container. As soon as the container has been mounted any automatic services defined in the container starts. These services may include email services, tracking services or any other services required by the applications installed in the container.

10.4.1 Interactive login

Containers may start when a user logs into the Neo device. During the login process, the identification and authorisation service identifies all the containers that the user can access. When all the containers have been identified, the private keys for the containers are used to unlock the various symmetric keys. The symmetric keys are used to mount the different containers and the automated start-up services in the containers are started.

Containers that have a policy to not be available offline works slightly differently. A private key is not available in the management container to unlock the symmetric key. In these cases, the identification and authorisation service tries to connect to the container's Gateway Controller. Where the Gateway Controller is available over the network the identification and authorisation service goes through a process to establish a secure connection. The Gateway Controller unlocks the container's symmetric key after a secure session has been established. The container is mounted and the private key is removed.

When a user logs into the Neo device, containers that the user can access starts, but they can also start using remote commands.

10.4.2 Remote initiated

In some cases, containers can also be started remotely. This is used most often in cases where a company is the owner of a container and for some reason require the services in the container to start. Chapter 11 describes a case where this feature allows companies to use Neo devices as a potential backup location.

Remote commands are initiated by the MDM, which interacts with Gateway Controllers to initiate connections to Neo devices. Every n-minutes, the Neo device gets a list of all the Gateway Controllers that are configured on the device. The Neo device tries to connect to each one and sends a heartbeat message to each one. A heartbeat message is basically an encrypted message identifying the Neo device and related container. In cases where there is a remote initiated command sent by the MDM the Gateway Controller replies to the heartbeat message with a start-up message.

When the Neo device receives a start-up message, the Neo device initiates a secure connection with the Gateway Controller. The Neo device or the Gateway Controller unlocks the symmetric key for the container and the Neo device mounts the container without user interaction. When the container has started, it may open a secure connection to the Gateway Controller and gain access to the network resources on the company network.

Remote initiated commands allow network administrators to ensure data is synchronised in the containers, update applications and many more applications.

10.5 De-provisioning actions

The Neo device does not allow containers to live forever on the device. Device owners or container owners may choose to remove containers from Neo devices. This section describes the process where a device owner interactively chooses to remove a container or where a company chooses to remove a container using a remote initiated command.

10.5.1 Interactive removal

A device owner can remove any container that has been provisioned on the Neo device. This allows a device owner to clean up and remove old, unwanted information.

The device owner initiates a command to get a list of containers on the Neo device. The device owner can then remove any of the containers on the list. When a container is removed, the container is removed as well as the symmetric and asymmetric keys used by the container. The removal process also removes any policies and Gateway Controller information associated with the container.

In cases where the container was owned by a business, a removal message to the company's Gateway Controller is queued on the Neo device. If the Neo device manages to send the message to the Gateway Controller, the Gateway Controller marks the device and container as de-provisioned. This information is sent to the MDM which informs the network administrator that device and container do not have access to the environment anymore.

The removal message stays in the queue for n-amount of days before the Neo device removes the message. This ensures that the Neo device does not try and send a message to a Gateway Controller that might not exist anymore.

10.5.2 Business de-provisioning

There is a number of times where a company would like to remotely remove a container from a provisioned Neo device. This can be in cases where an employee does not work at the company anymore but retained access to the Neo device, or where a consultant had temporary access to a company's network resources but does not require access anymore.

The network administrator initiates a removal message from the MDM. The removal message creates a special status message on the Gateway Controllers. When a Neo device sends a heartbeat message or initiates a connection to the Gateway Controller, the Gateway Controller sends a removal message to the Neo device. The Neo device ensures that the message corresponds to the removal of a container where the business is the owner of.

The removal process removes the container, keys and other container associated information. The Neo device creates a removal message and queues it on the Neo device. The removal message informs the Gateway Controller that the container is removed from the Neo device, which, in turn, sends a message to the MDM indicating that the removal process was successful.

10.6 Conclusion

This chapter described the life cycle of a container on the Neo device. Containers go through a provisioning process, that creates the container. It then spends some time in a state where the container is used and finally is removed from the device when it is not needed anymore by either the device owner or container owner.

The aim of the chapter was to provide greater insight into some of the components that were described in chapters 7, 8 and 9.

Section 10.2 described the different phases of a data and application secure container. It provided an overview of the three stages of a secure container. The stages describe the provisioning process, the run process and finally the de-provisioning process.

Section 10.3 provided details of the different activities that can cause a container to be created. The section described the initial start-up sequence of the Neo device. The initial start-up creates the appropriate device owner identifier and goes further to describe how the I/O peripherals are authorised and the initial container is created. The section further described the process when a device owner interactively creates a new user on the Neo device or where a device is provisioned into a company network.

Section 10.4 described when containers can start. They can start either when a user logs into the Neo device or when they receive a start command from a corporate network. Starting a container mounts the container and starts any automatic service configured in the container.

Section 10.5 describes what happens when a container is removed from the Neo device. Device owner or container owners can remove containers interactively or companies can initiate remote removal commands. Business containers always send a removal message to the associated Gateway Controller to ensure that the company knows the container has been removed from the device.

The next chapter provides a number of interesting implementation scenarios where the Neo device and Neo model is used for personal and\or business use.

Introduction

- Chapter 1 Introduction

The end game

- Chapter 2 Model Overview

Literature study

- Chapter 3 Isolation properties in mobile operating systems (SCP)
- Chapter 4 Isolation techniques in other technologies (SCP)
- Chapter 5 An overview of peripheral security (MAP)

Bridge

- Chapter 6 The drivers for a new model

Building the Neo model

- Chapter 7 Overview of the Neo Model
- Chapter 8 The Secure Container Property
- Chapter 9 The Mutual Authentication Property
- Chapter 10 Container life cycle
- **Chapter 11 Novel Implementations**

Evaluation and summary

- Chapter 12 Potential implementation of Neo Model
- Chapter 13 Conclusion and future work

Chapter 11 NOVEL IMPLEMENTATIONS

11.1 Introduction

During the development of the Neo model, two novel implementation scenarios were envisaged. Both of the implementations have been published as part of conference proceedings (du Toit & Ellefsen, 2015) (du Toit et al., 2016).

The aim of this chapter is to highlight the unique uses cases that are not possible without using something like the Neo model. This chapter provides a discussion of the possibilities of the Neo model. The chapter emphasises that the Neo model does not only achieve the primary goal set in the thesis, but it also opens up unique solutions for other problems.

The two scenarios are:

- **Section 11.2. Location-aware authorisation**. The Neo model and the associated network architecture allow companies to set different access and control levels, depending on the physical location of the Neo device. Section 11.2 describes the problem that the implementation resolves. It further describes the infrastructure components that provide the solution and the way in which it can be configured.
- **Section 11.3. Disaster recovery options**. The Neo model provides a novel location for company backup data. Section 11.3 provides an overview of the problem that the scenario is trying to solve. It also provides a description of the network architecture required to implement the solution and then discusses some of the implementation scenarios possible.

The first scenario described is the way in which different authorisation levels are used depending on where the Neo device is used on company premises.

11.2 Location-aware authorisation

This section focuses on the first scenario which is a special use case that allows authorisation of data and applications to take into account the physical location of a device. This section is structured as follow:

- **Section 11.2.1** describes some of the problems that can exist in corporate environments. The problems described does not only exist in big corporate companies, but it can also exist in smaller organisations.
- **Section 11.2.2** provides a discussion on the network architecture and other environmental requirements necessary to address the problem. This section shows that the Neo model is ideally placed to address the problem.

- **Section 11.2.3** describes the access control rules defined on the Gateway Controllers, as well as the policies and the effect that the policies have on the Neo devices when connected to the different areas of a network.
- **Section 11.2.4** provides a conclusion to the problem and solution described in this section.

11.2.1 Problem description

It is common practice to bring a number of smart devices to meetings in companies. These devices are used to display presentations, view meeting minutes, but they can also be used to record information in the meetings. Some meetings also carry a certain level of confidentiality and sensitivity. The information in these meetings should stay confidential and should be handled with a certain level of sensitivity.

Today's mobile devices can capture information in these meetings either legitimately or maliciously. In both cases, the company needs to ensure the information should stay confidential. The problem faced by the company is that it is difficult for them to control what users of the devices can do, given different environments. The company would like to allow their users access to a broad range of applications and functionality, but given certain environments and situations, tighten the controls.

Another area of control is found while working with minors. Photographs of minors are strictly controlled in cases where children are involved. The United Kingdom has strict legislation regarding the use of photos of children taken by childminders (Northamptonshire Childminding Association, 2015) (United Kingdom, 1998). In these cases, the employer of the childminder may need extra control over their employee's mobile devices with built-in cameras, while the childminder is on the premises of the employer.

Employers rely on mobile device management systems (MDM) to address bring your own device (BYOD) challenges. MDM systems allow companies strict control over enrolled mobile devices. MDM systems by default do not allow fine-grained control over devices given certain external factors, like the location.

The Neo model allows companies to control Neo devices given their physical location inside a company's premises. Control based on geographic location requires specific infrastructure requirements. The next section describes the infrastructure components that allow companies to apply certain policies on Neo devices depending on their location and other factors.

11.2.2 Infrastructure architecture

When the Neo device is used in a corporate environment, the Neo model allows the company to define rules that apply to Neo devices. The rules affect both the Neo device hardware and the containers inside the Neo device. This section describes the interaction of the various architectural components in the Neo model that allow companies to control the Neo devices in specific areas.

Chapter 8 section 8.6 described the concept of container controls. Container controls and device controls are set through policies. A Gateway Controller provides policies to the Neo device. The company network administrator creates policies using an MDM system interface.

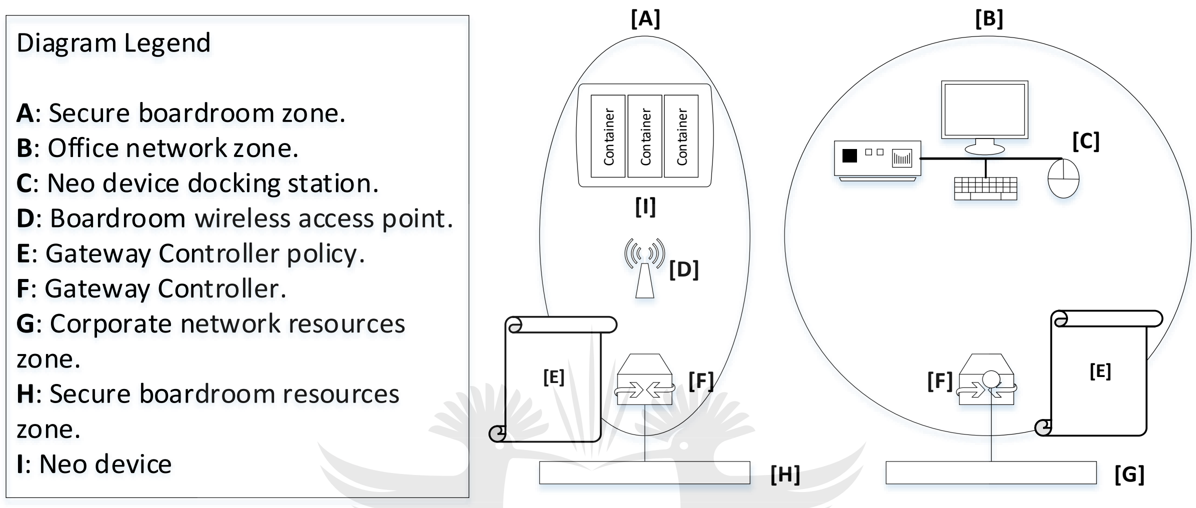


Diagram Legend
A: Secure boardroom zone.
B: Office network zone.
C: Neo device docking station.
D: Boardroom wireless access point.
E: Gateway Controller policy.
F: Gateway Controller.
G: Corporate network resources zone.
H: Secure boardroom resources zone.
I: Neo device
[A]
[B]
Container
Container
Container
[I]
[D]
[C]
[E]
[F]
[F]
[E]
[H]
[G]

FIGURE 48: DIFFERENT ZONE POLICIES (BY AUTHOR)

Figure 48 describes a network environment where a company has two network zones (A + B) and network resources (G + H). The company has an office network zone (B) where the Neo devices connect to for normal business purposes. While connected to this zone, the devices have access to the corporate network resources (G). The Neo devices normally connect to the corporate network through docking stations (C) located on the employees' desks.

The company also has a boardroom with its own dedicated wireless access point (D) that allow Neo devices (I) to connect to secure boardroom resources on the network (H). Both the boardroom and corporate networks are defined inside the Gateway Controllers (F) as zone resources. The network administrator defines policies (E) for the secure containers in the Neo devices while connected to the office or secure boardroom zones (A + B). The Gateway Controllers (F) connect the Neo devices from the office and secure boardroom zones to the corporate network resources and secure boardroom resources.

The company painted the boardroom with WiFi blocking paint (Smith, 2012) (Lee, 2009), to ensure WiFi signals cannot escape the boardroom. This means that the WiFi access point can only be used, while employees are physically inside the boardroom.

The infrastructure described in this section is required to enforce different rules for Neo devices depending on where they are used on the company network. The components defined and created that control these rules are a combination of access rules and policies defined by the network administrator. The next section describes the access control rules and policies for this specific scenario.

11.2.3 Policy and access control rules.

The company would like to ensure that while employees are in the boardroom that employees can only use the voice recorder application to record the meeting. These recordings are used for minute purposes and should be backed up to a voice recording server on the corporate network.

The company also allow employees to use any of their business applications while working from home or at the office. The company has seen that giving their employees this freedom increases productivity, but the company are still concerned with the security of the company data, while the user is on the business premises, but also when working from home.

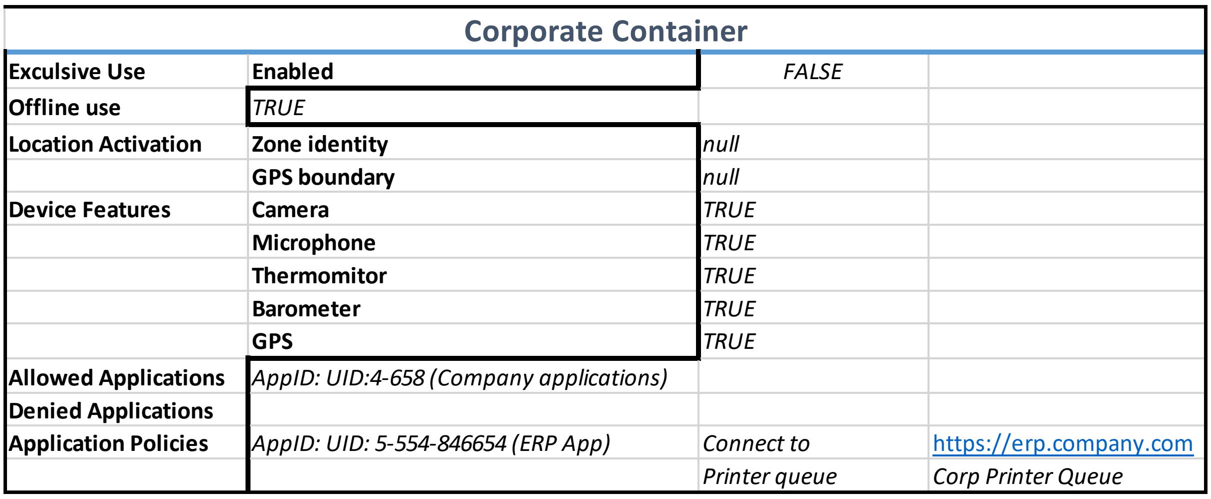
To accomplish the above requirement, the company created two company secure containers, managed by two separate policies stored on the company's Gateway Controllers. The Gateway Controllers also control access by the Neo device to network resources using access control rules defined on the Neo devices and Gateway Controllers.

- **Section 11.2.3.1** describes the two policies created to implement the given scenario.
- **Section 11.2.3.2** describes the access control rules that ensure access to the containers and objects on the corporate network.

11.2.3.1 Secure container policies.

The one policy, depicted in Figure 49, applies to the corporate container that is provisioned on the Neo device. The policy does not dictate exclusive use, this means that other containers may also be active while this container is running. This allows employees to use their personal containers and the corporate container at the same time, but still keeping the data separate from each other.

The corporate container policy further states that the user can use any of the Neo device features inside the container and allow any of the company applications to be installed and running inside the container. The company has a number of application policies, but only two are depicted in Figure 49 for demonstration purposes, which shows the network address that their ERP application connects to, as well as the printer queue that will handle any printing from the application.

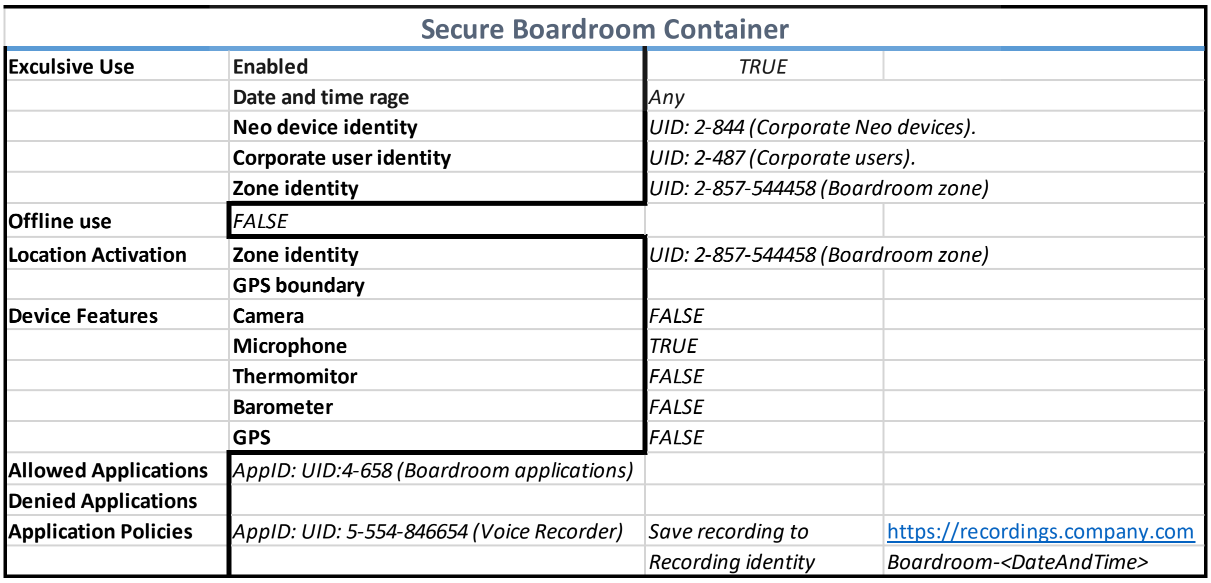


Corporate Container			
Exculsive Use	**Enabled**	*FALSE*	
Offline use	*TRUE*		
Location Activation	**Zone identity**	*null*	
	GPS boundary	*null*	
Device Features	**Camera**	*TRUE*	
	Microphone	*TRUE*	
	Thermomitor	*TRUE*	
	Barometer	*TRUE*	
	GPS	*TRUE*	
Allowed Applications	*AppID: UID:4-658 (Company applications)*		
Denied Applications			
Application Policies	*AppID: UID: 5-554-846654 (ERP App)*	*Connect to*	https://erp.company.com
		Printer queue	*Corp Printer Queue*

FIGURE 49: CORPORATE CONTAINER POLICY (BY AUTHOR)

The second policy defined by the network administrator is the policy for the Secure Boardroom. This policy is depicted in Figure 50. This policy defines the exclusive use of the Secure Boardroom container. This means that the container is the only container to be active. All other containers are paused when this container is active. It further states that the container can be activated at any time, by any of the corporate Neo devices and corporate users, but can only be activated while inside the Boardroom zone. The Neo device also ensures that the Secure Boardroom container activates automatically when the device detects that it is physically inside the secure boardroom.

The policy has a number of other rules, that ensure that the container can only be active while connected to a Gateway Controller. It disallows all device features, except for the microphone. The policy also only allow the boardroom applications from functioning inside the boardroom container, with specific application policies for the voice recorder application.



Secure Boardroom Container			
Exculsive Use	**Enabled**	*TRUE*	
	Date and time rage	*Any*	
	Neo device identity	*UID: 2-844 (Corporate Neo devices).*	
	Corporate user identity	*UID: 2-487 (Corporate users).*	
	Zone identity	*UID: 2-857-544458 (Boardroom zone)*	
Offline use	*FALSE*		
Location Activation	**Zone identity**	*UID: 2-857-544458 (Boardroom zone)*	
	GPS boundary		
Device Features	**Camera**	*FALSE*	
	Microphone	*TRUE*	
	Thermomitor	*FALSE*	
	Barometer	*FALSE*	
	GPS	*FALSE*	
Allowed Applications	*AppID: UID:4-658 (Boardroom applications)*		
Denied Applications			
Application Policies	*AppID: UID: 5-554-846654 (Voice Recorder)*	*Save recording to*	https://recordings.company.com
		Recording identity	*Boardroom-<DateAndTime>*

FIGURE 50: SECURE BOARDROOM CONTAINER POLICY (BY AUTHOR)

The Secure Boardroom Container – policy ensures that it activates the secure boardroom container, de-activates any other containers and strictly controls the device features and applications that are allowed to run inside the container.

With the two policies in this section, the company is in full control of the secure containers. The next section describes the access control rules that ensure access to the containers and objects on the corporate network.

11.2.3.2 Access control rules

The access control rules used inside the Neo model ensure access control to containers by users and I/O peripherals, but also ensure access is controlled by network resources. The access control rules are created on the Neo device when containers are provisioned, but they are also found inside Gateway Controllers. The container owners define access control rules stored on both Neo devices and Gateway Controllers. In our scenario, the company creates the access control rules to control access to data on the Neo devices, but also data on network resources.

Figure 51 shows the access control rules created on a Neo device that ensures that a user account using a specific I/O peripheral has access to both the Corporate Container (Access Rule 1) and the Secure Boardroom Container (Access Rule 3). The access control rules also ensure that the Corporate Container can access the Corporate Network Resources defined on the Gateway Controller (Access Rule 2). Access Rule 4 specifies that the Secure Boardroom Container can access the Gateway Controller and the network resources on the Secure Boardroom Network Zone.

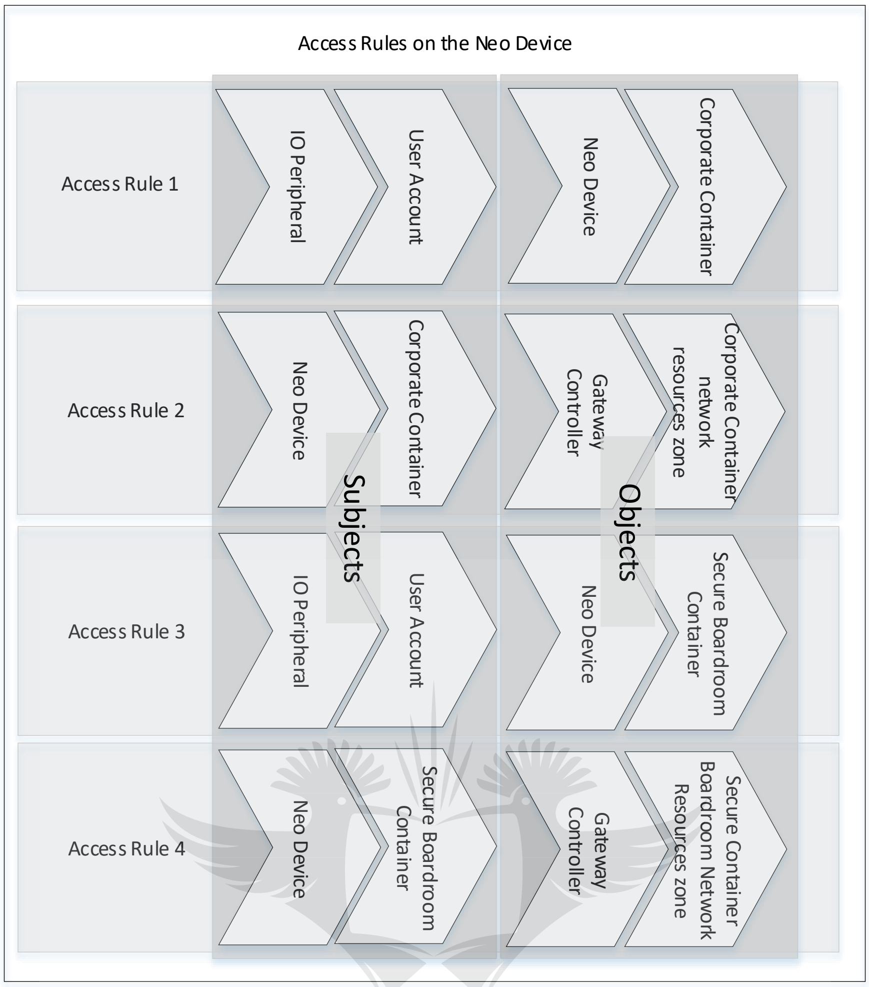


FIGURE 51: CORPORATE CONTAINER SPECIFIC ACCESS RULES (BY OWNER)

It should be noted that the "Neo Device" and "User Account" subjects and objects can also be group accounts, which simplifies the administration of the access control rules significantly.

The access control rules on the Neo device is created initially when the secure containers are created, but they are also updated every time the Neo device connects to a Gateway Controller. This ensures that an administrator can change the access to different containers throughout their life.

Figure 52 shows the access rules on the Gateway Controllers that ensure access to the different network resources is controlled. Access Rule 1 is created on the Gateway Controller that controls access to the Corporate Container network resources and Access Rule 2 is created on the other Gateway Controller.

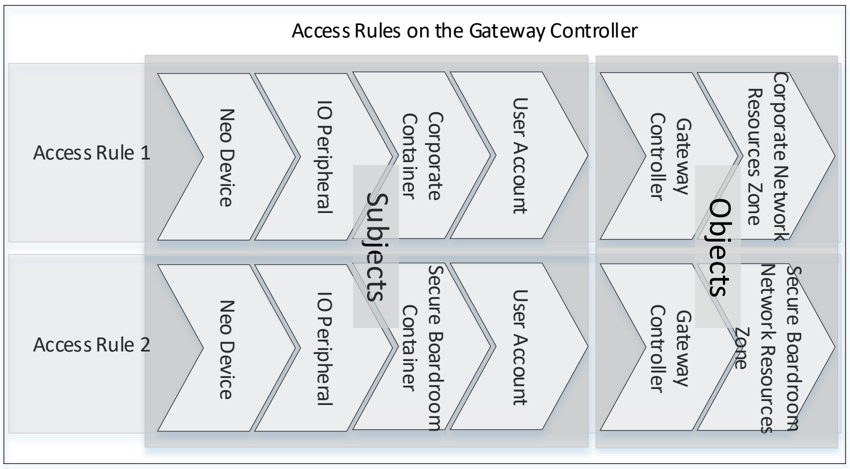


FIGURE 52: GATEWAY CONTROLLER ACCESS CONTROL RULES (BY OWNER)

The access control rules described in this section are created and managed by the network administrator. The Neo device access control rules are necessary to ensure access by the user and I/O peripherals to the secure containers on the Neo device. The access control rules on the Gateway Controller acts like firewall rules, which control access to the network resources.

11.2.4 Conclusion

Some companies need to control how devices can be used depending on their location within the company. The example of a restricted secure boardroom was used to demonstrate the ability of the features in the Neo model to ensure only a special secure boardroom container can be used while employees are inside the boardroom.

Special container policies ensure that the secure boardroom container can only be activated while within the confines of the boardroom. The policies also ensure that no other containers may be active while the secure boardroom container is active. The secure boardroom container automatically starts when the Neo device detects the device is inside the secure boardroom. This has an effect that only the secure boardroom container may be active while the Neo device is inside the secure boardroom.

This section also showed how the policies, first defined and described in Chapter 8 section 8.6, are used in a real-world example. The Gateway Controller and the Access Rules controlling access to network resources has also been elaborated. The access control rules and the role of the Gateway Controller were first introduced in Chapter 9 section 9.5.

The next section describes a scenario where the Neo model architecture can be used to implement a special backup application for companies that make use of the secure containers on Neo device to ensure data confidentiality and control by the company.

11.3 Disaster recovery options

One of the properties of the Neo model is the secure container property (SCP). The secure containers in the Neo model allow a data owner full control over their data. The data owner can decide who has access to their data and which I/O peripherals may access their data.

The SCP can be exploited in unique ways. One of the scenarios unlocked by this ability is to provide the company with an extra option for offsite backups.

The content of this section is published as a conference article for IST Africa 2016 conference, with the title "Bring Your Own Disaster-Recovery (BYODR)" (du Toit et al., 2016).

The structure of this section is as follow:

- The problem addressed by this section is described in **section 11.3.1**.
- The solution implementation is described in **section 11.3.2** and
- A conclusion for this problem is defined in **section 11.3.3**.

11.3.1 Problem description

Small companies usually do not have a dedicated IT department and one of the aspects that are normally lacking is adequate backup and recovery strategy (Preimesberger, 2009) (Eddy, 2013). The lack of backup and recovery is especially problematic in cases of disaster recovery. In cases of disaster recovery, not only do the company need adequate backup, but may also require off-site storage of their backed up data.

A popular solution used is to backup to the cloud (Mah, 2014). Backing up to the cloud, unfortunately, does not come without certain risks. These risks include unauthorised access, government influence and vendor-focused attacks (Neumann, 2014).

Other off-site solutions include backing up to external media, like tapes or external hard disks and then either through third-party services or employee responsibilities, removing the backup from the site. The solution described in the next section makes use of well-documented distributed backup system architecture but leveraging the Neo model to provide the confidentiality of the backed up data.

11.3.2 Solution implementation

A solution that enables small companies to ensure secure offsite backups can be implemented using distributed backup architecture and the architecture described in the Neo model. Distributed backup or distributed peer-to-peer file storage describes the ability to take data, divide the data into a number of fragments. From the data fragments, a number of redundancy fragments are created and all of these fragments are distributed to a number of storage locations (Giroire et al., 2010).

The Neo model assists in storing the data fragments and redundancy fragments in secure containers on Neo devices. The Neo devices act as an offsite storage of backup information. In the case of a disaster, the data stored inside the containers are retrieved to rebuild the backed up data. This section of the chapter is organised as follow:

- **Section 11.3.2.1** provides a short overview of peer-to-peer storage.
- **Section 11.3.2.2** describes how the architecture in the Neo model makes it possible to implement this solution.

11.3.2.1 Distributed peer-to-peer storage

The concept of peer-to-peer storage is also known as a global distributed file store. The distributed file store allows clients to store files in the distributed file store, with the assurance that the files are retrievable and the confidentiality and integrity of the files are intact (Bolosky et al., 2000).

The core idea of a peer-to-peer storage system is to make use of erasure codes. Erasure codes use the principle of taking a piece of data and breaking it up into s number of fragments. A further r-fragment is then generated that is then used to ensure that if one or more of the original s-fragments are lost, the s+r fragments can be used to recreate the original data. This type of coding is also known as forward error correction (Luby et al., 1997).

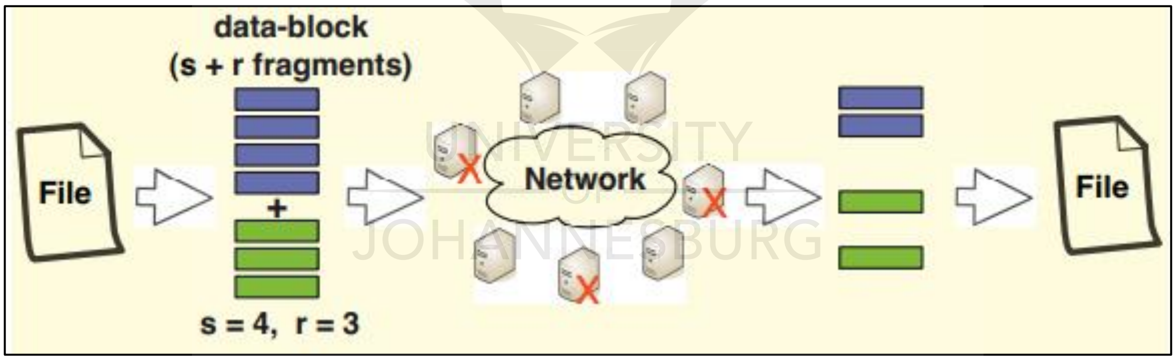


FIGURE 53: THE CONCEPT OF PEER-TO-PEER STORAGE (GIROIRE ET AL., 2010)

Figure 53 describes the use of erasure codes on a data file. The data file is fragmented into four s-fragments and three r-fragments. The seven fragments are then stored on various computers on the network. During file recovery, any four of the seven fragments can be used to recreate the original file (Giroire et al., 2010).

The distribution and re-assembly of the fragments are managed by a distributed application that keeps track of the number of application members on the network and how alive these members are. There are a number of these applications, which include Tahoe-LAFS (Tahoe-LFS, 2017), Resilio Sync (Historically known as Bittorent sync) (Resilio, Inc., 2017) and Freenet (The Freenet Project Inc. , 2017).

The minimum number of nodes on a peer-to-peer storage system is dependent on a number of factors. Some of these factors include the amount of data to store, the available storage on peers, the rate of data change, the redundancy level of the data and the aliveness of the peers (Are peers actively communicating or have they stopped communicating?).

More data would require potentially more peers. Peers with more space means fewer peers to potentially store the total amount of data. When the rate of change in the data increases, it means that you may need more peers to allow for positive growth otherwise you may need fewer peers if the growth is negative. The higher the redundancy level of the data the more space is required. To ensure data can be retrieved, the redundancy level should be offset against the aliveness of the peers. When peers have a low percentage of aliveness, more redundancy is required to ensure data can be retrieved.

In peer-to-peer storage systems, a network node can either be a client or both a client and server. This means one node can use the network as a distributed file system for the node's files or it can contribute to storage and act as a storage node on the network. The Neo model leverage this architecture, but provides an extra layer of assurance of control. The next section describes how the peer-to-peer storage application can be integrated into the Neo model architecture.

11.3.2.2 Distributed backup using the Neo model architecture

Using the Neo model to provide a peer-to-peer storage environment, gives a company the ability to identify and control the nodes on which data is stored. The implementation of a peer-to-peer storage system is summarised in Figure 54. At the top of the diagram is one or more computers with company data. The data on these computers are backed up to local storage by a special backup system. The backup system creates a special backup container, with a backup agent inside the container. The agent based container is distributed through the Gateway Controller to a number of Neo devices.

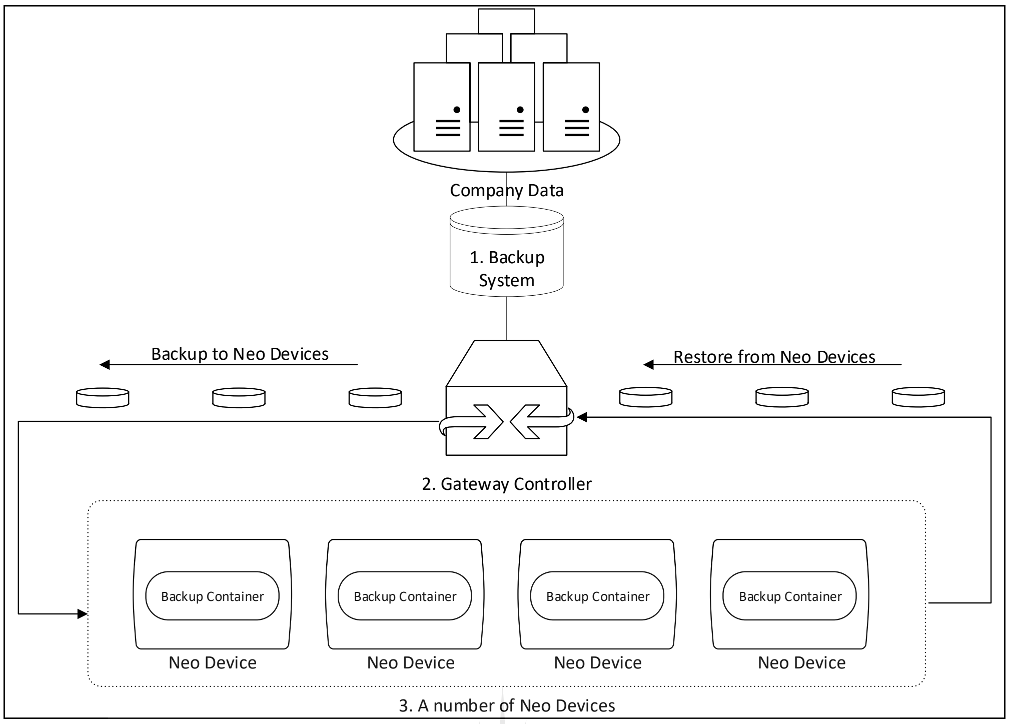


FIGURE 54: PEER-TO-PEER BACKUP USING THE NEO MODEL ARCHITECTURE (BY OWNER)

The implementation of the system makes use of a special backup application. The backup application is responsible for a number of things:

1. The backup application ensures that data from the various company data sources are backed up to a single storage area.
2. The backup application gives the network administrator the ability to enrol a number of Neo devices, to the collection of approved Neo devices. It is the administrator's decision to include only company-owned Neo devices or approve any Neo device that makes itself available to the backup system.
3. The backup system ensures that a backup container is defined on the Gateway Controller and sets the container policies and access control rules to allow the containers to be provisioned on the approved Neo devices. The access control rules and policy rules do not allow device users from accessing the container.
4. As soon as the Neo devices contact the Gateway Controller, the backup container is created on the Neo devices, with a software agent of the backup system inside the backup container.
5. When the backup container activates, the backup agent registers on the backup application on the corporate network.
6. The backup application ensures that there is a minimum amount of Neo devices registered to ensure all the data could be backed up with the correct level of redundancy.
7. The backup application uses the concepts of a peer-to-peer storage system to fragment the data and distribute it to the various backup containers on the registered Neo devices.

8. The backup application ensures a copy of a special password, called the recovery password, is sent to one or more Neo devices. This password is used in case of a disaster to recover the metadata of the backup system. The metadata includes at least the identification information of all the Neo devices that are part of the backup system.

In the event of a disaster, the network administrator needs to provide some basic infrastructure in order to start recreating the backup data on the central backup system. A disaster recovery happens in two phases. The first phase is to recover the backup system with all the backup data. The second phase is to recover the individual network applications from the central backup system.

Only the first phase is relevant for the discussion in this document. The second phase uses the recovery techniques appropriate to the central backup system to ensure the backed up data is restored to the various computers on the company network.

The network administrator needs to ensure that there is at least one Gateway Controller available for the Neo devices and a basic installation of the central backup system. The central backup system does not have any information about the data backed up or information about any of the Neo devices containing the backup data at this point in time.

The network administrator uses the recovery password on the Neo devices designated for this role to initiate the disaster recovery process. The recovery password activates the backup container on the Neo device. The backup application in the backup container is set to a disaster recovery mode, contacts the central backup system, and recovers the metadata from the Neo device to the central backup system.

The central backup system is set to recovery mode. Once the central backup system has the metadata restored, it starts initiating communication with the rest of the backup containers on the rest of the Neo devices and sets all of them to recovery mode. Once the Neo devices are in recovery mode, the data fragments are sent to the central backup system. The central backup system releases data to phase two as soon as the data has been recovered from the Neo devices.

11.3.3 Conclusion

The architecture described in the Neo model allows application developers and system developers to develop or extend their backup solutions to include a decentralised offsite backup solution, making use of Neo devices as an offsite repository. The Neo model provides data owners with a high level of assurance that the data being backed up is secure and can only be used for recovery purposes.

Some of the arguments against the use of the Neo model architecture as a backup solution may include aspects such as the device owner's reluctance to allow companies to use resources on their devices. Mobile devices may have limited storage capacity and device owners may not want to allow companies to reduce the already limited storage. Apart from the storage capacity, the backup and restore process also utilise network capacity and may incur costs to the owner of the mobile device. The time it takes to finish phase one of the recovery process may also detract companies from adopting this approach.

Advantages of the solution mean that in controlled environments, small company owners may find the solution to be a cheaper alternative to other offsite backup solutions. Data owners are also assured that their data is not stored in one central cloud location, minimising the risk of an attacker accessing one universally available location. Device owners may also be "paid" through company incentives such as subsidising device or mobile data costs, making the adoption of the solution by company employees more attractive.

11.4 Conclusion

The SCP with the MAP of the Neo model opens the gates to unique solutions. Two of the unique solutions explored include the ability to manage location-based access control levels of data and applications and an offsite backup solution for disaster recovery purposes. Both these solutions have been explored, documented, and presented at international conferences.

The Neo model fulfils the primary goal of this thesis, which is to provide a model that allows users to use a mobile device for both personal and business purposes and ensuring that the data and applications are isolated and fully controlled by the data owner. The Neo model also provides extra avenues of solutions otherwise not available for data owners.

This chapter investigated two use cases for the Neo model and the underlying architecture. The first use-case was the ability of companies to control, not only the data stored on mobile devices but to control the mobile devices themselves given their physical location. The use of a secure boardroom was demonstrated. The Neo model architecture allows an administrator to lock down Neo devices, while they are physically located inside the secure boardroom. Only a special secure boardroom container is allowed to be active while the device is inside the boardroom, exposing only a few applications to the device users. The secure boardroom container ensures any data generated by the applications while being used inside the boardroom are secured in the container and are only uploaded to pre-configured secure servers.

The second use-case described how the number of Neo devices can be used to store backup data. The Neo devices each store a number of data fragments from the backup data stored on a backup server. The data fragments are distributed to a backup application running inside a locked down backup container on the Neo devices. The backup container is not accessible by the device owner and can only be activated remotely or through a special recovery password in times of data recovery. The solution builds on the advantages and strengths of a distributed peer-to-peer storage system. The solution provides another layer of assurance to the business management that the backed up data is locked down inside secure containers, with a certain level of data redundancy.

The SCP and MAP allow for solutions not easily available using today's mobile architectures. The next chapter discusses the potential for implementing the Neo model using the technology available today.

Introduction

- Chapter 1 Introduction

The end game

- Chapter 2 Model Overview

Literature study

- Chapter 3 Isolation properties in mobile operating systems (SCP)
- Chapter 4 Isolation techniques in other technologies (SCP)
- Chapter 5 An overview of peripheral security (MAP)

Bridge

- Chapter 6 The drivers for a new model

Building the Neo model

- Chapter 7 Overview of the Neo Model
- Chapter 8 The Secure Container Property
- Chapter 9 The Mutual Authentication Property
- Chapter 10 Container life cycle
- Chapter 11 Novel Implementations

Evaluation and summary

- **Chapter 12 Potential implementation of Neo Model**
- Chapter 13 Conclusion and future work

Chapter 12 POTENTIAL IMPLEMENTATION OF NEO MODEL

12.1 Introduction

The primary goal of this document is to develop a model that describes a secure mobile operating environment that enables a mobile device to be used for both personal and business purposes (Chapter 1 section 1.4). The primary objective further states that personal and business data and applications must be isolated from each other and allow data owners full control over who can access the data and applications. If this was the only objective then it can be argued that many of today's mobile operating systems already fulfil this requirement. There is, however, a secondary objective, which states that the mobile operating environment should securely interact with multiple I/O peripherals.

The primary objective together with the secondary objective is problematic with today's mobile operating systems. The control aspect lies in most cases with the device owner, who can decide which I/O peripherals have access to which types of data. Even in solution implementations where business data are isolated from personal data, the device owner can still control what type of peripherals have access to the business data.

The Neo model fulfils both the primary and secondary objectives. The architectural components that make up the model consist of a number of components that enables the SCP and the MAP.

The goal of this chapter is to explore the viability of implementing the components that enable the SCP and MAP using the technology available today, without having to develop an operating system from scratch. This chapter discusses the possible implementation of the Neo model using technology and standards available at the time of writing this document.

Some aspects of the content of this chapter were published as part of conference proceedings (du Toit & Ellefsen, 2017).

The chapter is structured as follow:

- **Section 12.2** summarises and highlights the various components that enable the SCP. The components are then evaluated and the implementation possibilities of the various components are discussed.
- **Section 12.3** does the same but with the focus on the MAP. For each of the components, the problem and the viability of implementing it using today's technologies are explored.
- **Section 12.4** concludes the chapter providing an overall statement of the implementation possibility of the Neo model.

12.2 Implementing the secure container property

The SCP describes the ability of the Neo model to create containers that isolate data and applications from data and applications with a different ownership. The Neo model describes the ability of each container to isolate not only the data or applications but also the processes and abstracted devices.

This section evaluates each of the components that make up the SCP against its ability to be implemented using some type of technology or component available today.

This section is structured as follow:

- **Section 12.2.1** lists the various components of the SCP and goes through the evaluation process.
- **Section 12.2.2** provides a summary of the results from section 12.2.1.

12.2.1 The components enabling the SCP.

Chapter 8 that described the SCP is used to help identify the various components and activities that enable the SCP. The various components that enable the SCP are summarised in the following sections. Each section provides a brief overview of each of the components.

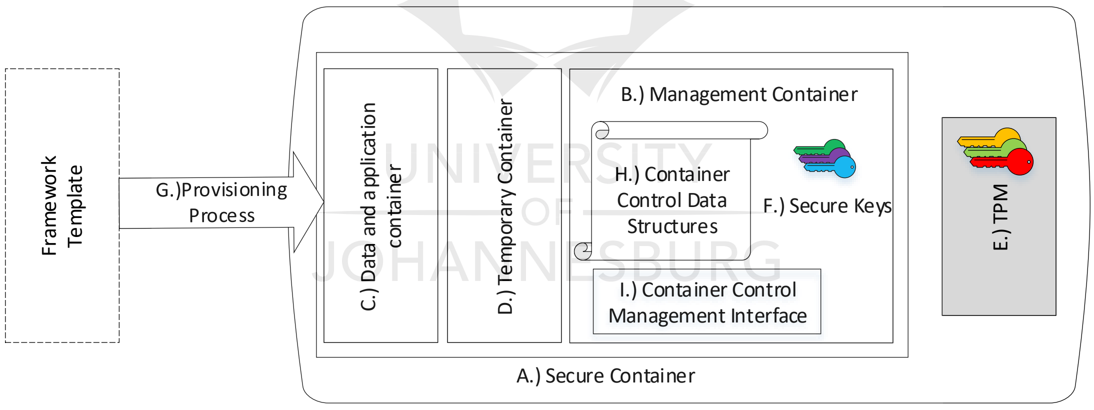


FIGURE 55: SECURE CONTAINER COMPONENTS (BY AUTHOR)

Figure 55 provides a visual reference to the various components. Each of the components in the diagram is described and the implementation possibility is discussed in the various sections referenced in the following list. The list uses the letter that corresponds to the component in Figure 55 and also references the section that discusses the component:

A.) **Section 12.2.1.1**. The primary component, containing three separate components in Figure 55, is the **Secure container (A)**. The secure container is the generalised concept of the three categories of containers.

B.) **Section 12.2.1.2**. The first of the three secure container categories is the **Management container (B)**.

C.) **Section 12.2.1.3**. The secure containers storing user applications and data is known as the **Data and application container (C)**.

D.) **Section 12.2.1.4**. **Temporary containers (D)** are used for network access and security.

E.) **Section 12.2.1.5**. The keys that enable the confidentiality of the Management container are located inside the **TPM (E)**.

F.) **Section 12.2.1.6**. The **keys (F)** responsible for the confidentiality of the various data and application containers and temporary containers are stored in the Management container.

G.) **Section 12.2.1.7**. The management container to provision new data and application containers use a Framework template. This process is known as the **Provisioning process (G)**.

H.) **Section 12.2.1.8**. Inside the Management container are a number of different data structures and services. The data structure that enables the management and control of the data and application container is the **Container control data structures (H)**.

I.) **Section 12.2.1.9**. The configuration and modification of the container control data structures occur through a **Container control management interface (I)**.

12.2.1.1 Secure container.

As defined in Chapter 8 section 8.2, a secure container is defined as a strongly isolated computing environment with a number of services and abstracted devices available to the services inside the container. The best way to visualise a secure container is by thinking about it as a virtual machine. The concept of a virtual machine describes the essence of a secure container.

The level of isolation should be such so that one container cannot access another container, with the exception of the management container. The management container should be able to manage and control the data and application containers and temporary containers stored and running in the system.

The secure container concept ensures applications, services and data inside a container are strongly isolated and protected from other containers. Access to the containers is controlled. The access control is described in more detail in section 12.3 of this chapter.

The architecture of the secure container can be mapped against the virtual machine architecture provided by the Xen hypervisor (The Linux Foundation, 2013) or Qubes-OS (Rutkowska, 2016). It may even be possible to use Linux Containers, except it will not fulfil the requirement of using strong isolation (Linux Containers, NA). Qubes-OS uses Xen as the underlying hypervisor with the management of the system running in a privileged Dom0 with individual virtual machines running as separate DomU virtual machines.

The advantage of the Xen hypervisor is that it is lightweight and a number of projects have already been launched to enable it for mobile operating system purposes. The first major breakthrough was with the enabling of the virtualisation extensions in the ARM processor. The Xen Hypervisor has since been ported to support the ARM processor.

The significance of the supporting ARM processors is because most mobile devices use ARM processors. This means that if the Neo model should be implemented the Neo device needs to be manufactured, most likely using an ARM processor.

The number of Xen implementations on ARM has been limited, but CentOS did provide a distribution specifically for the ARM64 processor (The CentOS Project, 2017). A number of people have been able to install this distribution on a Raspberry Pi 3 (Raspberry Pi Foundation, 2017) (The CentOS Project, 2017) . At the time of writing this, Qubes-OS has not been ported to run on the ARM processor.

The secure container concept, defined in the Neo model, may be implemented as virtual machines using a Xen hypervisor on an operating system like CentOS or Qubes-OS. Implementing a system that enables the concept of a secure container is possible, but the secure container, with all of its various aspects, as defined in the Neo model, may require significant modification.

12.2.1.2 Management container

The management container is a special container that administers the Neo device and provides an environment in which certain control services run. Users of the system cannot install extra components into the management container and the management container does not provide any user applications. Users interface with the management container during login and when authorising access to I/O peripherals. The authorising of I/O peripherals are described in more detail in Section 12.3 of this chapter.

The Management container is analogous to the Dom0 domain of the Xen hypervisor or the Dom0 in the Qubes-OS. In the Qubes-OS implementation, Dom0 does not have any access to the network, instead, it has a special network domain, through which all network traffic is routed. The networking and I/O peripheral access control aspects are described in more detail in Section 12.3 of this chapter.

From a secure container management perspective, a special mounting service is required in the management container that can make use of the security keys stored in the Management container to handle the encryption of the data and application containers as well as the temporary containers.

Other services and data structures that are categorised as control structures are described in more detail in sections 12.2.1.8, 12.2.1.9 and 12.3.5 and may need to be specifically developed for the Qubes-OS if the Qubes-OS is used as the basis for the Neo model. The Neo model differs from the basic Qubes-OS, which allows any logged in account to access any AppVM. In the case of the Neo model, logged in users will not have root access in the Dom0 domain as in the case of Qubes-OS, instead, each user only has access to their respective data and application containers.

12.2.1.3 Data and application container

The data and application containers are focussed on owner-centric control. This means there is at least one data and application container per user account on the system. These containers are simple containers with their own applications and abstracted devices.

The data and application containers are under full control of the services running in the management container. This means that the management container can control which users, I/O peripherals and network traffic can access a data and application container.

Data and application containers fall neatly into the architecture provided by Qubes-OS.

12.2.1.4 Temporary container

Temporary containers fulfil the role of network or data protection buffers. The Neo model identifies two types of temporary containers. Category one containers act as firewall containers, shielding data and application containers from direct network access. Multiple secure containers can share one category one temporary container. The networking domain in Qubes-OS can easily fulfil the requirement of a category one temporary container.

Category two temporary containers are used when data is shared with the container. The Management container creates the temporary containers automatically when users want to copy data between containers or share data with other users on the network. They are also used in cases where a company wants to send updated control information to a corporate container or provision a new corporate container.

Qubes-OS provide a mechanism to create new temporary containers on the fly. The model of Qubes-OS, however, does not support these temporary containers to access the network directly. Even with this restriction, the functionality required for a temporary container to send or receive information from the network is possible using the networking domain in Qubes-OS.

A number of interfaces and services are required in the management container to automatically create the temporary containers, configure the networking domain and interface with the application and data containers dynamically. These extra interfaces and services would need to be custom developed for the Neo model.

Even though temporary containers are being used by the system, the temporary containers should be transparent and the user should not be made aware that an application is currently running inside a temporary container. Qubes-OS provides a very clean window manager, with user interface clues to indicate which AppVM a user is currently working in. This functionality needs to be updated so that temporary containers should not display as a different container from the normal data and application container.

A very important role of the temporary container is to act as an interface between the data and application container and foreign data sources. The temporary container acts as a sandbox for any applications. Changes to the data in the temporary container should be sanitised and synchronised back to the actual data and application container. A significant amount of planning and research is needed to describe how the sanitation and synchronisation process may function. The sanitation problem itself should ensure that only valid changes are synchronised to the data and application container, ensuring data integrity, but minimising the risk of unauthorised changes or malicious attacks to the data and application container.

The clipboard functionality of Qubes-OS also requires modification to ensure data goes through the sanitation process when it synchronises to the data and application container. The clipboard should further interface with the identification and authorisation services in the management container to ensure data is authorised before it can by copies or moved.

The modification of the windows management in Qubes-OS and the custom interfaces and services mentioned in this section require significant modification and the sanitation problem require future research.

12.2.1.5 Secure keys in TPM

The Neo model provides confidentiality of the management container and other containers by encrypting the whole system on the disk. The cryptography keys used for the encryption and decryption of the management container are stored in the TPM. TPM is a technology available today, that provides a separate cryptographic processor that also has the ability to store security keys and security configuration information.

The Neo model requires that Neo devices use TPM with a bootloader to load the management container. A bootloader that interacts with TPM already exists in Linux and is called Linux Unified Key Setup (LUKS) (Fruhwirth, 2016).

The implementation challenge with the Neo model and the encrypted management container is not the decryption and mounting of the management container. The challenge is that today's mobile devices provide certain functionality from an unencrypted partition in the system. Limited functionality such as telephony for emergency numbers may be provided. Currently, the Neo model does not allow the mounting of an unencrypted partition. This means a Neo device provides no functionality, if not authorised by a user, during device initialisation.

12.2.1.6 Secure keys in Management container

The storing of private keys in order to decrypt other symmetric keys are common practice in operating systems. Modern operating systems have built-in libraries that expose the encryption and decryption algorithms. The discussion so far in this chapter alluded that most of the implementation of the Neo model may be achieved by using Qubes-OS. Qubes-OS is Linux based and exposes the normal Linux crypto library (Linux man page, nd), which enables the development of software to use cryptographic algorithms.

At the time of writing, Qubes-OS does not support multiple users. This means that Qubes-OS in its current version and form cannot be used to secure the security keys for different user containers. This is a major problem for implementing the Neo model and acts as a showstopper for implementation efforts.

Qubes-OS also does not encrypt each guest operating system, using its own key. The starting and stopping of operating system guests need to be modified so that the start process must first decrypt the corresponding operating system image and then starting it. The shutdown or stop process needs to ensure the guest operating system is encrypted after the operating system has shut down.

12.2.1.7 Provisioning process

The provisioning process is responsible for the dynamic creation and removal of containers. The provisioning process creates temporary containers and data and application containers. The provisioning process uses a secure container framework. The secure container framework is loosely defined in Chapter 8 section 8.5 as a specific directory structure, with a number of default services that start as soon as the operating system starts.

Qubes-OS uses the principle of a template to create new guest operating systems. The template either is pre-compiled or is a source-code version of a basic guest operating system. The Qubes-OS ships with a number of tools to update templates, create new operating system guests from templates or even removing operating system guests.

The only modification required is to ensure that during the provisioning process, a symmetric key for the guest operating system is created. The symmetric key must then be encrypted using the container owner's public key. Either this public key must be generated, together with the corresponding private key, or the company that is the owner of the container must provide the public key and keep the private key secure and private.

Qubes-OS allows the Neo model to be implemented with some modification of the existing provisioning tools. The provisioning process in the Qubes-OS does not support multiple users and requires major changes to support multiple users, as is required in the Neo model.

12.2.1.8 Container control data structures

The container-control-data-structure defines the rules that apply to a specific secure container. The data structure as described in Chapter 8 section 8.6 contains a representation of such a control. The control may be implemented as an advanced data structure, but can easily be stored in either a database or separate eXtensible Markup Language (XML) or JavaScript Object Notation (JSON) files.

The Neo model does not prescribe the way in which the data structures should be stored, but an example of a JSON representation of some of the elements in the data structure first described in Chapter 8 section 8.6 is shown in Figure 56.

The creation and storing of the data structures do not provide many implementation problems aside from ensuring that a standard is defined that describes the control. The standard should be adhered to throughout the implementation.

```
{
   "exclusiveUse":
   {
      "enabled":"TRUE",
      "dateAndTime":[
         {"day":"Monday","timeFrom":"06:00:00","timeTo":"15:00:00"},
         {"day":"Tuesday","timeFrom":"06:00:00","timeTo":"15:00:00"},
         {"day":"Wednesday","timeFrom":"06:00:00","timeTo":"15:00:00"},
         {"day":"Thursday","timeFrom":"06:00:00","timeTo":"15:00:00"},
         {"day":"Friday","timeFrom":"06:00:00","timeTo":"15:00:00"}
      ],
      "neoDeviceId":"",
      "corpUserId":"",
      "zoneId":""
   },
   "offlineUse":"FALSE",
   "locationActivation":
   {
      "zoneId":"",
      "gpsBoundary":"",
   },
   "deviceFeatures":
   {
      "camera":"FALSE",
      "microphone":"FALSE",
      "thermometer":"TRUE",
      "barometer":"TRUE",
      "gps":"TRUE"
   }
}
```

FIGURE 56: JSON REPRESENTATION OF CONTROL DATA STRUCTURE (BY AUTHOR)

12.2.1.9 Container control management interfaces

The container control defined in the Neo model has a certain scoped impact in the Neo device. The impact scope can be described with three levels of scope:

- **Device wide impact.** Settings such as "exclusive use" stops any other containers from being active while the container in focus, is currently running. This type of setting affects not just the container to which the setting applies, but also other containers.

- **Container wide impact**. Most of the other settings affect the container and the way in which the container can work. Examples of these settings are the "device features". When a device feature is disabled, the relevant virtual device is not enabled in the data and application container. There are also other examples of container wide settings, such as the location activation settings, which activates the container, when the Neo device is located inside a specific network zone or GPS boundary.
- **Application impact**. Other settings have to reach inside the container and modify the configuration of applications. Each application configuration setting requires special handling and the container control data structure should effectively describe the exact configuration file and value inside the file that needs to be managed.

A specialised management interface is required to read the container control data structures, but also apply the settings defined within. The interface should have the ability to apply the setting according to the specific scope. The interface also needs to allow owners from creating, modifying or deleting settings inside the container control data structures.

The container-control-management-interface needs to be specifically developed. No such service currently exists in either Qubes-OS or the native XEN hypervisor. The container control interface should ensure that any connections to the container-control-data-structures are authenticated and ensure settings are applied in real time.

12.2.2 Summary of implementation possibility.

The SCP ensures access to data and applications are strictly controlled. Data and applications are isolated based on ownership. The SCP is realised using a number of separate components. The implementation of these components using technology or systems available during the writing of this document was discussed in section 12.2.1.

The architecture in Qubes-OS supports many of the components that define the SCP. Some level of modification is required in order to implement the various components. Table 7 is a brief summary of the discussion of the various components in section 12.2.1. Table 7 highlights the specific component and then define an existing technology or system that may be used to implement the specific component. Table 7 further summarises the ability to use the technology or component in its native form or whether some level of modification is required.

TABLE 7: SCP IMPLEMENTATION SUMMARY

Component	Existing systems	Modification required
Secure container	Linux Containers Xen hypervisor Qubes-OS	**A significant amount** **A significant amount** **Some modification**
Management container	Xen hypervisor Qubes-OS	**Special mounting service to be developed** **Multi-user modification required**
Data and application container	Qubes-OS	**Modification required to be user-centric**
Temporary container	Qubes-OS	**Modification of user interface required** **Synchronisation service development** **Sanitation problem requires future research**
Secure keys in TPM	Qubes-OS with LUKS	**Modification required for initial start-up**
Secure keys in Management container	Qubes-OS	**Significant modification required for multi-user support**
Provisioning process	Qubes-OS	**Modification required in the standard provisioning process**
Container control data structures	XML or JSON	**All container control data structures need to be created**
Container control management interface	Qubes-OS	**Custom management interface needs to be developed**

The SCP is one of the two aspects that describes the essence of the Neo model. The MAP is the second property that applies to the Neo model. The next section discusses the implementation possibility of the MAP and the components that exist to support the MAP.

12.3 Implementing the mutual authentication property

The MAP includes many of the relevant aspects of the communication security. The various aspects and activities that establish the MAP are summarised in Table 6 in Chapter 8 section 9.7. This section uses Table 6 as a basis for discussing the implementation possibility of the various activities and components that enable the MAP.

This section is organised according to the security services described by the columns in Table 6. Each section describes the potential implementation of the security service using that technology available today or available in existing systems.

The columns in Table 6 and the sections describing their implementation possibility is structured as follow:

- **Section 12.3.1. Initial Identification and Authentication**. This section evaluates the identification and authentication that occurs when two Neo model components are newly introduced to each other.
- **Section 12.3.2. Initial and Subsequent Authorisation**. The initial authorisation steps and the subsequent authorisation steps are evaluated in the same section even though they are two separate columns in Table 6. The reason why they are grouped together is that they modify and use the same mechanisms.
- **Section 12.3.3. Subsequent Identification and Authentication**. The mechanisms that allow any reoccurring identification and authentication between previously authorised components are discussed in this section.

- **Section 12.3.4. Channel Confidentiality**. This section evaluates the possibility to ensure a confidential communication channel between two Neo device components.
- **Section 12.3.5. Summary of implementation possibility**. This section provides a summary of the results from the discussion in sections 12.3.1 to 12.3.4.

12.3.1 Initial identification and authentication

There are three aspects that are important during the initial identification and authentication of the various components in the Neo model. The three aspects describe how identification information is exchanged during the initial identification and authentication phase. The identity information can be between users, devices and containers. The primary goal of this phase is to exchange or create identity information with some type of secret that can be used for future authentication. The three aspects are:

- **Creation of user accounts by device owner or start-up process**. Most modern operating systems support the creation of user accounts. During initial installation, all modern operating systems, create an initial user account with a password. Even though the creation, storing and using of user accounts are well established in operating systems, Cubes-OS does not support multiple user accounts. Section 12.2 highlighted the fact that Cubes-OS can be used to help establish SCP but requires significant modification to allow multiple user accounts.
- **Device identities exchanged using an OoB mechanism**. The exchange of device identities requires some type of OoB mechanism. Most modern mobile devices support near-field communications (NFC). NFC lends itself to establish identities between wireless components. The exchanging of identity information is established with a lot of research describing many implementation aspects (Sarvabhatla et al., 2015) (Ji & Xia, 2016) (Klauer et al., 2017). The Cubes-OS support NFC through the NFC subsystem available for Linux. A module to handle the exchange of identity of information needs to be specifically written to support NFC. NFC can be used to exchange identity information between devices to create a category of trusted devices, with their secrets.
- **Container identities created by the administrator**. When a company defines a corporate container, the system interface must automatically assign a unique identifier to the corporate container. The unique corporate container can then be added to a group identity, which may be used during the authorisation processes. The unique identity and adding the group identity can be managed by the mobile device management system, that oversees the creation of corporate containers. The whole mobile device management system needs to be developed as this does not fit in with existing mobile device management systems.

The identification and authentication of devices and users in the Neo model utilise existing concepts found in computing. The unique aspects of the Neo model, however, requires that I/O peripherals be allowed to store identity information to ensure mutual authentication happens. No mobile device management currently exists to manage the creation of identities for secure containers that can later be used for authorisation purposes. The next section discusses the implementation possibilities to enable initial and subsequent authorisation between components.

12.3.2 Initial and subsequent authorisation

A lot of the initial authorisation between the various components in the Neo model allow authorisation through the act of creating the user identity of the component. **Example**: When a device owner creates a new user on the Neo device, the device owner implies that the user may have access to the Neo device. During the user-creation process, the Neo device owner should select the existing I/O peripherals that the new user can use to interact with the Neo device. The Neo device owner can then also allow the newly created user account access to the secure containers where the Neo device owner is also the container owner.

All of the above means that initial authorisation updates access rules either automatically through a pre-defined process or manually by the device owner or business administrator. For all of this, the concept of an identification and authorisation service manages the access through objects called access rules.

The access rules are not access-control rules that exist on file system level. Instead, the identification and authorisation service verifies object access through access rules, which may be defined in a database that exists in the management container. The implementation of this may be through the identification and authorisation service that runs in the management container and may use a very lightweight SQL-based database to query access levels.

The challenge with identification and authorisation service is that if an existing system, like Cubes-OS, is used, the existing mechanisms that implement actions on subjects in the existing operating system needs to be modified. Instead of a major modification of an existing system's authorisation mechanisms, the interfaces to the existing authorisation mechanisms may be modified to interface with the identification and authorisation service. This means the identification and authorisation service provides an abstracted layer to some of the existing mechanisms. The introduction of the identification and authorisation service would require modification of most existing authorisation mechanisms.

The authorisation of users, devices and containers are managed using the identification and authorisation services. The access rules used by the identification and authorisation service may be stored in a local protected database. Existing authorisation mechanisms need to be modified to interface with only the identification and authorisation service. All of this needs to be custom developed to enable this aspect of the Neo model.

12.3.3 Subsequent identification and authentication

The initial identification and authentication concern itself with establishing and exchanging identity information between different security subjects. Subsequent identification and authentication occurs after the creation of identity information and ensures subjects are identified and authenticated during communication.

In the Bluetooth protocol stack, devices that exchanged identity information and created secrets are known as trusted devices. During the authentication process of the Bluetooth protocol, a verifying device generates a random input string, known as a verifier, which is sent to the claimant. A claimant is a device that claims to be trusted and wants to connect to a verifying device. The claimant uses the verifier together with the claimant's Bluetooth address and encrypts it with the shared symmetric key to produce a result. The result is sent to the verifying device. The verifying device also generates a result using the verifier and the claimants Bluetooth address and encrypts them both with the shared symmetric key. If the result generated and received by the claimant match, then the claimant is authenticated (Bluetooth SIG, Inc, 2017) (Padgette, 2017).

In the Neo model, the Bluetooth protocol stack can be used to identify and authenticate I/O peripherals and Neo devices without any or very little modification. The Linux Bluetooth stack can be enabled in Qubes-OS Dom0 to support Bluetooth communication between Neo device and I/O peripheral.

In the case where the Neo device has to authenticate with a Gateway Controller, the shared secret generated during the initial identification process can be used to establish an application layer authentication protocol. The easiest way to implement this is to use existing technologies like OpenVPN (OpenVPN Inc., 2018) to establish an encrypted session with the Gateway Controller.

Qubes-OS already supports OpenVPN. OpenVPN also supports authentication methods that include either a username or password combination or key pairs with certificates or with a shared secret key. The shared secret key method can be used to configure a connection between the data and application container and the Gateway Controller. Some modification of the process may be required to ensure the shared secret key does not stay static but changes over time.

The biggest drawback of using Cubes-OS as the basis for implementing the Neo model is the authentication that is required by the Neo device and different users. As already mentioned, Cubes-OS does not support multi-users, even though it is a built-in feature of most modern operating systems, like Linux.

Once a subject has been identified and authenticated, the next step is to ensure the channel between the subject and the object is confidential. The next section describes how channel confidentiality can be implemented in the Neo model.

12.3.4 Channel confidentiality

The implementation of channel confidentiality is different for the communication between Neo device and I/O peripherals and Neo device and Gateway Controllers. The Bluetooth communication protocol is ideally suited for communication between I/O peripherals and Neo devices, while a higher-level encryption protocol should be used to ensure secure communication between the Neo device and Gateway Controller.

Bluetooth is ideally suited for short range, low powered devices. Bluetooth automatically encrypts data between devices. OpenVPN can be used to ensure the channel between a secure container and the Gateway Controller is encrypted. As mentioned in section 12.3.3, OpenVPN can make use of a shared secret that exists between the Neo device and Gateway Controller.

Most of the configuration and security that is established while identifying and authenticating the various components of the Neo model, ensures that components can communicate with each other over encrypted channels. The next section extracts some of the pressing aspects mentioned in this chapter and describes areas that can add value to the implementation of the Neo model if they were researched further.

12.3.5 Summary of implementation possibility

Sections 12.3.1 to 12.3.4 described various aspects defining the MAP. To implement the MAP, modification or development of new services or mechanisms are required

Table 8 provides a summary of the implementation possibility of the various components. The first column describes the effort required to implement the specific technology or mechanism to ensure the security service is implemented in a specific communication scenario.

Table 8 also groups the categories of implementation efforts together. The grouping aspect allows the reader to view the effort and the number of components it addresses. Not only does the table allow the reader to get a quick glimpse of the various implementation categories, it can also be used as a guide for further research.

Using the information in Table 8 it can be seen that "Significant work to design and create the Identification and Authorisation service" is the aspect that will solve most of the issues to implement the Neo model. The Identification and Authorisation service plays a significant role in ensuring components are identified, authorised and communication is confidential. Many of the security functionality of the MAP is included in the Identification and Authorisation service.

TABLE 8: SUMMARY OF MAP IMPLEMENTATION POSSIBILITY

Implementation effort	Communication Scenario	Security scenario	Technology or mechanism
Leverage the confidentiality provided by Bluetooth	**I/O Peripheral to Neo Device**	Channel Confidentiality	Session-based encryption
Little work to leverage the identification and authentication built into Bluetooth	**I/O Peripheral to Neo Device**	Subsequent Identification and Authentication	Shared secret
Modify Cubes-OS to include OpenVPN	**Neo Device to Gateway Controller**	Subsequent Identification and Authentication	Shared secret
Modify Cubes-OS to include OpenVPN using a shared secret	**Container to Gateway Controller**	Initial Authorisation	Shared secret
Modify session setup using OpenVPN to modify shared secrets	**Container to Gateway Controller**	Channel Confidentiality	Session-based encryption
	Neo Device to Gateway Controller	Channel Confidentiality	Session-based encryption
Dependent on identification and authentication mechanism	**User to Container**	Channel Confidentiality	N/A
	User to Neo Device	Channel Confidentiality	N/A
Significant development work to implement and modify a mobile device management system	**Container to Gateway Controller**	Initial Identification and Authentication	Created by Administrator
Significant modification of Cubes-OS	**User to Container**	Subsequent Identification and Authentication	User account and authentication mechanism Cached Credential
	User to Neo Device	Initial Identification and Authentication	User account either created during initial start-up or created by Neo device owner.
		Subsequent Identification and Authentication	User account and authentication mechanism
A small implementation of NFC in Cubes-OS	**Neo Device to Neo Device**	Initial Identification and Authentication	Out-of-band (NFC)
Significant modification of Cubes-OS with federated identity management aspects	**User to Container**	Initial Identification and Authentication	User account either created during initial start-up or created by Neo device owner or linked to business identity
Significant work to design and create the Identification and Authorisation service	**Container to Gateway Controller**	Subsequent Authorisation	Access Rules (Identification and Authorisation service)
		Subsequent Identification and Authentication	Access Rules (Identification and Authorisation service)
	I/O Peripheral to Neo Device	Initial Authorisation	Established through initial identification and authentication
		Subsequent Authorisation	Access Rules (Identification and Authorisation service)
	Neo Device to Gateway Controller	Initial Authorisation	Established through initial identification and authentication
		Subsequent Authorisation	Access Rules (Gateway Controller, Zones)
	Neo Device to Neo Device	Initial Authorisation	Established through initial identification and authentication
		Subsequent Authorisation	Access Rules (Identification and Authorisation service)
	User to Container	Initial Authorisation	Access Rules (Identification and Authorisation service)
		Subsequent Authorisation	Access Rules (Identification and Authorisation service)
	User to Neo Device	Initial Authorisation	Access Rules (Identification and Authorisation service)

Implementation effort	Communication Scenario	Security scenario	Technology or mechanism
		Subsequent Authorisation	Access Rules (Identification and Authorisation service)
Significant modification of Cubes-OS with federated identity management aspects	**User to Container**	Initial Identification and Authentication	User account either created during initial start-up or created by Neo device owner or linked to business identity
A small implementation of NFC in Cubes-OS. Create an identity exchange process.	**I/O Peripheral to Neo Device**	Initial Identification and Authentication	Out-of-band (NFC)
	Neo Device to Gateway Controller	Initial Identification and Authentication	Out-of-band (NFC or docking station)
Use OpenVPN to help establish connectivity over IP network connectivity	**Neo Device to Neo Device**	Channel Confidentiality	Session-based encryption
		Subsequent Identification and Authentication	Shared secret

This section described the various processes and components that enable the MAP of the Neo model. The focus of section 12.3 describes the implementation possibilities of the various aspects of the MAP.

12.4 Conclusion

The Neo model is a model that describes a mobile operating environment that ensures data and applications are secured and isolated based on ownership. The Neo model further describes the use of various I/O peripherals that can be used with a hypothetical Neo device. Chapter 12 highlighted the effort required to implement the Neo model.

The Neo model consists of two important properties. The two properties are the SCP and MAP. The implementation effort of the various components of the SCP is described in section 12.2. Section 12.3 describes the implementation effort of the MAP.

Cubes-OS is a system that can assist to implement both the SCP and MAP. Cubes-OS provides an ideal system that already isolates and protects different operating environments. Cubes-OS also provides a management domain, called Dom0, that can be used to implement the Management container of the Neo model.

Cubes-OS already provides a container mechanism to quickly create containers, that are needed to provision new containers for users or for companies. The provisioning of containers also allows temporary containers to be created. Cubes-OS needs modification to properly define and link temporary containers to data and application containers. The synchronisation and sanitation of data between temporary containers and their linked data and application containers require exciting future research aspects.

Significant modification of Cubes-OS is required to ensure the support for multiple user accounts. The Identification and Authorisation service, that manages many of the security services of the MAP requires future research to assist with the implementation of an existing system such as Cubes-OS.

This chapter highlighted that an implementation of the Neo model requires a significant amount of work and future research. Even though a lot of work is required and some aspects need future research, the Neo model is not just a model that can only be used in research, instead, the model describes many aspects that can add value to existing modern mobile operating systems.

Introduction

- Chapter 1 Introduction

The end game

- Chapter 2 Model Overview

Literature study

- Chapter 3 Isolation properties in mobile operating systems (SCP)
- Chapter 4 Isolation techniques in other technologies (SCP)
- Chapter 5 An overview of peripheral security (MAP)

Bridge

- Chapter 6 The drivers for a new model

Building the Neo model

- Chapter 7 Overview of the Neo Model
- Chapter 8 The Secure Container Property
- Chapter 9 The Mutual Authentication Property
- Chapter 10 Container life cycle
- Chapter 11 Novel Implementations

Evaluation and summary

- Chapter 12 Potential implementation of Neo Model
- Chapter 13 Conclusion and future work

Chapter 13 CONCLUSION AND FUTURE WORK

13.1 Introduction

The previous chapter evaluated the potential implementation possibility of the Neo model. The aim of this chapter is to evaluate the Neo model against the stated architectural requirements summarised in Chapter 6 section 6.4. The chapter also evaluates the objectives and hypothesis defined in Chapter 1. The research conducted during the development of this thesis is evaluated against the contribution towards the body of knowledge. The last aim of this chapter is to evaluate areas of future research flowing from the work conducted in the thesis.

This chapter is organised as follow:

- Section 13.2 evaluates each requirement identified in Chapter 6 and see whether it has been addressed during the development of the Neo model in chapters 7 through 11.
- Section 13.3 evaluate the objectives of the research and tests the hypothesis of the thesis.
- Section 13.4 identifies areas in the thesis that specifically contribute to the body of knowledge.
- Section 13.5 highlights areas of future research flowing from the work done while constructing the Neo model.

The evaluation of the Neo model starts by conducting a decisive evaluation against the specific architectural requirements that helped shape the Neo model.

13.2 Model requirement evaluation

During the literature review in Chapters 2 to 5, a number of drivers were identified. The drivers highlighted important aspects and problems in existing mobile operating systems, existing isolation mechanisms and security aspects of I/O peripherals. The drivers acted as input that led to the formulation of a number of architectural requirements for the Neo model.

During the development and formulation of the Neo model, the Neo model incorporated and addressed the requirements. Table 9 highlights the various secure container property requirements and cross-reference the location where the requirements were addressed. Table 10 lists the various requirements for the mutual authentication property.

Both tables highlight the various requirements. For each requirement, a corresponding entry exists in either the Chapter 7, Chapter 8, Chapter 9 or Chapter 10 columns. The corresponding entry highlights the section number inside the chapter, as well as the page number where the requirement is addressed.

Example: Table 9 shows requirement SCP1.1, which highlights the design methodology of the Neo model to adopt a hypervisor level of isolation. SCP1.1 is addressed in Chapter 8, section 8.2 page 114.

For each requirement listed in Table 9 and Table 10 a corresponding entry exists, that addresses the requirement. This leads to the conclusion that the Neo model has addressed each architectural requirement originally defined in Chapter 6 section 6.4.

The next section evaluates the original objectives of the thesis as well as the hypothesis.

Please note that Table 9 and Table 10 is located on the next two pages. This chapter resumes with section 13.3 on page 205.

TABLE 9: SCP REQUIREMENT TRACKING

Secure Container Property - Requirements		Chapter 7	Chapter 8	Chapter 9	Chapter 10
SCP1. Design Methodology					
SCP1.1	Hypervisor level isolation		8.2 (p114)		
SCP2. Application communication					
SCP2.1	Applications inside containers communicate with each other using modern operating system objects.		8.3.1 (p116)		
SCP2.2	Applications in different containers cannot communicate with each other.		8.2 (p113)		
SCP2.3	Communication between containers is only allowed through strongly controlled mechanisms.		8.5 (p126); 8.6 (p127)		
SCP3. Privacy protection					
SCP3.1	Owner based isolation of data and applications in containers.		8.2 (p114)		
SCP3.2	Configuration and access control using policy enforcement.	7.5 (p107)	8.6 (p126)		10.3.3.4 (p162)
SCP4. Protected management environment					
SCP4.1	Unauthorized changes to the management environment are not allowed.		8.3.2 (p118)		
SCP4.2	Updating of the management environment should guarantee integrity during update and after the update.		8.3.3.1 (p118)		10.3.1.1 (p157)
SCP4.3	The integrity of the management environment should be confirmed by the hardware		8.3.2 (p117)		10.3.1.1 (p157)
SCP4.4	Network connections to the management environment are not allowed.		8.3.3.1 (p118)		
SCP5. File system deployment					
SCP5.1	The use of templates provides baseline container configuration.		8.5 (p125)		10.3.2.3 (p159)
SCP5.2	Access to removable storage is strongly controlled.		8.3.3.2.2 (p120)		
SCP5.3	Deployed containers are encrypted on the hard disk		8.4.1 (p122)		10.3.3.3 (p161)
SCP6. Network access security					
SCP6.1	No direct access from inside any application and data containers to the network, instead of indirect mechanisms should be used.	7.6 (p108)	8.3.3.2.1 (p119)		

Secure Container Property - Requirements		Chapter 7	Chapter 8	Chapter 9	Chapter 10
SCP6.2	Data and application containers can only access network traffic directed to and from it.		8.3.3.2.1 (p119)		

TABLE 10: MAP REQUIREMENT TRACKING

Mutual Authentication Property - Requirements		Chapter 7	Chapter 8	Chapter 9	Chapter 10
MAP1: Identification and authentication					
MAP1.1	Mutual authentication is required for Neo device and I/O peripheral	7.4 (p105)		9.4.2.2 (p140)	10.3.1.2 (p158)
MAP1.2	OoB mechanisms must be used during initial identification and authentication.	7.3.2 (p104)		9.4.2.1 (p139)	10.3.1.2 (p157)
MAP2: Authorisation					
MAP2.1	Access control triplets must be used to determine access. The access control triplet consists of a user, I/O peripheral and container identifier.			9.5.1 (p142)	10.3.3.4 (p162)
MAP2.2	Access control is managed by the management system of the Neo model.	7.3.2 (p103)		9.5.2 (p146)	
MAP2.3	The owners of containers set access control.			9.5.1 (p141)	10.3.2.4 (p159)
MAP3: Confidentiality					
MAP3.1	Only encrypted communication is allowed between Neo device and I/O peripheral.	7.3.2 (p104)		9.6.1 (p147)	10.3.1.2 (p158)
MAP3.2	The management of encryption keys should occur in the management environment and not from within containers where users have access.		8.3.1 (p117)	9.6.1 (p146)	10.3.2.2 (p159)

13.3 Research evaluation

When evaluating the research conducted during the development of the thesis, is it important to take cognisance of the original problem statement, together with the objectives defined for the research. The problem statement defines the problem that needs to be solved by the research.

This section reflects on the problem statement, hypothesis and objectives defined in Chapter 1.

The problem defined in Chapter 1 section 1.2 states that users that access company data may make use of personal devices. The utilisation of personal devices creates a requirement for users to share information between the personal device and company infrastructure. The problem is that the device owner, which has full control over the device and the data on the device, does not own the data.

The hypothesis is stated as:

It is hypothesised that it is possible to design a secure mobile operating environment that facilitates the use of a mobile device for both personal and business purposes, using hardware that is not readily available today but is possible in the future. The mobile device runs a purposely designed mobile operating system that ensures different form factor input/output (I/O) devices can connect to the special mobile device. The mobile operating environment ensures that the information on the device is classified and isolated. Access and secure communication are established on an application and data layer, instead of just the operating system level. Communication between the mobile device and I/O devices are mutually authenticated.

In line with the problem statement and hypothesis, a primary and secondary objective was defined. The primary objective of the research is defined as:

Primary Objective: Develop a model that is called the Neo model that describes a secure mobile operating environment that can be used for both personal and business purposes. The Neo model must ensure that personal and business data and applications are isolated from each other. The Neo model must further ensure that data owners have full control over their data and applications.

The secondary objective of the research is defined as:

Secondary objective: Expand the Neo model to allow multiple I/O peripherals to be used securely within the mobile operating environment.

The question to be asked is whether the problem statement has been addressed by completing the primary and secondary objectives.

The primary objective broadly focusses on the Secure Container Property (SCP). The drivers for the SCP was identified by evaluating isolation mechanisms used by existing mobile operating systems and other operating system mechanisms, discussed in Chapter 3 and Chapter 4. The drivers were further refined into a set of SCP requirements. These requirements were used to help form the Neo model. Chapters 7, 9 and 10 specifically contributed to establishing the SCP of the Neo model. Section 13.2 proved that all the SCP requirements were addressed in the Neo model. This leads the author to establish that the primary objective of the research has been reached.

The secondary objective broadly focusses on the Mutual Authentication Property (MAP). As with the SCP, the drivers for the MAP were identified in the literature review, specifically in Chapter 5. The drivers for the MAP were refined into the MAP requirements. The MAP requirements were addressed in Chapters 7, 8, 9 and 10. Section 13.2 proved that all the requirements were addressed. The author can now state that the secondary objective of the research has been reached.

The problem is directly addressed by reaching both the primary and secondary objective. The hypothesis is also addressed by the primary and secondary objectives. This leads to the conclusion that research objectives, the hypothesis and the problem statement has all been addressed in this research.

The next section weighs the knowledge contributed during the research conducted of this thesis.

13.4 Research contribution

This section evaluates the research in terms of knowledge contribution. It specifically tries to answer the question of whether the research conducted demonstrates new or improved concepts (Davis, 2005).

The research conducted in the development of the thesis produced an artefact called a model. Models should provide clarity, differentiation or generalisation (Olivier, 2009, pp.46-47). The statements below, argue that the Neo model provides both clarity and differentiation in certain aspects of knowledge.

The following statements argue that new and improved concepts have been achieved in the research and development of the Neo model and this thesis:

1. The research contributes knowledge by summarising the existing methods and mechanisms used for isolation. This highlighted gaps in existing mobile operating systems using application-level isolation. It further highlighted problems with owner-based control of data in existing mobile operating system environments.
2. The research contributes knowledge by summarising security aspects in I/O peripherals. This highlighted gaps in authorisation in wired I/O peripherals.

3. The Neo model provides a comprehensive model that describes isolation based on ownership and other criteria inside a mobile operating environment. It is the author's belief that this type of principle is established and affirmed in the Neo model and is ***different*** from other mobile operating environment models.
4. It further describes mutual authentication, confidentiality and authorisation of I/O peripherals against not only devices but also isolated secure containers. The Neo model ***clarifies*** this type of principle clearly in the MAP.
5. The research process has also yielded a number of academically accepted papers at conferences and journals, as stated in Chapter 1 section 1.9. This provides further evidence that the knowledge has been peer reviewed and has been generally accepted.

Thus: The research conducted during the development of the Neo model contributes to existing knowledge by clarity and differentiation.

13.5 Future research

The research conducted during the development of the Neo model opened up a number of interesting areas for future research. Some of these areas may provide even more clarification, but some of the principles may be explored in other areas of cyber security.

The first area that was highlighted in Chapter 12 section 12.3.5 was the development of a comprehensive Identification and Authorisation service. What makes the Identification and Authorisation service unique is that it requires special attention when managing access between different operating system objects and subjects. Some of the objects and subjects are not the typical file system objects, but also include the various secure containers and I/O peripherals. Further research that can provide an efficient mechanism to control access between these objects and subjects will allow the Neo model to be even more widely accepted as a potential for a mobile operating environment.

The second area is more a concept that flows from the research done while developing the Neo model. The concept is the containerisation of data and applications based on ownership. The concept of owner-based containerisation may be applied in other areas of cyber security. One such area is the decentralisation of data in web applications. Decentralisation of data in web applications allows users to stay in full control of their own personal information stored in decentralised personal online data stores (POD). The web application utilises metadata on the web server to link to the personal information stored on the PODs (Mansour et al., 2016) (du Toit, 2018).

The principle of owner based data isolation is a research field that the author is currently exploring with interesting solutions in the fields of web applications and internet of things (IoT).

13.6 Conclusion

This chapter focussed on providing a conclusion to the research done in developing the Neo model. The chapter evaluated the requirements that influenced the design of the Neo model. Each requirement was cross-referenced against the relevant location in the thesis that addressed the specific requirement. All SCP and MAP requirements were addressed and considered in the design of the Neo model.

Once all the design requirements were evaluated, the original problem statement, hypothesis and objectives of the research were reflected on. In Chapter 1 the hypothesis was formalised from the original problem statement. From the hypothesis, a primary and secondary objective was defined. This chapter argued that both the primary and secondary objectives were fulfilled.

After the primary and secondary objectives were evaluated a statement on the contribution of the research of this thesis was made. The Neo model contributes to both clarity and differentiation. The Neo model provides clarity on I/O defined access control into a secure container level of the Neo model. The Neo model also provides differentiation in owner based data and application isolation.

By establishing the research contribution the aspect of future research was addressed. The identification and authorisation service provides an interesting research field. The identification and authorisation service is one of the single most important aspects that would make the Neo model more attractive for implementation. Further research in establishing a fast and efficient, but still secure, identification and authorisation service is needed. The aspect of the containerised application and data isolation can also be adopted in other areas of cyber security research.

The Neo model has shown that it provides a secure mobile operating environment. It describes the isolation of application and data based on ownership. It also describes how I/O peripherals can be controlled and provided access to the isolated computing environments. The controlling of access to the secure containers provide assurance to data owners that data and applications are secure and controlled.

The Neo model allows users with mobile devices to use their mobile devices for both personal and business purposes. The Neo model minimises the dependency on potential insecure cloud storage. The Neo model further allows companies to stay in control of their data and applications even when used on devices that they do not own.

References

Ahmad, M.S., Musa, N.E., Nadarajah, R., Hassan, R. & Othman, N.E., 2013. Comparison between android and iOS Operating System in terms of security. In *Information Technology in Asia (CITA), 2013 8th International Conference on*., 2013. IEEE.

Aiken, M., Fähndrich, M., Hawblitzel, C., Hunt, G. & Larus, J., 2006. Deconstructing Process Isolation. In *Proceedings of the 2006 Workshop on Memory System Performance and Correctness*. San Jose, California, 2006. ACM.

Ali, M.E., Anwar, A., Ahmed, I., Hashem, T., Kulik, L. & Tanin, E., 2014. Protecting Mobile Users from Visual Privacy Attacks. In *Proceedings of the 2014 ACM International Joint Conference on Pervasive and Ubiquitous Computing: Adjunct Publication*. Seattle, Washington, 2014. ACM.

Android, 2016. *Android*. [Online] Available at: https://www.android.com [Accessed 4 July 2016].

Android, n.d. *Security Tips. Android Developers*. [Online] Available at: http://developer.android.com/training/articles/security-tips.html [Accessed 5 February 2016].

Andrus, J., Dall, C., Hof, A.V.T., Laadan, O. & Nieh, J., 2011. Cells: a virtual mobile smartphone architecture. In *Proceedings of the Twenty-Third ACM Symposium on Operating Systems Principles*., 2011. ACM.

Andrus, J. & Nieh, J., 2012. Teaching Operating Systems Using Android. Raleigh, 2012. ACM.

Apple Inc., 2014. *iCloud*. [Online] Available at: https://www.icloud.com [Accessed 11 September 2014].

Apple Inc., 2015. *iOS Security Guide*. [Online] Available at: https://www.apple.com/business/docs/iOS_Security_Guide.pdf [Accessed 5 February 2016].

Apple Inc., 2016. *App Extensions*. [Online] Available at: https://developer.apple.com/app-extensions/ [Accessed 24 June 2016].

Apple Inc., 2016. *iOS and the new IT*. [Online] Available at: http://www.apple.com/iphone/business/it/ [Accessed 29 June 2016].

Apple Inc., 2016. *OS X*. [Online] Available at: http://www.apple.com/za/osx/ [Accessed 10 February 2016].

ARM Ltd., 2016. *ARM. The architecture for the digital world.* [Online] Available at: http://www.arm.com [Accessed 18 July 2016].

Arthur, W. & Challener, D., 2015. *A Practical Guide to TPM 2.0: Using the New Trusted Platform Module in the New Age of Security*. 1st ed. Apress Media, LLC.

Bastille Networks Internet Security, 2016. *KeySniffer*. [Online] Available at: https://www.keysniffer.net [Accessed 15 March 2018].

Beuchelt, G., 2013. Unix and Linux Security. In J.R. Vacca, ed. *Computer and Information Security*. Bedford: Elsevier. pp.165-81.

Bluetooth SIG, Inc., 2017. *How it works. Bluetooth Technology Website*. [Online] Available at: https://www.bluetooth.com/what-is-bluetooth-technology/how-it-works [Accessed 28 January 2017].

Bluetooth SIG, Inc, 2017. *Bluetooth*. [Online] Available at: https://www.bluetooth.com [Accessed 28 January 2017].

Bolosky, W.J., Douceur, J.R., Ely, D. & Theimer, M., 2000. Feasibility of a Serverless Distributed File System Deployed on an Existing Set of Desktop PCs. *SIGMETRICS Perform. Eval. Rev.*, 28(1), pp.34-43.

Bui, T., 2015. *arXiv.org*. [Document] (1) Available at: http://arxiv.org/abs/1501.02967 [Accessed 28 April 2018].

Calarco, G. & Casoni, M., 2013. On the effectiveness of Linux containers for network virtualization. *Simulation Modelling Practice and Theory*, 31, pp.169-85.

Canonical Ltd., n.d. *LinuxContainers.org Infrastructure for container projects.* [Online] Available at: https://linuxcontainers.org [Accessed 2 August 2016].

Cellrox Ltd., 2015. *cellrox*. [Online] Available at: http://www.cellrox.com [Accessed 9 February 2018].

Charles, A., 2014. *Naked celebrity hack: security experts focus on iCloud backup theory | Technology | The Guardian*. [Online] Available at: http://www.theguardian.com/technology/2014/sep/01/naked-celebrity-hack-icloud-backup-jennifer-lawrence [Accessed 10 September 2014].

Combe, T., Martin, A. & Di Pietro, R., 2016. To Docker or Not to Docker: A Security Perspective. *IEEE Cloud Computing*, 3(5), pp.54-62.

Creasy, R.J., 1981. The origin of the VM/370 time-sharing system. *IBM Journal of Research and Development*, 25(5), pp.483-90.

Dall, C., Andrus, J., Van't Hof, A., Laadan, O. & Nieh, J., 2012. The design, implementation, and evaluation of cells: A virtual smartphone architecture. *ACM Transactions on Computer Systems (TOCS)*, 30(3), p.9.

Davi, L., Dmitrienko, A., Sadeghi, A.-R. & Winandy, M., 2011. Privilege Escalation Attacks on Android. *Information Security: 13th International Conference, ISC 2010, Boca Raton, FL, USA, October 25-28, 2010, Revised Selected Papers*, pp.346-60.

Davis, G., 2005. Advising and Supervising. In D. Avison & J. Preis-Heje, eds. *Research in Information Systems: A Handbook for Research Supervisors and Their Students*. 1st ed. Amsterdam: Elsevier Butterworth Heinemann.

Docker Inc., 2018. *Title*. [Online] Available at: https://www.docker.com [Accessed 5 September 2018].

DotC United, 2016. *Flash Keyboard Emojis & More*. [Online] Available at: https://play.google.com/store/apps/details?id=com.dotc.ime.latin.flash [Accessed 18 July 2016].

Drake, J.J., Lanier, Z., Mulliner, C., Fora, P.O., Ridley, S.A. & Wicherski, G., 2014. *Android Hacker's Handbook*. 1st ed. John Wiley & Sons.

DropBox Inc., 2014. *DropBox*. [Online] Available at: https://www.dropbox.com [Accessed 11 September 2014].

du Toit, J., 2018. Digital Rights Management to Protect Private Data on the Internet. In Jøsang, A., ed. *17th European Conference on Cyber Warfare and Security ECCWS 2018*. Oslo, Norway, 2018. Academic Conferences and Publishing International Limited.

du Toit, J. & Ellefsen, I., 2015. A Model for Secure Mobile Computing. In *Science and Information Conference (SAI)*. London, 2015. IEEE.

du Toit, J. & Ellefsen, I., 2015. Location Aware Mobile Device Management. In *2015 Information Security for South Africa*. Rosebank, 2015. IEEE.

du Toit, J. & Ellefsen, I., 2017. Secure Peripherals in a Converged Mobile Environment. In Tryfonas, T., ed. *Human Aspects of Information Security, Privacy and Trust*. Vancouver, Canada, 2017. Springer International Publishing.

du Toit, J., Ellefsen, I. & von Solms, S., 2016. Bring your own disaster recovery (BYODR). In *2016 IST-Africa Week Conference*. Durban, 2016. IEEE.

Eddy, N., 2013. *Small Businesses Lack Adequate Data Protection Plans*. [Online] Available at: http://www.eweek.com/small-business/small-businesses-lack-adequate-data-protection-plans [Accessed 22 October 2015].

Eiband, M., Khamis, M., von Zezschwitz, E., Hussmann, H. & Alt, F., 2017. Understanding Shoulder Surfing in the Wild: Stories from Users and Observers. In *Proceedings of the 2017 CHI Conference on Human Factors in Computing Systems*. Denver, Colorado, USA, 2017. ACM.

Elenkov, N., 2015. *Android Security Internals: An in depth guide to Android's security architecture*. 1st ed. San Francisco, CA: No Starch Press.

England, P. & Manferdelli, J., 2006. Virtual machines for enterprise desktop security. *Information Security Technical Report*, 11(4), pp.193-202.

Extron Electronics, 2018. *Understanding EDID - Extended Display Identification Data*. [Online] Available at: 2018 [Accessed 15 February 2018].

Ferdowsi, A., 2011. *The DropBox Blog: Yesterday's Authentication Bug*. [Online] Available at: https://blog.dropbox.com/2011/06/yesterdays-authentication-bug/ [Accessed 10 September 2014].

FreeBSD, n.d. *Chapter 14. Jails*. [Online] Available at: https://www.freebsd.org/doc/handbook/jails.html [Accessed 2 August 2016].

Fruhwirth, C., 2016. *on-disk-format.pdf*. [Online] Available at: https://www.kernel.org/pub/linux/utils/cryptsetup/LUKS_docs/on-disk-format.pdf [Accessed 2 January 2018].

Gartner, 2018. *Bring Your Own Device (BYOD)*. [Online] Available at: https://www.gartner.com/it-glossary/bring-your-own-device-byod [Accessed 15 March 2018].

Gellman, B. & Poitras, L., 2013. *U.S., British intelligence mining data from nine U.S. Internet companies in broad secret program*. [Online] Available at: http://www.washingtonpost.com/investigations/us-intelligence-mining-data-from-nine-us-internet-companies-in-broad-secret-program/2013/06/06/3a0c0da8-cebf-11e2-8845-d970ccb04497_story.html# [Accessed 11 September 2014].

Giroire, F., Monteiro, J. & Perennes, S., 2010. Peer-to-Peer Storage Systems: A Practical Guideline to be Lazy. In *2010 IEEE Global Telecommunications Conference GLOBECOM 2010*., 2010. IEEE.

Google Inc., 2014. *Google Drive*. [Online] Available at: https://drive.google.com [Accessed 11 September 2014].

Google Inc., 2016. *Android Security. 2015 Year In Review*. [Online] Available at: http://static.googleusercontent.com/media/source.android.com/en//security/reports/Google_Android_Security_2015_Report_Final.pdf [Accessed 18 July 2016].

Google Inc., 2016. *BroadcastReceiver*. [Online] Available at: https://developer.android.com/reference/android/content/BroadcastReceiver.html#ProcessLifecycle [Accessed 12 July 2016].

Google Inc., 2016. *Google for work*. [Online] Available at: https://www.google.com/work/android/ [Accessed 14 July 2016].

Google Inc., 2016. *Google git*. [Online] Available at: https://android.googlesource.com [Accessed 6 July 2016].

Google Inc., 2016. *Google Maps URL Scheme*. [Online] Available at: https://developers.google.com/maps/documentation/ios-sdk/urlscheme [Accessed 24 June 2016].

Google Inc., 2016. *Remove corporate data from mobile device*. [Online] Available at: https://support.google.com/a/answer/173390?hl=en [Accessed 14 July 2016].

Google Inc., 2016. *Saving File*. [Online] Available at: https://developer.android.com/training/basics/data-storage/files.html [Accessed 13 July 2016].

Google Inc., 2016. *Set up Managed Profiles*. [Online] Available at: https://developer.android.com/work/managed-profiles.html [Accessed 14 July 2016].

Google Inc., 2016. *Storage Access Framework*. [Online] Available at: https://developer.android.com/guide/topics/providers/document-provider.html [Accessed 13 July 2016].

Google Inc, 2016. *Context*. [Online] Available at: https://developer.android.com/reference/android/content/Context.html [Accessed 13 July 2016].

Gregor, S. & Hevner, A.R., 2013. Positioning and Presenting Design Science Research for Maximum Impact. *MIS Quaterly*, 37(2), pp.337-56. Available at: https://doi.org/10.25300/MISQ/2013/37.2.01.

Haataja, K., Hyppönen, K., Pasanen, & Toivanen, P., 2013. *Bluetooth Security Attacks: Comparative Analysis, Attacks, and Countermeasures*. Berlin, Heidelberg. Available at: http://dx.doi.org/10.1007/978-3-642-40646-1_2.

Harbach, M., Von Zezschwitz, E., Fichtner, F., De Luca, A. & Smith, M., 2014. It's a hard lock life: A field study of smartphone (un) locking behavior and risk perception. In *Symposium on usable privacy and security (SOUPS)*. Menlo Park, CA, 2014. USENIX Association.

Helmke, M., Hudson, A. & Hudson, P., 2015. *Ubuntu Unleashed*. 2015th ed. Indianapolis: Pearson Education, Inc.

Huang, K., Zhang, J., Tan, W. & Feng, Z., 2015. An Empirical Analysis of Contemporary Android Mobile Vulnerability Market. In *2015 IEEE International Conference on Mobile Services*. New York, NY, 2015. IEEE.

Huber, M., Horsch, J., Velten, M., Weiss, M. & Wessel, S., 2016. A Secure Architecture for Operating System-Level Virtualization on Mobile Devices. In Lin, D., Wang, X. & Yung, M., eds. *Information Security and Cryptology*. Cham, 2016. Springer International Publishing.

Hu, V.C., Ferraiolo, & Kuhn, R.D., 2006. *Assessment of access control systems*. US Department of Commerce, National Institute of Standards and Technology.

IDC Research, Inc., 2015. *Smartphone OS Market Share, 2015 Q2*. [Online] Available at: http://www.idc.com/prodserv/smartphone-os-market-share.jsp [Accessed 19 July 2016].

IEEE, 2017. *IEEE-SA -IEEE Get 802 Program - 802.11: Wireless LANs.* [Online] Available at: http://standards.ieee.org/about/get/802/802.11.html [Accessed 28 January 2017].

Institute of Electrical and Electronics Engineers, Inc, 2012. *802.1X-2010 - Revision of 802.1X-2004*. [Online] Available at: http://www.ieee802.org/1/pages/802.1x-2010.html [Accessed 29 August 2017].

International Telecommunication Union, 2008. *X.1205 : Overview of cybersecurity*. [Online] Available at: https://www.itu.int/rec/T-REC-X.1205-200804-I [Accessed 4 July 2017].

ISO. 1989. *ISO 7498-2:1989*. ISO.

ISO. 1994. *ISO 7498-1:1994*. ISO.

Ji, Y. & Xia, L., 2016. Improved Chameleon: A Lightweight Method for Identity Verification in Near Field Communication. In *2016 International Symposium on Computer, Consumer and Control (IS3C)*., 2016. IEEE.

KeeLog, 2016. *VideoGhost*. [Online] Available at: https://www.keelog.com [Accessed 28 January 2017].

Kerrisk, M., 2010. *The Linux programming interface: a Linux and UNIX system programming handbook*. No Starch Press.

KeyCarbon LLC, 2017. *KeyCarbon Computer Security Hardware*. [Online] Available at: http://www.keycarbon.com [Accessed 28 January 2017].

Khasawneh, M., Kajman, I., Alkhudaidy, R. & Althubyani, A., 2014. A Survey on Wi-Fi Protocols: WPA and WPA2. In P.G. Martínez, S.M. Thampi, R. Ko & L. Shu, eds. *Recent Trends in Computer Networks and Distributed Systems Security: Second*

International Conference, SNDS 2014, Trivandrum, India, March 13-14, 2014, Proceedings. Berlin, Heidelberg: Springer Berlin Heidelberg. pp.496-511.

Kizza, J.M., 2015. *Guide to Computer Network Security*. 3rd ed. London: Springer-Verlag.

Klauer, B., Haase, J., Meyer, D. & Eckert, M., 2017. Wireless sensor/actuator device configuration by NFC with secure key exchange. In *2017 IEEE AFRICON*., 2017.

Lee, D., 2009. *BBC*. [Online] Available at: http://news.bbc.co.uk/2/hi/technology/8279549.stm [Accessed 27 November 2017].

Lettner, M., Tschernuth, M. & Mayrhofer, R., 2011. Mobile platform architecture review: android, iphone, qt. In *International Conference on Computer Aided Systems Theory*., 2011. Springer-Verlag Berlin Heidelberg.

Linux Containers, NA. *Linux Containers*. [Online] Available at: https://linuxcontainers.org [Accessed 2 January 2018].

Linux man page, nd. *crypto(3) - Linux man page*. [Online] Available at: https://linux.die.net/man/3/crypto [Accessed 23 January 2018].

Luby, M.G., Mitzenmacher, M., Shokrollahi, M.A., Spielman, D.A. & Stemann, V., 1997. Practical Loss-resilient Codes. In *Proceedings of the Twenty-ninth Annual ACM Symposium on Theory of Computing*. New York, NY, USA: ACM. pp.150-59.

Mah, P., 2014. *How to Build a Storage and Backup Strategy for Your Small Business*. [Online] Available at: https://www.cio.com/article/2378019/small-business/how-to-build-a-storage-and-backup-strategy-for-your-small-business.html [Accessed 10 December 2017].

Mansfield-Devine, S., 2010. Security through isolation. *Computer Fraud & Security*, 2010(11), pp.8-11.

Mansour, A., Sambra, A.V., Hawke, S., Zereba, M., Sarven, C., Ghanem, A., Aboulnaga, A. & Berners-Lee, T., 2016. A Demonstration of the Solid Platform for Social Web Applications. In *Proceedings of the 25th International Conference Companion on World Wide Web*. Montreal, Quebec, Canada, 2016. International World Wide Web Conferences Steering Committee.

Microsoft Corporation, 2016. *Windows and Windows 10 - Microsoft*. [Online] Available at: https://www.microsoft.com/en-za/windows [Accessed 10 February 2016].

Miller, C., Blazakis, D., Zovi, D.D., Esser, S., Iozzo, V. & Weinmann, R.-P., 2012. *iOS Hacker's Handbook*. John Wiley & Sons.

Misra, A. & Abhishek, D., 2013. *Android Security: Attacks and Defenses*. 1st ed. [eBook] Boca Raton, FL: CRC Press.

National Institute of Standards and Technology, 2016. *National Vulnerability Database*. [Online] Available at: https://web.nvd.nist.gov/view/vuln/statistics [Accessed 15 July 2016].

Negus, C. & Bresnahan, C. 2012. *Linux Bible (Bible)*. [online]. [Accessed 12 February 2016].

Net Applications.com, 2016. *NetMarketShare*. [Online] Available at: https://www.netmarketshare.com/operating-system-market-share.aspx?qprid=8&qpcustomd=1 [Accessed 4 July 2016].

Netmarketshare, 2016. *Mobile/Tablet Operating System Market Share*. [Online] Available at: https://www.netmarketshare.com/operating-system-market-share.aspx?qprid=8&qpcustomd=1 [Accessed 19 July 2016].

Neumann, P.G., 2014. Risks and Myths of Cloud Computing and Cloud Storage. *Communications of the ACM*, 57(10), pp.25-27.

Northamptonshire Childminding Association, 2015. *Legal requirements for childminders*. [Online] Available at: http://www.childmindinguk.com/legal-requirements/ [Accessed 2 March 2015].

O'Brien, D., 2016. *The Apple Threat Landscape*. Symantec Corporation.

Olivier, M.S., 2009. *Information Technology Research. A practical guide for Computer Science and Informatics*. 3rd ed. Pretoria: Van Schaik Publishers.

OpenVPN Inc., 2018. *OpenVPN*. [Online] Available at: https://openvpn.net [Accessed 29 January 2018].

Oxford University Press, 2018. *Oxford Living Dictionary*. [Online] Available at: https://en.oxforddictionaries.com/definition/form_factor [Accessed 31 May 2018].

Padgette, J., 2017. Guide to bluetooth security. *NIST Special Publication*, 800, p.121.

Peffers, K., Tuunanen, T., Rothenberger, M.A. & Chatterjee, S., 2007. A Design Science Research Methodology for Information Systems Research. *Journal of Management Information Systems*, 24(3), pp.45-77. Available at: http://0-search.ebscohost.com.ujlink.uj.ac.za/login.aspx?direct=true&db=bth&AN=28843849&site=eds-live&scope=site.

Preimesberger, C., 2009. *Survey Indicates Half of SMBs Have No Disaster Recovery Plan*. [Online] Available at: http://www.eweek.com/c/a/Data-Storage/Survey-Indicates-Half-of-SMBs-Have-No-Disaster-Recovery-Plan-687524 [Accessed 22 October 2015].

Raho, M., Spyridakis, A., Paolino, M. & Raho, D., 2015. KVM, Xen and Docker: A performance analysis for ARM based NFV and cloud computing. In *2015 IEEE 3rd*

Workshop on Advances in Information, Electronic and Electrical Engineering (AIEEE)., 2015. IEEE.

Raho, M., Spyridakis, A., Paolino, M. & Raho, D., 2015. KVM, Xen and Docker: A performance analysis for ARM based NFV and cloud computing. Riga, 2015. IEEE.

Raspberry Pi Foundation, 2017. *Raspberry Pi 3 Model B*. [Online] Available at: https://www.raspberrypi.org/products/raspberry-pi-3-model-b/ [Accessed 2 January 2018].

Reshetova, E., Karhunen, J., Nyman, T. & Asokan, N., 2014. Security of OS-level virtualization technologies. In *Nordic Conference on Secure IT Systems.*, 2014. Springer International Publishing.

Resilio, Inc., 2017. *Resilio*. [Online] Available at: https://www.resilio.com [Accessed 12 December 2017].

Rosenblum, M., 2004. The Reincarnation of Virtual Machines. *Queue*, 2(5), pp.34-40.

Rosenblum, M. & Garfinkel, T., 2005. Virtual Machine Monitors: Current Technology And Futur Trends. *Computer*, 38(5), pp.39-47.

Rutkowska, J., 2016. *Qubes OS*. [Online] Available at: https://www.qubes-os.org [Accessed 26 July 2016].

Rutkowska, J., 2018. *Copy and Paste | Qubes OS*. [Online] Available at: https://www.qubes-os.org/doc/copy-paste/ [Accessed 4 February 2018].

Rutkowska, J., 2018. *Copying Files between qubes | Qubes OS*. [Online] Available at: https://www.qubes-os.org/doc/copying-files/ [Accessed 4 February 2018].

Rutkowska, J., 2018. *Qrexec3 | Qubes OS*. [Online] Available at: https://www.qubes-os.org/doc/qrexec3/ [Accessed 4 February 2018].

Rutkowska, J. & Wojtczuk, R., 2010. *Qubes Architecture Overview*. [Online] Available at: https://www.qubes-os.org/attachment/wiki/QubesArchitecture/arch-spec-0.3.pdf [Accessed 6 September 2016].

Sandhya, S. & Devi, K.S., 2012. Analysis of Bluetooth threats and v4.0 security features. In *2012 International Conference on Computing, Communication and Applications (ICCCA).*, 2012. IEEE.

Sarvabhatla, M., Reddy, M.C. & Vorugunti, C.S., 2015. A Secure and Light Weight Authentication and Key Establishment Framework for Wireless Mesh Network. In *Proceedings of the Third International Symposium on Women in Computing and Informatics*. Kochi, India, 2015. ACM.

Smith, O., 2012. *Signal-blocking wallpaper stops Wi-Fi stealing (and comes in a snowflake pattern!)*. [Online] Available at: http://edition.cnn.com/2012/07/18/tech/signal-blocking-wallpaper-stops-wi-fi-stealing-and-comes-in-a-snowflake-pattern/index.html [Accessed 2015 March 2015].

Smith, J.E. & Nair, R., 2005. The architecture of virtual machines. *Computer*, 38(5), pp.32-38.

Subgraph, 2014. *Subgraph*. [Online] Available at: https://subgraph.com/index.en.html [Accessed 26 July 2016].

Tahoe-LFS, 2017. *Tahoe-LFS*. [Online] Available at: https://tahoe-lafs.org/trac/tahoe-lafs [Accessed 12 December 2017].

Tanenbaum, A.S. & Bos, H., 2015. *Modern Operating Systems*. New Jersey: Pearson Education, Inc.

The CentOS Project, 2017. *CentOS Project*. [Online] Available at: http://mirror.centos.org/altarch/7/isos/aarch64/CentOS-7-aarch64-rootfs-7.3.1611.tar.xz [Accessed 2 January 2018].

The CentOS Project, 2017. *Raspberry Pi with CentOS 7*. [Online] Available at: https://www.centos.org/forums/viewtopic.php?t=62548 [Accessed 2 January 2018].

The containerd authors, 2018. *containerd*. [Online] Available at: https://containerd.io [Accessed 5 September 2018].

The Freenet Project Inc. , 2017. *Freenet*. [Online] Available at: https://freenetproject.org/index.html [Accessed 12 December 2017].

The Kubernetes Authors, 2018. *kubernetes*. [Online] Available at: https://kubernetes.io [Accessed 5 September 2018].

The Linux Foundation, 2013. *The Xen Project*. [Online] Available at: https://www.xenproject.org [Accessed 6 September 2016].

The Linux Foundation, 2013. *The Xen Project*. [Online] Available at: https://www.xenproject.org [Accessed 2 January 2018].

Thiel, D., 2016. *iOS application security: The definitive guide for hackers*. 1st ed. San Francisco, CA: No Starch Press, Inc.

Treat, D.G., 2002. Keyboard Encryption. *Potentials*, 21(3), pp.40-42.

United Kingdom. 1998. *Data Protection Act*. The National Archives.

USB 3.0 Promoter Group, 2017. *USB Type-C Authentication Specification*. [Online] Available at: http://www.usb.org/developers/docs/usb_32_020718.zip [Accessed 15 February 2018].

Wang, T., Lu, K., Chung, S. & Lee, W., 2013. Jekyll on iOS: When Benign Apps Become Evil. In *22nd USENIX Security Symposium*. Washington, DC, 2013. Usenix Security 13.

WiFi Alliance, 2013. *Wi-Fi CERTIFIED Miracast™: Extending the Wi-Fi® experience to seamless video display - Consumer (2013)*. [Online] Available at: http://www.wi-fi.org/file/wi-fi-certified-miracast-extending-the-wi-fi-experience-to-seamless-video-display-consumer [Accessed 19 November 2014].

Xavier, M.G., Neves, M.V., Rossi, F.D., Ferreto, T.C., Lange, T. & De Rose, C.A.F., 2013. Performance Evaluation of Container-Based Virtualization for High Performance Computing Environments. In *2013 21st Euromicro International Conference on Parallel, Distributed, and Network-Based Processing*., 2013. IEEE.

Yang, L., Cao, J., Yuan, Y., Li, T., Han, A. & Chan, A., 2013. A Framework for Partitioning and Execution of Data Stream Applications in Mobile Cloud Computing. *SIGMETRICS Perform. Eval. Rev.*, 40(4), pp.23-32. Available at: http://0-doi.acm.org.ujlink.uj.ac.za/10.1145/2479942.2479946.

Appendix A: Special foldouts

TABLE 11: SECURE CONTAINER PROPERTY - REQUIREMENTS

Secure Container Property - Requirements	
SCP1. Design Methodology	
SCP1.1	Hypervisor level isolation
SCP2. Application communication	
SCP2.1	Applications inside containers communicate with each other using modern operating system objects.
SCP2.2	Applications in different containers cannot communicate with each other.
SCP2.3	Communication between containers is only allowed through strongly controlled mechanisms.
SCP3. Privacy protection	
SCP3.1	Owner based isolation of data and applications in containers.
SCP3.2	Configuration and access control using policy enforcement.
SCP4. Protected management environment	
SCP4.1	Unauthorized changes to the management environment are not allowed.
SCP4.2	Updating of the management environment should guarantee integrity during update and after the update.
SCP4.3	The integrity of the management environment should be confirmed by the hardware
SCP4.4	Network connections to the management environment are not allowed.
SCP5. File system deployment	
SCP5.1	The use of templates provides baseline container configuration.
SCP5.2	Access to removable storage is strongly controlled.
SCP5.3	Deployed containers are encrypted on the hard disk
SCP6. Network access security	
SCP6.1	No direct access from inside any application and data containers to the network, instead of indirect mechanisms should be used.
SCP6.2	Data and application containers can only access network traffic directed to and from it.

TABLE 12: MUTUAL AUTHENTICATION PROPERTY - REQUIREMENTS

Mutual Authentication Property - Requirements	
MAP1: Identification and authentication	
MAP1.1	Mutual authentication is required for Neo device and I/O peripheral
MAP1.2	OoB mechanisms must be used during initial identification and authentication.
MAP2: Authorisation	
MAP2.1	Access control triplets must be used to determine access. The access control triplet consists of a user, I/O peripheral and container identifier.
MAP2.2	Access control is managed by the management system of the Neo model.
MAP2.3	The owners of containers set access control.
MAP3: Confidentiality	
MAP3.1	Only encrypted communication is allowed between Neo device and I/O peripheral.
MAP3.2	The management of encryption keys should occur in the management environment and not from within containers where users have access.