Analyzing, Designing & Implementing a Secure, Centralized, Generic, Extensible, Input-Sanitizing, Intuitive Configuration Management Station

Master's Thesis, November 2015

E. Ton
<<Page left blank intentionally>>
Analyzing, Designing & Implementing a Secure, Centralized, Generic, Extensible, Input-Sanitizing, Intuitive Configuration Management Station

THESIS

Submitted in the partial fulfillment of
The requirements for the degree of

MASTER OF SCIENCE

in

COMPUTER SCIENCE
TRACK INFORMATION ARCHITECTURE

by

E. Ton
Born in The Hague, the Netherlands

Web Information Systems
Department of Software Technology
Faculty, EEMCS, Delft University of Technology
Delft, the Netherlands
http://wis.ewi.tudelft.nl

Fox-IT
Olof Palmestraat 6
Delft, the Netherlands
http://www.fox-it.com
Analyzing, Designing & Implementing a Secure, Centralized, Generic, Extensible, Input-Sanitizing, Intuitive Configuration Management Station

Author: E. Ton  
Student ID: 1344862  
Email: e.ton@student.tudelft.nl

Abstract

This thesis describes the analysis, design and implementation of a management station that is able to support different security network products. It is designed to be centralized so it can support multiple devices from one interface. The management station is generic and extensible, to support a range of different products. As the management station could be used in high assurance organizations, security of the product is taken into account during the whole process. For usability we looked into ways to make the usage more intuitive to users. It also contains a model that improves the quality of configurations used by connected devices by sanitizing user input on correctness.

Keywords: security, management station, centralized, genericity, extensibility, input sanitation checking, intuitive, configuration management

Future work: extending functionalities of FMS, hardening the device in the network

Graduation Committee:
Prof. dr. ir. G.J. Houben, Faculty EEMCS, TU Delft  
Dr. ir. A.J.H. Hidders, Faculty EEMCS, TU Delft  
Dr. ir. J.H. Weber, Faculty EEMCS, TU Delft  
ir. A.N.de Jong, Fox-IT

1 Electrical Engineering, Mathematics and Computer Science
Preface

I have been interested in information security for a long time. When I got the chance to do my Master's thesis at Fox-IT, it was a really nice opportunity to be able to combine my security interests with work experience. I have worked during the first part of this project part-time at Fox-IT. There I got the chance to use professional security products, had interesting security related discussions and learned new useful technologies.

During the second part of this project where I had to write the thesis, I worked as a freelance software developer and later as security specialist at another company. This combination was far from optimal and finishing the thesis took me therefore a lot of time. I am very happy to finally finish this thesis.

From Fox-IT I want to thank my supervisor Adriaan de Jong for his help during the project and feedback on this thesis. From the TU Delft I want to thank my supervisor Jan Hidders for his guidance, help and input.

Finally I want to thank my boyfriend, friends and family for their support.
Contents

Preface ...................................................................................................................... ii
List of Figures ........................................................................................................ vii
1. Introduction ......................................................................................................... 1
  1.1. About Fox-IT .................................................................................................. 1
  1.2. Problem description ....................................................................................... 2
    1.2.1. Introduction ............................................................................................. 2
    1.2.2. Research problem, goal & objectives .................................................. 3
    1.2.3. Deliverables .............................................................................................. 3
  1.3. Approach ......................................................................................................... 4
    1.3.1. Methodology ........................................................................................... 4
    1.3.2. Research approach and outline .......................................................... 4
    1.3.3. Validation ................................................................................................ 5
2. Related work ......................................................................................................... 6
  2.1. Secure systems ............................................................................................... 6
    2.1.1. High Assurance demands & classifications ........................................ 7
    2.1.2. Secure software development: CLASP & SDL .................................. 8
    2.1.3. Validating the security of a system ..................................................... 11
  2.2. Generic and extensible software .................................................................... 13
    2.2.1. Approach for analysis and architectural design. ............................... 13
    2.2.2. Implementation ....................................................................................... 14
  2.3. Sanity testing of user input .......................................................................... 15
3. Environment analysis ......................................................................................... 16
  3.1. Security demands & development process .................................................. 16
  3.2. Genericity & extensibility: analysis current Fox-IT Security Devices ........... 17
    3.2.1. Approach ............................................................................................... 17
    3.2.2. General description of the different devices ....................................... 17
    3.2.3. Analysis security devices ...................................................................... 19
  3.3. Usability analysis: current problems with devices experienced by customers ... 23
    3.3.1. Stakeholders & user roles ..................................................................... 23
    3.3.2. Analysis / interviews outcomes ......................................................... 23
  3.4. General requirements ..................................................................................... 24
    3.4.1. Requirements by Fox-IT and problem statement (chapter 1) .............. 24
# List of Figures

Figure 1. Information Systems Research Framework [2] ........................................... 4
Figure 2: The model-based security testing method [15] ........................................... 12
Figure 3: Overview of the DataDiode ...................................................................... 18
Figure 4: Overview of a Sina network ...................................................................... 18
Figure 5: Overview of Skytale (payload encryptor) in a vehicle .............................. 19
Figure 6: general workflow within the management station .................................... 32
Figure 7: example of the FMS within a network with current Fox-devices ............. 33
Figure 8: Specific application of the FMS for the Data Diode ................................. 34
Figure 9: Running locally ......................................................................................... 35
Figure 10: Containers and their interactions ............................................................. 35
Figure 11: Major components and interactions ......................................................... 37
Figure 12: the bootstrapping problem ..................................................................... 39
Figure 13: High level data flow ............................................................................... 42
Figure 14: Trust zones ............................................................................................. 43
Figure 15: Attack tree access LAN/VPN .................................................................. 44
Figure 16: Attack tree of attacks on the FMS ........................................................... 45
Figure 17: Mitigations of possible attacks ................................................................. 49
Figure 18: Components and packages in prototype ................................................. 51
Figure 19: screenshot of the GUI .............................................................................. 53
Figure 20: Class diagram of the itemgroups .............................................................. 54
Figure 21: Configuration with different itemgroups (tabs) and validation errors ...... 55
Figure 22: Workflow of the configuration process .................................................... 56
Figure 23: Overview of the device .......................................................................... 57
Figure 24: class diagram of the device and config structure .................................... 58
Figure 25: use case diagram of the devicetype component ..................................... 59
Figure 26: Screenshot for defining the device type’s configuration schema .......... 60
Figure 27: Screenshot for adding a new configuration item with its variables ...... 61
Figure 28: Screenshot of the ‘configuration layout’ screen ...................................... 62
Figure 29: class diagram of the devicetype specification ......................................... 63
Figure 30: validation process .................................................................................. 65
Figure 31: screenshot for adding an individual validation to a configuration item ...... 66
Figure 32: screenshot of step 1: add a validation rule .............................................. 67
Figure 33: adding propositions ............................................................................... 67
Figure 34: Adding implications .............................................................................. 68
Figure 35: extending functionality - adding new groups with new rights ............. 70
Figure 38: One of the needed items being added .................................................... 84
Figure 39: List of items ........................................................................................... 85
Figure 40: Defining the layout of the configfile ....................................................... 85
Figure 41: Part of the generated configuration screen ............................................. 86
Figure 42: Configuring the FMS module ................................................................. 86
1. Introduction

Network and system administrators have to manage and configure complex networks, different security network devices (e.g. routers, firewalls, VPN-solutions) and servers with different security software. Usually each of these devices is configured through its own user interface, running locally on the device. Their configuration is adapted through running programs with parameters from the shell, changing local text-based configuration files or through their own graphical user interfaces.

These procedures have different pros and cons: on the one hand it is obvious and quite secure – you need to have access to all the different devices/servers to be able to modify them, on the other hand it is time-consuming and difficult to get into: each device has its own format and procedures to be able to configure the device. Configuring multiple devices also takes longer: each device has its own login procedures which need to be followed first.

Fox-IT creates and sells these security devices. They are mostly configured by the customers themselves. Fox-IT wants to offer them a solution that improves this process.

This thesis is about improving the way end-users configure their (security) devices.

In this chapter an introduction about the organization where this research is done is given and the research problem, deliverables and approach are formalized.

A vocabulary is added to explain specific terminology in Appendix 1.

1.1. About Fox-IT

Fox-IT is a company in Delft which focuses on IT security. It is currently one of the largest commercial IT security companies in the Netherlands. Its goal is to create technical and innovative solutions that help create a more secure society. They do this by developing and offering advanced services and solutions in the field of cyber security and cyber defense. These solutions are used worldwide.

Fox-IT is active in different fields: they create security solutions that help protect secret information within governments, but they also offer security consultancy, forensic research for law enforcement, help banks detect fraud and monitor company infrastructure for security breaches. [1]

This research has been done at Fox-IT’s Crypto R&D division, which focuses on developing new solutions that enhance national security.
1.2. Problem description
In this section the topic of this research is elaborated on: the problem Fox-IT faces and the research problem of this thesis with its derived research questions and deliverables.

1.2.1. Introduction
Each security device developed by Fox-IT has its own configuration method: a specific configuration file, services on servers that have to be started with certain parameters, or its own graphical user interface. This is not optimal because:

- Text based configuration files are in one way complete and obvious, but for many end-users their use is not: the mix of comments and variables with their values and cohesion can make an overview unclear.
- Configurations are not checked for sanity before used by the device, which makes it sensitive to user errors.
- If there are relations between different network systems, they are not checked in relation to each other. Errors made in one device can impact the security on another device and weaken the total network.
- Every interface is different: there is no uniform procedure for an end-user and there is not a specific recognizable GUI from Fox.

Fox-IT wants to offer their customer a secure centralized management station, but centralized solutions also bring possibly new problems: the possibility to configure and manage multiple (different) devices from one place will make that management station a possible security flaw. It will also be a more interesting target: compromise one system and get access to all connected systems.

Another problem is that most Fox Crypto-developers are back-end developers, they are able to create front-end interfaces, but prefer not to. A front-end system that could be reused with minimum effort to suit new products would allow the developers to work on more loved projects. That front-end system should be set up generic and extensible, to be able to incorporate these new products.
1.2.2. Research problem, goal& objectives

Based on this problem description, the following research problem and questions can be abstracted:

**Design a secure, centralized, generic, extensible, intuitive, input-sanitizing configuration management station and create a proof of concept of this design.**

To be able to achieve this goal, the following research questions have to be answered:

**RQ1**
What are the (quality) **security** demands by Fox-IT?
- How can these demands be met in a design?
- How is it possible to validate if these demands are met?

**RQ2**
What are (security) problems that arise when creating a **centralized** management station instead of decentralized management interfaces?
- Can these issues be mitigated? How?
- Is it feasible; does it weigh against the cons?

**RQ3**
What type and to what level is **extensibility** and **genericity** is required?
- How to apply this in a software design?
- How can this be tested?

**RQ4**
How can the configuration process done by end-users be more **intuitive** to them?
- What are problems that they currently experience?

**RQ5**
How can configurations be **checked for sanity**?
- Can these checks be captured in a model?
- What is an appropriate language to describe these checks?
  - What can that language do and what are its limitations?
  - How to implement that language efficiently?

1.2.3. Deliverables

We create a solution for the research problem by developing a functional and architectural design. This will be partially implemented in a prototype that demonstrates some of the concepts. Together with the design, it solves the research problem.
1.3. **Approach**

This section describes the approach we will use to come to the design and prototype.

1.3.1. **Methodology**

This thesis is a design / engineering thesis. This needs a different approach than normal design or behavioral science researches: with only a design approach there is a chance the designed artifact doesn't really connect with the organization, and with a behavioral science approach theories might not work in a 'real' organization. [2]

A combination of these approaches is needed, where based on an environment analysis and theoretical knowledge, theories are formed and developed, which are tested on the environment. This is a repeating process. A summary of this approach is displayed in the diagram below:

![Information Systems Research Framework](image)

**Figure 1. Information Systems Research Framework** [2]

1.3.2. **Research approach and outline**

Based on this methodology, a general approach can be formalized for this thesis project.

**Environment**

First we get to know the organization and the products. Because the research assignment is done most of the time at Fox-IT itself, a good overview of the organization and people is possible and we can answer RQ1 partial.

To be able to answer the organizational genericity and extensibility (RQ3 partial) aspect of the research problem, we look into security devices already developed by Fox-IT. We try to map the functionality and similarities of these devices and extract a level at which point genericity and extensibility should be supported. We also describe the results of interviews done with people within the organization to be able to answer the other research questions and determine current problems that customers experience when
using their products (RQ1, RQ4). In chapter 3 this analysis is described. It answers the specific research questions on the environment.

**Knowledge Base**

In chapter 2 we do a literature survey to be able to answer some of the theoretical research questions (RQ1, RQ2, RQ3 technically, RQ5), and to get a better overview on secure, generic and extensible software development.

**IS Research**

After finishing the analysis and literature survey, we will be able to form requirements. We then make a general design of a system that might be able to solve the research problem. The design and implementation of the prototype is an iterative process and is discussed during weekly meetings with the company supervisor.

Some of the concepts of this design will be implemented in a prototype. Together with this design they will be validated for answering the research problem.

In chapter 4 the requirements are used to create a general design. In chapter 5 a security architecture is formed, based on Common Criteria and a threat analysis. Chapter 6 discusses the implementation of the design in a prototype. Conclusions and recommendations on this research can be found in chapter 7.

**1.3.3. Validation**

We can validate if the research problem has been solved through the following mechanisms:

- **Security (RQ1, RQ2):** we create a security architecture that checks for the coverage of mitigations on possible threats and attacks on the system. This architecture is reviewed by an expert.
- **Extensibility, genericity (RQ3):** incorporate a new product into the management station that was not taken into account during the design and implementation phase of the management station.
- **Intuitiveness and correctness of configurations (RQ4, RQ5):** logical reasoning on the new configuration and sanitization procedure.
2. Related work

This chapter is a literature survey, done to be able to answer the theoretical research questions. We focus here on the topics that go into:

- Secure system development and standards for high secure networks (RQ1, RQ2)
- Generic and extensible software (RQ3)
- Improving usability by reducing configuration errors in the user interface (RQ5)

We chose to look into secure system development because devices that Fox-IT develops, land in high secure environments, with high demands on their security products. We look at these demands by studying the Common Criteria[3]: these are open standards that products have to meet to be certified at a certain assurance level and be deployed at a network that's secured on that level. This will answer our first research question RQ1. It also gives direction on implementing security during the design phase of the architecture.

In section 2.1 we will focus on secure software development processes and ways to validate a system as secure. This will also contribute to research question RQ1.

Section 2.2 goes into methodologies and technologies for designing and implementing generic and extensible software, after that we will be able to answer research question RQ3.

In section 2.3 we will look at literature for sanity checking of user input to answer research question RQ5.

Based on these topics together with the analysis done in chapter 3, requirements for the management station can be formed. These will be described at the end of chapter 3.

2.1. Secure systems

'Security' is a difficult requirement to implement: 100% security is not reachable and cannot be fully guaranteed. It often depends on an attacker's effort and resources if a system can be compromised. Security is not a feature you can 'add' later in the development process, it has to be taken into account during the whole development process: from analysis to design, to implementation, testing and auditing.

Securing a system is therefore an ongoing process of removing as much uncertainty as possible. It is still a broad research field and vulnerabilities can never be fully excluded. It is possible to give a certain security quality assurance on a device by having the development phase and product audited and certified at a certain level. This certification says the product has certain security requirements. It doesn't say it is secure though. Governmental organizations working with secure information are for example required to use security products that are certified at a certain level. We call these organizations 'high assurance organizations'.
In this section we will look into demands of high assurance organizations by looking at the so-called 'Common Criteria': an international standard for certification of security products.

After that we will look at the CLASP model, which is a methodology to support the secure system development process. We conclude this section with an investigation on ways to validate a system as secure: by auditing the complete development process and white and black box security testing.

### 2.1.1. High Assurance demands & classifications

When working with high secure environments where data loss, modifications or unavailability cannot be risked, all possible efforts must be made to secure these networks. In these (e.g. governmental) environments, information and networks are given a certain classification based on the possible impact of an attack. To protect these networks, security products with an appropriate assurance level to mitigate the identified threats in the environment, are used.

High assurance products are more trusted and give more assurance than other products. This is accomplished through exhaustively and rigorously testing of source code, documentation and review of the development process by external auditors. After that, the product can be certified. There are usually different assurance levels.

The certifications are given when devices are functionally correct and satisfy appropriate safety and security properties [4]. Usually there are different levels of certification, depending on the extensiveness in time or depth of the audit. Fox-IT develops high secure products, and these are also certified by different certification institutes at different levels worldwide.

We look in more detail at the standards of Common Criteria because these are standards that Fox-IT uses, they are open, accessible and widely used.

**Common Criteria**

The Common Criteria is an international standard for computer security certification [ISO/IEC 15408]. With this model a vendor can certify that a product’s security functions meet certain security requirements.

A customer can create a Protection Profile (PP) with security functional requirements (SFR) for their environment, which vendors can claim their device meet. Their device will be evaluated by a Common Criteria testing institute for having these security functional requirements. Examples of these functional requirements are: ‘using authentication roles’ and ‘the ability to limit access on role-level’.

Products can also be tested at different levels of assurance. Depending on the extent of testing, a certain Evaluation Assurance Level (EAL) level of confidence can be given to a product. It goes from range EAL-1 (least tested) until EAL-7 (most excessively tested).
There is criticism on the use of certification for security products: it doesn’t guarantee that the product is actually secure, it takes a lot of time and money before it is actually certified (and then the product could be obsolete already). Also open source products are not usually certified because of the cost aspect. They are therefore not used by organizations (e.g. governments), that have a policy only to use certified products. [5]

**NLNCSA**
The Netherlands National Communications Security Agency (NLNCSA) is a unit of the AIVD (the General Intelligence and Security Service of the Netherlands). One of their tasks is advising the Dutch government on securing high-secure networks [6]. They do this by certifying products and giving advice (‘inzetadvies’) on how to configure and install these products.

**2.1.2. Secure software development: CLASP & SDL**
To develop secure software, we looked at the OWASP CLASP (Comprehensive, Lightweight, Application Security Process) model and Microsoft SDL (Security Development Lifecycle).

**CLASP**
CLASP is a formalized approach for secure software development. It addresses security in a broad perspective and can be flexibly used within a current or new development project[7][8].

Using CLASP in a development process can help improve the security of the overall development process by providing overviews of vulnerabilities, security principles, best practices, supporting tools and roadmaps that can be followed. The methodology is structured, repeatable and measurable.

As traditional programmers were supposed to implement security in a system, CLASP assigns the different security activities to roles in a development team:

- **Project manager**: promotes the importance of security in general within the development team, (users) and stakeholders.
- **Requirements specifier**: also identifies requirements that have possible security issues on the business level.
- **Architect**: understands the security impact of each used technology, defines user roles and defines their access on parts of the system and how these parts interact.
- **Designer**: chooses technologies with their vulnerabilities and how to address these, documents the attack surface of the complete system.
- **Implementer**: implements the design in a secure way, notifies and documents designers on risks and concerns and end-users on their responsibilities.
- **Test Analyst**: security testing, but hasn’t got in depth knowledge on security.
- **Security Auditor**: audits the security requirements, analyzes the design and inspects the implementation for vulnerabilities.
Here the designer is most technically security aware (instead of the programmer). Each role has its responsibilities and high-level activities (like creating a threat model or performing a source-level security review) [9]. The project manager makes an assessment of the relevant activities that have to be done for a project.

The roadmaps CLASP provides support for different kinds of software development frameworks: a waterfall approach, as well as an iterative approach are supported.

**SDL**

Microsoft has developed a secure development process that’s also used internally since the development of Windows Vista. It helped reducing bugs in Windows Vista in comparison with Windows XP.

It consists of a set of phases with activities that address multiple security issues and can be integrated in Microsoft’s development process. It is extensive, detailed and practical in how to execute each activity: every activity had methods and tools that can be used [10][11]

The different phases of the lifecycle are:

- **Training**: a core security training for software developers, that include secure design, threat modeling, secure coding, security testing and best practices surrounding privacy.
- **Requirements**:
  - Establish security requirements
  - Create a minimum 'bug bar': the least quality the software has to reach.
- **Design**
  - Establish security and privacy concerns
  - Attack surface analysis/reduction
  - Threat modeling
- **Implementation**
  - Create a list of approved tools and their security checks
  - Deprecate unsafe functions
  - Perform static analysis
- **Verification**
  - Perform dynamic analysis: test application for security issues
  - Fuzz testing: test application on random or malformed data
  - Attack surface review
- **Release**
  - Create incident response plan
  - Conduct final security review: review all security activities
  - Certify release and archive all documents
- **Response**
  - Execute incident response plan: protect customers from security and privacy flaws after release.
Comparison
In [12], Grégroire et al. compare the 2 development processes by defining phases in the general development process, where the activities of both methodologies are grouped under. These phases are compared on differences and similarities:

Education and awareness: Both methodologies promote an initial and periodical education on security. SDL focuses on education of the developers, CLASP states that all roles should include training.

Project inception: Both methodologies appoint a security advisor that overviews the whole project. Differences in this phase are:

- CLASP uses security metrics that enforce accountability or detection of security issues.
- CLASP develops a global organizational policy for security requirements during this phase.
- SDL defines roles and their interactions during this phase.
- SDL sets up tools that track security issues and defines the bug bar here.
- SDL tests if the type of project is suitable and is fully covered by the methodology.

Analysis: SDL has no activities for this phase. CLASP identifies trust boundaries and specifies the security requirements here.

Design: Both methodologies do threat modeling here. CLASP creates a security architecture. It also acknowledges third-party components, which get a security assessment. SDL does a risk assessment on resources to identify which ones are vulnerable for attacks.

Implementation, testing and verification: SDL has no implementation activities. SDL focuses on black box testing, but does a code review in the end. CLASP focuses on white box testing. They both aim at reducing attack surfaces and test individual components. Only SDL also tests the system as a whole.

CLASP has 2 additional activities, where it uses tools to create an automated implementation security analysis and collects metrics. During this phase, CLASP also checks if all coding guidelines are met and all security requirements are implemented.

Deployment and support: Both methodologies produce procedures and documentation for deployment and response, for when a new vulnerability is discovered (releasing advisories etc.).

In addition, CLASP recommends signing the code, so stakeholders can validate the origin and integrity of the software.

Conclusion
Grégroire et al. conclude that SDL is "more heavyweight and rigorous, making it more fit for large organizations. CLASP is lightweight and more affordable for small organizations with less strict security demands".
2.1.3. Validating the security of a system
There are different ways to validate the security of a system. We found 3 different approaches to this:

- Validating the complete development process: this is done by external certification institutes and security auditors. [13]
- White box security testing: ‘inside’ access to the programming code and infrastructure is needed. [14]
- Black box security testing: testing is done from the perspective of an external user or unauthorized person, where the system is viewed as a box where the content is not known. [15]

In this section, we will look into the different ways of security testing.

**White box security testing**
There are different forms of white box testing, which can be done at different phases of the development process: source code can be reviewed by another person or automated detection programs [16][17], but also during the development process there are measures that can be taken to detect security violations in an early state. We will look here in more detail at code reviews, security unit tests and model based security testing.

**Code reviews**
A code review can be done by auditors or other (more advanced) developers, checking for code or logic flaws in the programming code. It helps finding problems in an early state, improves the quality of the software and helps developers improve their skills.

**Security unit tests**
A unit test is a small piece of code inside software that exists during the development phase. It tests pieces of the software for defects or malfunctioning. A way to scan for coding flaws that breaches security during the development phase, is creating these unit tests: they can translate security requirements that will alert the developer when they are broken [18][19].

**Model-based security testing**
Model-based security testing is a relatively new research field. Its testing is based on generating relevant tests automatically based on a security model, on which the system is developed. This methodology has been formalized, among others, in the ITEA2 Diamonds methodology[20].
Black box security testing
The most used form of black box security testing is penetration testing: a live test of the effectiveness of security defences through mimicking the actions of real-life attackers [21].

Initial tests can be automated by making use of tools like OWASP Mantra [22] or Zed Attack Proxy for web application testing, for e.g. the OWASP top 10: the 10 most important security flaws in web applications[23], or the Backtrack Linux distribution for scanning and searching for vulnerabilities in complete environments.

More specific tests can be done manually or semi-automated after that. The success rate in finding vulnerabilities depends on the skills of the penetration security tester.

---

2.2. **Generic and extensible software**

To be able to create a design that is also usable for future products, we looked at theory on extensible software and its implementation in frameworks and middleware.

“Software extensibility is a system design principle where the implementation takes future growth into consideration.” [25]. Software is extensible when it can be extended by adding modules or extensions that enhance the functionality of a system, without changing or impacting current functionalities.

Generic software is software that can be used for a number of different purposes without requiring modification.

2.2.1. **Approach for analysis and architectural design.**

We looked for an approach that answers research question 3: "how to apply extensibility in a software design?". We found the article by Moore[26], who published his findings on a collaboration between 25 science and engineering domains to design an open source distributed data management system. It focuses on interoperability and extensibility of the system.

Each domain had its own unique semantics, data formats, types of data, analysis procedures, management policies, descriptive metadata, and hardware systems (including unique network access protocols). It has its own existing infrastructure that manages legacy data, provides analysis services, and serves as an authoritative resource for domain knowledge.

The approach they used in building a generic extensible system for all domains, was building middleware that captured all domain knowledge, which is able to translate each specific domain protocols and views to a unified view for the infrastructure-independent collaboration system, where all the domain-specific infrastructure could still be used.

To be able to capture the needed domain knowledge, interoperability mechanisms can be used. They can be categorized in the following namespaces:

- Users (group formation, authorization, authentication, audit)
- Resources (storage interaction, remote application execution, queuing)
- Files (replication, versioning, distribution, streaming, transport)
- Collections (access controls, archiving, soft links, registration)
- Metadata (schema, ontologies, vocabularies)
- Policies (enforcement points, automation, versioning)
- Procedures (workflow provenance, re-execution, versioning, sharing)

This knowledge can be used to implement the **generic, interoperable** new collaboration system.

To also make the system **extensible** and **sustainable** for future new technology, extensibility mechanisms can be used and implemented dynamically in a system. These mechanisms are:
• Policies that control the execution of procedures, management of data, and verification of assessment criteria.
• Micro-services that manage interactions with external network protocols, encapsulate specific operations, and encapsulate workflow operators (conditional tests, loops, arithmetic).
• Drivers that apply data grid operations at remote storage locations

2.2.2. Implementation
To implement portable, modular and extensible software, often a component based approach is chosen to create a plug & play-like environment. Also frameworks and middleware are used to create fast, secure and platform independent environments: frameworks have design patterns based on communication patterns embedded and help develop in an autonomous secure way. With middleware, platforms are able to transfer independent, network transparent objects between components [27].

Django framework
Django is a Python web-framework that is loosely based on the MVC (Model View Controller) design pattern. Its key advantages for this project are that it has a loosely coupled component-based approach, development of software is relatively fast and robust (once you get to know the framework) and the included solutions in the framework for different security problems that arise in web application programming[28].

Middleware for distributed applications
In [29] Schantz et al. give an overview of middleware and its applications. Middleware is invented to reduce the high effort in complexity and costs of developing distributed systems. Its role is to bridge the gap between applications and lower level hardware and software, by coordinating how they are connected. It provides reusable services, which make it possible to create more rapid and robust distributed systems by integrating different components. It decreases complexity, cycle time and the level of effort for developers. It provides them ways to formalize and coordinate the composition of parts and ways to monitor and validate different components, ensuring end-to-end Quality of Service (QoS). There are different types of middleware, on different levels: Host infrastructure middleware (Java VM, .NET), distribution middleware (SOAP), common middleware (CORBA, .NET Webservices) and domain specific middleware.

Bernstein [30] describes middleware as a layer of software which bridges formerly isolated component activities to a information utility, which is accessible by a user of an information utility. Middleware makes it easier for software developers to perform communication and input/output, so they can focus on the specific purpose of their application.
2.3. Sanity testing of user input

To answer research question 5: “how can configurations be checked for sanity?”, we looked into literature on input validation.

The quality of data in an information system is very important: it needs to be complete, valid, consistent, on time and accurate to be able to be used in operations, decision making and planning [31]. To optimize data quality, data cleansing and data enrichment can be used. Data validation is the process of ensuring that a program operates on clean, correct and useful data [32]. Invalid data can cause security issues, but can also pollute an information system.

Data validation can be done on multiple levels:

- Checks on unsafe input (hacking): if not validated, wrong input can be exploited by malicious users, allowing them to overwrite buffers and execute harmful code
- Checks on wrong basic type input (e.g. input of a string instead of an integer, input exceeds maximum length of the type or database field)
- Checks on wrong advanced type input (e.g. an invalid email address or URL)
- Wrong input fields in relation to other

The validation of new data can processed in 2 steps: validation checks and post-checks.

There are different validation checks that can be used:

<table>
<thead>
<tr>
<th>Allowed character checks</th>
<th>Hash totals</th>
</tr>
</thead>
<tbody>
<tr>
<td>Batch totals</td>
<td>Limit check</td>
</tr>
<tr>
<td>Cardinality check</td>
<td>Logic check</td>
</tr>
<tr>
<td>Check digits</td>
<td>Presence check</td>
</tr>
<tr>
<td>Consistency checks</td>
<td>Range check</td>
</tr>
<tr>
<td>Control totals</td>
<td>Referential integrity</td>
</tr>
<tr>
<td>Cross-system consistency checks</td>
<td>Spelling and grammar check</td>
</tr>
<tr>
<td>Data type checks</td>
<td>Uniqueness check</td>
</tr>
<tr>
<td>File existence check</td>
<td>Table Look Up Check</td>
</tr>
<tr>
<td>Format or picture check</td>
<td></td>
</tr>
</tbody>
</table>

The validation methods mentioned above can e.g. be implemented by defining validation rules or enabling white lists as regular expressions.

Post checks are done after the initial validation checks have passed. They can reject or change the sent data, accept the data but advise on ‘strange’ input or request a verification of the data from the user, ‘is this really what you want to send?’
3. Environment analysis

In this chapter an analysis is done on the environment to be able to answer the research questions further:

- What are the (quality) security demands by Fox-IT? (RQ1)
- To what level is genericity and extensibility (re-use of the software) needed by Fox-IT? (RQ3)
- How can the configuration process done by end-users be made more clear? (RQ4)
  - What are problems that they currently experience?

We looked more specifically at the division Crypto to be able to define the security requirements they demand on their own products and their approach to develop secure software. We then look at the security devices developed and implemented by this division and the people working with these devices to get a grasp on the level of extensibility and genericity of the new management software. We use the interoperability and extensibility mechanisms from chapter 2 to help defining this.

We interviewed people who work with customers of the devices, to be able to define current problems that customers experience when configuring the devices.

In the last paragraph requirements are formulated based on this analysis and literature survey in chapter 2.

3.1. Security demands & development process

To be able to determine the level of security for the new management system, we looked at the division Crypto where devices are being developed, we looked at security demands of their customers and the current security quality level during the development process.

Fox-IT Crypto is mainly focused on government security by developing products that improve the security in this field. Their products are certified by different certification institutes on different levels until NATO and state secret level, so they can be found in high secure environments. These customers demand security products that are certified at a certain assurance level. Next to developing products Crypto is also doing consultancy on security issues within organizations and helps with implementations of secure environments.

To be able to meet these demands, the development process is arranged according to the requirements of the NLNCSA: during the whole process, the process and technology is documented and extensively reviewed and tested after each iteration. The project team is highly security aware on implications of their design and implementation decisions. When the product is completed, there is another large review.
3.2. Genericity & extensibility: analysis current Fox-IT Security Devices

To get more feeling on the possibilities and use of security devices, security products developed or sold by Fox-IT Crypto were analyzed. We used the extensibility and genericity mechanisms described at chapter 2 for this analysis.

3.2.1. Approach

To be able to capture the right information from the devices to extract the right level of genericity, we tried to map how each device handles its access control, resources, files, collections, metadata and their policies and procedures [26]. We start with a general description of the devices themselves.

3.2.2. General description of the different devices

First we discuss the general purpose, use and functionality of each device to get an overview of the different products.

**OpenVPN-nl**

OpenVPN-nl is a modified version of the open source software OpenVPN. With the software it is possible to create an encrypted connection between a client and a server over an untrusted network. In OpenVPN-nl a lot of 'insecure' features are removed compared to the 'normal' OpenVPN version, default values set to more secure ones and instead of OpenSSL it makes use of PolarSSL, which is a smaller SSL library that is easier to evaluate by certification authorities. OpenVPN-nl is certified at NLNCSA criteria Level 2 ("Departementaal vertrouwelijk").

**DataDiode**

The DataDiode is a hardware device that can be put between 2 networks and only allows data transfer in one direction. The DataDiode works with 2 proxyservers that handle the transfer of data through the DataDiode: this is done by translating diverse protocols that work normally on TCP/IP to a custom, error-correcting transfer protocol, which sends no confirmation of received data. The DataDiode is used to separate high secure networks from lower secure networks (making sure no data is leaked to the other side) and is worldwide certified as such.
There is currently a new version of software for the proxyservers in development. For this thesis the 'old' appliance is analyzed.

**Sina & Sina Management**

Sina is a product line developed by Secunet and sold and deployed by Fox-IT. It connects high secure networks and workstations over insecure networks by encrypting all data between these networks through OSI IP level (Sina L2) or network level (Sina L3). These devices can be configured by the product 'Sina Management'.

**Figure 3: Overview of the DataDiode**

**Figure 4: Overview of a Sina network**

**Sina Management** is an administrative application to configure all Sina devices by writing configurations to smartcards. These smartcards have to be inserted in the actual Sina boxes or workstations to be able to enable the secure network connections.

**Skytale**

Skytale is a product in development, designed for ad-hoc military vehicle networks with low available bandwiths, that still want to securely communicate over a possible hostile
network. It is a payload encryptor that encrypts all data that goes out of the vehicle through the IP layer.

![Diagram of network security setup]

**Figure 5: Overview of Skytale (payload encryptor) in a vehicle**

### 3.2.3. Analysis security devices

In this paragraph the security devices are analyzed in functionality, their configuration possibilities and workings and the way the configuration is actually picked up by the device and further implemented.

We used the interoperability mechanisms to categorize and analyze the different elements and functions of the devices.

**Core functionality:**

- OpenVPN-nl: creating an encrypted VPN connection from a client to a server over an unsecure network
- DataDiode: only allowing data from a network to 1 other network, not the other way around
- Sina: creating an encrypted connection over an unsecure network between 2 endpoints
- Sina MS: configure the Sina network elements and write their configurations to smartcards
- Skytale: create an encrypted connection between different vehicles and a elsewhere located trusted network over an unsecure network

**Users/ user access**

**User access to configuration parts:**

- OpenVPN-nl: has no user management component included in the software - access on client/server needed to place the configuration files. When there is physical access to these files, they can be modified and controlled.
- DataDiode: has a user management component built into the web interface of the proxyservers where users and roles can be defined for access to parts of the web interface and services pushed through the DataDiode.
- Sina MS: also has user management implemented in its software to limit access to parts of the system.
- The software runs on a SINA workstation, where a smartcard/token is needed to boot the workstation.
- Skytale: As this product is in development, it has no user management yet. Access on the devices is needed to start shell configuration commands. But user management is wanted to configure the devices and load these settings on a fill-device or access the devices from a centralized management station to change the settings.

**Files, resources & policies**

**Device 'needed' configuration format:**

The format or structure of a configuration that the device is able to use:

- OpenVPN-nl: text file with a certain filename and a flat variable-value structure.
- DataDiode: configuration from user interface is converted to a nested Json format.
- Sina: configuration from management station is written to an encrypted smartcard that is put into the device.
- Skytale: n/a, configuration should be converted to Unix shell commands on devices.

**Collections & metadata**

**Supported input by device:**

An analysis of the configuration structure of OpenVPN-nl devices and DataDiode have been done. We did not look into SINA and Skytale here: Skytale was still under development and not available for testing during this project. As this project is not about replacing the SINA management station, we chose to look at SINA as a general comparison on how it handles its devices as a management station instead of looking into its numerous functionalities' input handling.

For OpenVPN-nl we looked at the textual configuration file, with the DataDiode the user interface has been analyzed.
We made a distinction between basic and complex types and only registered the types we encountered:

<table>
<thead>
<tr>
<th>Basic types</th>
<th>DataDiode</th>
<th>OpenVPN-NL</th>
</tr>
</thead>
<tbody>
<tr>
<td>String</td>
<td>x</td>
<td>x</td>
</tr>
<tr>
<td>Integer</td>
<td>x</td>
<td>x</td>
</tr>
<tr>
<td>Boolean</td>
<td>x</td>
<td>x</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Complex types</th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>value: array of basic types (for selecting/checking)</td>
<td>x</td>
<td>x</td>
</tr>
<tr>
<td>value: list of different basic types (in 1 text input field)*</td>
<td></td>
<td>x</td>
</tr>
<tr>
<td>ip address</td>
<td>x</td>
<td>x</td>
</tr>
<tr>
<td>Subnet</td>
<td>x</td>
<td>x</td>
</tr>
<tr>
<td>email address</td>
<td>x</td>
<td></td>
</tr>
<tr>
<td>filename **</td>
<td></td>
<td>x</td>
</tr>
<tr>
<td>hostname (string)</td>
<td>x</td>
<td></td>
</tr>
<tr>
<td>executable / command</td>
<td>x</td>
<td>x</td>
</tr>
</tbody>
</table>

*e.g., the user is asked for a hostname and port, the actual hostname and port are the users input type and a complex type: it consists of a list of different elements, a string and an integer.

** e.g., the name of a file

**Policies & procedures**

**Configuring devices** (how is the configuration done):

- **OpenVPN-NL**: through 1 text based configuration file, the configuration file and keys are deployed on the client and server. On restart of the software these settings are used.
- **DataDiode**: through a web interface on each proxyserver, input gets directly picked up by the proxyserver after settings are saved.
- **Sina MS**: has its own configuration application, after configuration is done, smartcards or tokens are written, when these are deployed in the device, these settings are used.
- **Skytale**: Skytale doesn't have a management station yet, all configurations are done by starting services from a terminal on the device self- this is not the eventual configuration.

**Connection user management interface to device**:

- **OpenVPN-NL**: n/a
- **DataDiode**: on proxyserver itself -> live update
- **Sina MS**: after placing the smartcard, the device can connect with the trusted network. There is no direct connection to the management station, relevant changes on the management station are sent to an LDAP server, which the device retrieves there.
- Skytale: wish for an external management station that connects with Skytale boxes when possible (n/a yet).

**Other specific user interface functionalities:**

- OpenVPN-nl: n/a
- DataDiode: call specific functions on proxyserver (e.g. view heartbeat, restart device services, ..)
- Sina MS: Blacklist administrators / identities
- Skytale: call specific device functions (e.g. zeroisation of device, view heartbeat)

**Bootstrapping:**

How does the device gets its initial configuration data?

- OpenVPN-nl: keys and configuration need to be put on the client/server.
- DataDiode: needs to be configured by physical access to the proxyserver.
- Sina: configuration data is created by the management station and put on a smartcard or token, this is put into the Sina device.
- Skytale: configuration data is put on a fill device and is connected to the device.

**Abstractions**

**Network - similarities in functionality of network devices**

Except for the DataDiode, all devices encrypt their data on different network layers, after which the data can be transferred over an untrusted network. All devices are placed on the border of the red (trusted) and black (untrusted) network.

**User management**

Not all devices have their own user management components in the software (yet): OpenVPN-nl doesn't have it, Skytale not yet but it is wished. DataDiode and Sina both have user management with user roles implemented.

**Configuration similarities**

- All devices have to be configured individual and manually
- Different sides have to be configured individual and manually
3.3. **Usability analysis: current problems with devices experienced by customers**

To be able to improve the usability of the management station, we distinguished the different users of the current interfaces in user roles. To be able to analyze the problems customers experience when configuring the security devices, interviews with people involved with the future management station have been done.

### 3.3.1. Stakeholders & user roles

We looked at people from the department of Crypto and chose the following stakeholders who are (indirectly) involved with the development of the management station:

- Software developers: all very technical software developers, almost no front end designers/developers. They are now usually also developing the frontend for the devices.
- Support engineers: handle customers who have problems or questions about the devices, but also deploy devices themselves at customer locations, so have to work with the devices too in the complex networks of customers.
- Product managers: make decisions about the development and functionality of the devices.

Other stakeholders are the end-users of the devices who buy the devices. We couldn't interview them for this thesis project, but from interviews with the stakeholders mentioned above and testing the devices ourselves, the following customer roles could be distinguished:

- Technical/System administrator: configures the devices, writes smartcards for end-users, creates other users for the system with less rights.
- End-users: use workstations from locations where an encrypted connection is needed.
- Key manager (crypto administrator): administrates the cryptographic keys belonging to this product, and takes care of transport of the devices.
- Auditor (security officer): has read only access to systems and checks keys and settings of devices.

### 3.3.2. Analysis / interviews outcomes

After interviewing the different stakeholders, we could categorize the different problems customers experience with the current user interfaces as follows:

- **Software not up to date**
  - Users don't have the latest version of software installed but an older version where some problems occur.
- **Users don't understand** the configuration software
  - Users did not read the manual
  - The structure of the screens in the software is not logical
- **Incorrect user input**
  - Incorrect format of input characters (e.g., allow IP-addresses with alphabetic characters)
  - Different fields that are connected contain conflicting input
• **Lack of knowledge on their own network infrastructure**
  o makes users configure the device incorrect with wrong information.

### 3.4. General requirements

We can conclude the analysis done in chapter 3, together with the literature survey in chapter 2 in the general requirements described next. After that they will be further specified by different concepts (which relate to the research questions in chapter 1.2.2) in the next section. These will be implemented in an architectural design (chapter 4) and some of the concepts implemented in a prototype, which will be discussed in chapter 6.

#### 3.4.1. Requirements by Fox-IT and problem statement (chapter 1)

These are the requirements that we used as starting point:

**Non-Functional:**
- Management station should be secure
- Management station should support future new devices
- Management station should be deployed at customer location
- Management station should make the configuration process easier

**Functional:**
- Preferably development in Django / Python, because Python is an often used language at the company and it would make it easier to adapt by other developers.

#### 3.4.2. Requirements by literature survey (chapter 2)

**Security:**
- Define a (CC based) security architecture on the architecture
- Use of CLASP activities during the architecture, design and implementation phase
- Use of a secure software development techniques to rule out at least the OWASP top 10 security risks

**Usability:**
- Use of validation rules for checking of the configuration done by the user

**Genericity/extensibility:**
- Use of middleware to contain all application knowledge, enables interoperability and re-use of software without much effort in the future.
- Use of an MVC framework to maximize clarity, robustness, maintainability and security.
- Possibility to extend the validation methods
- Component-based / modular approach
3.4.3. Requirements by analysis (chapter 3)

Usability:

- The configuration should be checked for user errors before enabling it in the device.
- The management software update process should be easier and faster than it is now.
- It should be possible to give explanations on input fields.
- To increase the apprehensiveness of configuration screens, it should be possible to order items in groups of matching configuration items.
- Validation of basic and complex input fields.
- Validation of connected fields by setting validation rules that match more items.

Security:

- Use of user access and roles for authentication and authorization on the management station, role access should be limited to certain parts or devices in the system.

Genericity/extensibility:

- Management station should support the functionality and input mechanisms from the current devices or have a generic architecture so that functionalities could be included or extended later with minimum effort.

In the next paragraph, we refine these requirements more and specify them into different concepts.
3.5. Functional specification

In this section we translate the general requirements from the previous section in general functionalities that can be implemented in a system. These will be used as input for the architecture described in chapter 4. We group these functionalities in 5 general concepts: improving the configuration process, centralized management of multiple devices, sanity checking, extensibility & genericity and security.

The refinement of the requirements named in the previous section in this functional specification was done due to an iterative process that lasted during the whole research period. Functions started out as ideas or logical steps, based on literature and analysis. Later these ideas were refined by discussions with stakeholders or received feedback after demonstrating parts of the prototype to stakeholders.

3.5.1. Overall concept: Improving the configuration process for the user

Here we describe functionalities that are a possible answer to the problems customers currently experience, which were inventoried during the interviews we did with the support department. We came to these functions by logical reasoning and knowledge and technologies described in chapter 2:

- Manage multiple (different) devices from one user interface. This makes the configuration process faster and easier to understand: the user spends more time with one system, instead of different systems.
- Improving and clarifying the user interface:
  - Create a tabbed structure for the different configuration items: related items can be grouped, different levels of groups are possible.
  - Advanced configuration items can be put away in a different view, so that the user won't be overloaded with settings. Only the critical ones will be shown at first sight.
  - It should be possible to have a default value for a configuration item.
  - Have help text on configuration items
- Sanity checking:
  - Check values of the user for correctness, also in relation to other fields.
  - Give a clear explanation when an input value is not correct.
  - Give the user optionally the possibility to continue, even if the configuration is not sane in the eyes of the system. When this option is chosen, the user has the final say on sanity instead of the system.

The sanity checking part will be further described in 3.5.3, but it is also a part of the improvement to the user configuration process.
3.5.2. Concept 1: Configuring multiple devices centralized

To be able to configure multiple devices through 1 user interface, the following functionalities should be implemented:

Device management:

- New device type modules can be imported in the system, after that instances (devices) can also be configured through the management interface.
- Multiple devices from different device types should be able to be configured through 1 interface.
- Configurations of devices should be validated before being active on the device.
- Multiple configurations can be created, one can be set to active which will be send to the device.
- The device has a module installed that can connect to the management server.

Device type specification:

- The configuration of a device type should be made available in the device specific format, that the device can handle directly. The device doesn't have to rewrite it to its own format.
- A Fox-developer should be able to create a device type specification that can be used in the management station. This consists of:
  o Configuration items with its helptexts, itemgroups and default values
  o Validationrules on the items and related items
  o A specification of how it should be exported to the device
- The device type specification can be exported and imported as a module in the management station by a customer (or pre-installed on the management station).

3.5.3. Concept 2: Sanity checking of device configurations

To be able to improve the correctness of configurations entered by users of the management station, the configurations should be checked for sanity when creating and editing settings:

- Incorrect / impossible input (e.g., invalid ip-address-structure, alpha characters instead of numbers)
- Unsafe / wrong input in combination with other settings.

These can be reduced by implementing validation on 2 different levels:
1. Individual field validation
2. Validation rules on different fields that are dependent on each other. Linked fields can be within the own device type, but also of dependent device types. We call this "global validation".

Both type of validators should be defined by the Fox-developers.
Global validation
The goal of global validation is to check on wrong or strange input, in combination with other settings. The device type-creator (Fox-developer) can define if the rule only gives an explanation why it is incorrect, but can continue anyway, or cannot save it with this input at all.

This can be done by implementing a rule system that works with propositions of user input and implications on other fields.

3.5.4. Concept 3: Extensibility & genericity of the management station
The management station should be set up in a generic way: with device type-modules and extension of the software it can be further extended by Fox-developers so it also suits future new device types.

This has the following benefits:

- Devices don't need to write their own user interface for configuration, only a module for the device type. This saves in total production time.
- The same look & feel for managing all devices by users.

To extend the functionality of the management station, the management station-software can be extended by developers.

Modules: supporting new device types
To be able to manage new 'future' devices, custom device types can be configured by Fox developers on the management station. These device types contain configuration schemes (configuration item groups, items and their valuetypes with their validators), configuration layouts and validation rules.

The device type configuration can be exported by the Fox developers and imported (or by default installed) on the management station web-client by the end-user.

New users & groups
New users and groups with their own permissions can be added later through a user administration section.

Extending the management station software
The software should be relatively easy adaptable in the future by Fox developers.

3.5.5. Concept 4: Security
We further specify the requirement on security in the following functions:

- Implementation of user management:
  - Create, edit and delete users
  - Connect users to groups and groups to user-roles with their own rights in the system
• User authentication & authorization to be able to use functionalities

• Use of an encryption connection to communicate between user/device and management system

• The module that runs on the device needs to authenticate to the management station before it can receive its configuration.

• It should not be possible to retrieve another device’s configuration

• A configuration should only be used by a device if it is sure that it comes from the management station.

• Use of PKI to sign device type specifications and encrypt configurations.

These requirements on the security of the system, possible threats and mitigations are discussed in chapter 5, as they are partially based on the threats of the architecture defined in chapter 4.
4. General design of the management system

We based the architecture of the Fox Management System (FMS) on the different requirements and functionalities defined in the last chapter. After and during designing this architecture, we did a threat analysis and created a security architecture to eliminate threats and attacks on the system as much as possible.

In this chapter we start by describing the architecture. After that we do the threat analysis and describe the security architecture (chapter 5), which contains measures to enhance the security of the system.

The methodology and structure of this chapter is based on a top down approach, as outlined in [33]. We start with the big picture and work our way to more details on implementation. For this section we look at 3 levels: a big picture overview, container view and a major components view with its interactions and workflows. How the concepts are implemented, will be discussed in chapter 6.

4.1. High level design

In this section a high level design, based on requirements will be defined. We start with the general context of the management station and how it fits the current environment. After that we show an overview diagram of the complete system and display the general workflow for configuring devices. We end with an example of the management station, how it could be used with current Fox devices within a network.

4.1.1. General context

The centralized management station can be used in the local network of the customer. It can manage different devices that are available within this local network or connected through a VPN connection from another location.

All configuration information is saved in the database of the management station. After being checked for correctness, a configuration can be sent to the device. The device has to activate it then in the device’s own environment.

To be able to receive and activate this configuration, there is a small program installed on the device. It is only responsible for retrieving the configuration and activating it. Logic on converting a configuration to the device-specific file format is available at the management station.

There are different connections possible: a continuous active connection, which enables a push mechanism of feeding the device with new configurations or managing the device and a pull mechanism, which lets the device contact the management station once in a predefined while.
**Current environment adjustments**

The devices need to have the device-specific module installed before they are able to communicate with the management station. Also the management station has to be implemented within the network.

4.1.2. **Overview of the system**

4.1.3. **General design decisions**

**Centralized solution**

A centralized management station is chosen. It is possible to run the management station on the device itself though. This still has advantages for the developers of the device, they only have to create a device module for the device, instead of writing a completely new user interface.

**Middleware for configuration logic**

We chose to have as much knowledge of the device on the management station, that way the device specific module / application can be very general and installed on different devices. It only has to be configured once to make contact with the management station. When there are updates on the management usage of the device, only the management station has to be updated, not the individual devices.

**Centralized configuration storage**

Configurations are now also stored at a central location, instead of only the devices itself. Configurations on the management station are leading: adjustments made will be transferred to the device. This has implications on the current configuration process of these devices, adjustments made in the configurations on the device itself won't last until the next update from the management station. A choice has to be made to configure it through the management station or traditionally.
4.1.4. General workflow
In the following diagram the general workflow displays the different parties and their main activities. These will be further specified in the lower level architecture descriptions in the next sessions.

![Diagram showing general workflow]

**Figure 6:** general workflow within the management station
4.1.5. Example of FMS with other devices

The following diagram gives an overview of the solution with the current devices:

![Diagram of FMS with other devices](image)

**Figure 7**: example of the FMS within a network with current Fox-devices

The diagram demonstrates the use of the management station (FMS) on all current devices that are connected to a VPN over an insecure (black) network. When they are
connected through their own protocols, the management station is reachable and devices can be configured.

**Alternative setup 1: Data Diode special case**

The networks connected to the Data Diode are separated and only traffic in one direction is allowed. Depending on the way the Data Diode is installed, traffic can go only in or only go out. The most used application is only data in, not out (to prevent data leaks). In this case, both separated networks need their own FMS server to configure their proxy server. This is displayed in Figure 7.

In the other case the Data Diode is installed in a different way: data can go out, but not come in (protecting the integrity instead of the confidentiality). This setting is used in energy or water environments, to give statistic or information for monitoring purposes. In this case it could also be used for pushing the configurations to the devices with one FMS server. Only one management server is needed (Figure 8):

![Diagram showing alternative setup 1 for Data Diode](image)

**Figure 8: Specific application of the FMS for the Data Diode**

**Alternative setup 2: Running the FMS + module on the device itself**

It is also possible to run the FMS server software and the FMS module on the device itself:
In this case the benefits of a centralized management station are not there, but the FMS functionality is available at the device. This setup could be used if there is only one device within the whole network (e.g. the Data Diode black proxy).

4.2. Containers & interactions

In this section the architecture is described on a more detailed level. For this view, we group main components in containers.

In the following diagram, the different containers are displayed with their connections to other main components:

The FMS GUI is the graphical interface from which the user can configure the different connected devices. It is a web application that connects over an HTTPS connection.
(encrypted http traffic through SSL/TLS). To be able to connect with the device, he first logs into the system with his credentials. After configuring, the configuration is available through a webservice (device 2), or pushed to the device directly (device 1), depending on the kind of connection is chosen (PUSH or PULL). The FMS is an MVC application.

The **FMS-module** is the application that runs on the device, it has to be configured after installation on: connection type, FMS server address and authentication credentials.

The **webservice** is used when pull is chosen as connection type, and is based on REST. It is also running on https. The device has to authenticate to the service with its own credentials before it is able to retrieve its configuration.

When push is chosen for the connection to the management server, the device, when online, creates a continues active connection to the FMS. This is also over an https connection, where the device has to authenticate to the FMS.
4.3. Major components & interactions
In this section the different components and their connections of the management station are further specified. Each component embodies functions defined in the last chapter.

4.3.1. Specification of components
In the following UML2 component diagram the different components for the FMS software and device module are displayed:

![Diagram of major components and interactions]

Figure 11: Major components and interactions

The **device type** includes a specification is used by the Fox-developer to specify the device. It includes a specification for the layout of a configuration file that can be activated directly by the device itself, configuration items with their helptexts, default values and groups (for a tabbed structure), validation rules on these items and default connection type.

The **device connector** is the functionality that encapsulates the pull and push connections for the device. It also gives the opportunity to run specific functions on the device. It connects to the **device module**. The pull mechanism is handled by a Restful
Webservice. The **webservice** offers the **configuration** to the device. These are by default in a JSON format, but can also be delivered in the specific device format if this is defined by the Fox-developer.

The end-user views or edits the **device** configuration through the **GUI**, after a change the configuration is checked by the **validator**, after which it is available for the device itself.

### 4.3.2. Workflows

In this section, the configuration of devices through the FMS is described through the perspective of the end-user.

**Installation**

The process of installation of the management system and configuring of devices by a user is as follows:

- The user installs the FMS on a server and the device type specification (if not pre-installed) of the devices he wants to control from the FMS.
- User defines and configures the actual device
- If not pre-installed on a bought device (recommended), the user installs the 'FMS-service' application on the actual device.
- The device module has to be configured by entering the URL of his specific configuration on the FMS server and the device’s authentication credentials, or the server address and credentials when using PUSH.
- FMS makes the configuration available through:
  - A webservice: this is available through a unique URL. Also authentication information for the device is generated. The device connects every X minutes to the specific URL with its authentication data to retrieve its configuration and implements the settings.
  - Or pushes the configuration through an active connection with the device.

**Supporting new device types**

The user can install the FMS with relevant device type specification already in the software implemented, or implement the device type module later. After that this device type with its configuration items, validation rules and configuration layout can be selected and those devices configured.

**Manage & configure devices**

After configuring the device, the configuration is validated by validation rules setup in the device type module.

It is possible to have multiple configurations per device, to be able to deliver an example configuration into a device specification. The one that is set to active is being send to the device.

**Bootstrapping problem**

The following diagram displays the bootstrapping problem that we have with our architecture in VPN situations:
Because we made a design decision that the FMS server is not available from outside the network (LAN or VPN), a new device which is not initialized in the same network as the FMS, is not able to be configured through the FMS: first an initial VPN connection has to be made.

There are 2 solutions for this:

1. Install and configure the device for the first time within the LAN as the FMS
2. Configure the device through an alternative local configuration. Once a VPN connection is created, the FMS can take over and the FMS module will overwrite the original configuration.
5. Security Architecture (target)

In this chapter we define a security architecture for the management station. This is used as part of the validation for the overall security of the product, as mentioned in paragraph 1.3.2 (Approach: validation). The architecture is further validated by an expert in chapter 7 (Validation).

This security architecture is based on the different elements and interactions from the design in chapter 4. We used some of the components of the Common Criteria to describe the architecture (the architecture document is called a Security Target in Common Criteria). With this methodology, we can systematically verify if all identified threats are covered by our security objectives and requirements of the FMS. We don’t go in as much detail as in the usual Common Criteria documents, as we don’t actually evaluate this product now: this is out of scope of this thesis.

For the identification of the threats and possible attacks, we make use of attack trees, assumptions on the environment and create trust zones between different components of the FMS.

Finally we map these found threats with the security objectives and requirements we identified in this chapter, and the ones we identified after the analysis in chapter 3, to see if all threats are covered.

5.1. Security Problem

First we do a security analysis on the environment: assets that need to be protected, data flow between components and threats that arise. Then we can make assumptions about the operational environment. This way we get an overview of the security problem that will be addressed by this security architecture.

5.1.1. Assets

When we look at the assets we want to protect, we can make a distinction between primary and secondary assets: primary assets are the assets themselves, secondary assets are the assets that help protect these primary assets (e.g. cryptographic keys).

When we look at the architecture of the FMS, the following assets can be distinguished:

**Primary assets**
- Configuration data of the devices
- Content of the FMS database
- Device type specification import files

**Secondary assets**
- Authentication data for the FMS GUI and devices (to connect to the webservice)
- Private key files of the FMS and devices (for signing and communicating)
• Physical server and devices
• FMS webservice, GUI, database and device module: give indirect access to the primary assets

**Mapping of the assets against CIAA**
We can map the different assets to what needs to be protected. We look here at protecting the confidentiality, integrity, availability and authenticity of each asset. We don’t take accountability and non-repudiation here, because they are coupled directly to the integrity of the assets.

<table>
<thead>
<tr>
<th>Primary assets</th>
<th>Confidentiality</th>
<th>Integrity</th>
<th>Availability</th>
<th>Authenticity</th>
</tr>
</thead>
<tbody>
<tr>
<td>Configuration data of the devices</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>x</td>
</tr>
<tr>
<td>Content of the FMS database</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td></td>
</tr>
<tr>
<td>Device type specification import files</td>
<td></td>
<td>x</td>
<td></td>
<td>x</td>
</tr>
<tr>
<td><strong>Secondary assets</strong></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Authentication data for the FMS GUI and devices (to connect to to webservice)</td>
<td>x</td>
<td>x</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Private key files of the FMS and devices</td>
<td>x</td>
<td>x</td>
<td></td>
<td></td>
</tr>
<tr>
<td>FMS server application and device module</td>
<td></td>
<td>x</td>
<td></td>
<td>x</td>
</tr>
<tr>
<td>FMS webservice, GUI and database</td>
<td>x</td>
<td>x</td>
<td></td>
<td>x</td>
</tr>
</tbody>
</table>

**Primary assets**

**Configuration data of the devices**

We want to protect the **confidentiality** of the configurations because it can give a potential attacker valuable information (e.g. the network structure, configurations of devices and users of the system), which can be used for an attack. Also the **integrity** must be protected: an altered configuration can lead to harmful (less-secure) settings. It also must be **available** to devices when they use the pull-mechanism. To be able to be sure the configuration is created on the FMS GUI and it is really sent to the device, the **authenticity** of both the device and FMS server must be determined.

**Content of the FMS database**

The content of the FMS database should be **confidential, integer** and **available**. This data is directly related to the settings as viewed from the application, direct access to the database should not be possible.

**Devicetype specification import files**
We want to keep the devicetype specification integer and available, as it is loaded into the FMS and used for all configurations send to the device. Also the authenticity should be verified before modifying the FMS-system with this import file.

Secondary assets

Key material

The authentication data and keys have to be protected for confidentiality and integrity, because they are essential to securing all the other assets.

FMS server application and device module

When the software is installed by the end-user themselves, they have to be sure of the integrity and authenticity: the software is really created by Fox-IT, and not an alternative malicious version.

FMS webservice, GUI and database

The GUI and webservice’s authenticity has to be confirmed: if the device would be authenticating to a malicious version of the webserver, it would give its authentication data and could retrieve potential wrong configuration data. Also the other way around: the webserver should be sure the configuration is send to the real device and not a malicious version that's interested in the configuration data. Next to that, the webserver has to be available and its content should be confidential. The same for the database.

5.1.2. Data flow between components

The following data flow can be distinguished between the main components of the system:

![High level data flow diagram]

Figure 13: High level data flow
5.1.3. Threat analysis

In this section we do a threat analysis on the environment and define preliminaries that an organization needs to satisfy: assumptions on the environment. Then we define trust zones between different locations of the FMS components and see which data is accessible and vulnerable to attacks. After that we look at possible security issues by creating an attack tree that displays threats and vulnerabilities.

Assumptions

The architecture of the FMS gets incorporated in existing operational networks. We need to make certain assumptions on the security and measures taken in this environment to be able to secure the FMS architecture:

- The device/client that a user uses to access the FMS is secure (e.g. virus-free, up to date, runs virus detection and firewall software, etc.)
- Employees that have access to the FMS GUI and devices are trusted, have no malicious intentions and are security aware. Only users that specifically need to have access to the FMS have access to the device.
- *extra measures are taken in the FMS architecture to secure this*
- The FMS and devices are installed and connected properly
- There are SLAs with datacenters to restrict physical access and to manage the FMS server (e.g. updating the used services and operating system). Malicious users shouldn't be able to reach the physical machines (e.g. changing cables or plugging malicious hardware into the device)
- There are different trust zones, based on the different locations of each physical asset:

![Figure 14: Trust zones](image-url)
The user of the FMS system and the devices are only allowed to connect to the FMS from within the LAN or VPN. Because this VPN connection can be set up from another location and is created over the untrusted internet, the location of these devices are different zones. Each zone is individually vulnerable to attacks.

**Attack trees**

Based on the previous sections, we could create the following attack trees [34] with possible attacks on the FMS infrastructure. Because the access to the FMS itself is restricted to only access from LAN/VPN (no internet access), we start with an attack tree for getting on the LAN/VPN:

![Attack tree access LAN/VPN](image)

**Figure 15: Attack tree access LAN/VPN**

In our environment we assume that no malicious users have access to the company LAN or VPN. Because this is not possible to rule out completely, we add this to our attack tree.

When an attacker has access to the LAN or VPN, we distinguish three attack goals and present an attack tree for each of them: tampering with the configurations (**integrity**) that the device receives, attacking the **confidentiality** of the configurations and compromising the **availability** of the FMS:
Figure 16: Attack tree of attacks on the FMS.
5.2. **Security Objectives and Requirements**

After having defined the security problem, we can create security objectives, which are abstract solutions to the different identified threats. We also list our specific security requirements here (which are more specified technical security features of the FMS) as we identified those during the analysis in chapter 3.

All objectives are numbered by an O and a number, these will come back in the mapping of these objectives against threats in the section 'Evaluation' of this chapter.

### 5.2.1. General security

The following general security objects are taken to secure the server and network:

**Server:**

- O1: User and activity logging of the FMS server to be able to do forensics after a possible breach of security.
- O2: A dedicated (virtual) server with own authentication prevents other services or users on the physical machine being able to access the FMS application indirectly.

**Network:**

- O3: IDS software and monitoring of activities within the network to detect abnormalities and to take fast actions in case of a security breach.
- O4: Firewall or whitelisting to limit ip access to the management station or trusted networks.

### 5.2.2. Policies and assumptions

**O5: Security awareness training**

This is a necessity for employees who work (in)directly with information systems and helps to prevent social engineering. People are often the weakest link in the security chain, and security is as strong as the weakest component.

**O6: SLAs with datacenters**

SLAs with datacenters are needed to restrict physical access to devices and service of the FMS.

**O7: Security policies**

Not only implementation measures have to be taken to maximize the security of the system, also policies on rules and regulations have to be made that impact the whole organization.
O8: Roles & least privileges employees

By implementing employee roles and giving these roles least privileges on company systems and resources, chances of an employee having access to a systems he's not allowed to are minimized. This is also based on a risk assessment on the resources.

5.2.3. Methodologies/technologies
O9 and O11: Use of encryption: PKI + SSL/TLS

To be able to mitigate MITM attacks and tampering with configuration files as much as possible, an implementation of Public Key Encryption should be used:

The management station (FMS) gets a certificate, created by Fox·IT. With this unique certificate it can sign any configurations created. The FMS is also a certificate authority (CA) and can give out certificates to connected devices. These will be used for secure communication between these components and signing configurations. Devices will check the configurations first before implementing its settings.

If a device type-module is not pre-installed, this can be handed out by Fox·IT later, and signed with their certificate and checked before being installed.

The certificates can also be used to create an encrypted connection between the different components (SSL/TLS) and ensure their authenticity.

O10: Secure software development

To reduce the chance of implementation security issues, a secure development methodology should be used for the implementation of the FMS (e.g. CLASP).

5.2.4. Specific functionalities in FMS

Here the security functionalities that are specific within the implementation of the FMS are being further discussed.

O12: User password change after first login

After the user has logged in for the first time, he is forced to change his password. This way it reduces the chance of documented login data to be misused. He also gets notified that he shouldn't use a password used elsewhere (this isn't forced or checked though).

O13: Access control on FMS module

Specific login data is needed to be able to modify the settings on the FMS module.

O14: Configuration is stored encrypted and signed by FMS

The configuration data is stored on the device encrypted, so it isn't directly changeable or read by another user on the device. Also it is signed by the FMS to be sure it is generated by the FMS.
O15: Block ip and user after a number of failed logins for X time

When there are too many failed attempts to login to the FMS or FMS module, the ip-address and user are blocked for an amount of time.
5.3. **Evaluation (Security Objectives Rationale)**

In this section we make a mapping of the identified threats and the security objectives and requirements, to see if all identified threats are countered. Within the Common Criteria, this is called a **Security Objectives Rationale**.

The following table gives an overview of the security objectives and requirements (O1 - O15) and how they impact the different kind of threats (related to the numbering of attacks in the attack trees: a-c and 1-10):

<table>
<thead>
<tr>
<th>Access to LAN/VPN</th>
<th>Access to GUI</th>
<th>Access to device</th>
</tr>
</thead>
<tbody>
<tr>
<td>O1 User and activity logging, monitoring</td>
<td>x</td>
<td>x</td>
</tr>
<tr>
<td>O2 Dedicated (virtual) server for FMS application</td>
<td>x</td>
<td>x</td>
</tr>
<tr>
<td>O3 IDS Software</td>
<td>x</td>
<td>x</td>
</tr>
<tr>
<td>O4 Firewall - limit ip access</td>
<td>x</td>
<td>x</td>
</tr>
<tr>
<td><strong>General security</strong></td>
<td></td>
<td></td>
</tr>
<tr>
<td>O5 Security awareness training (assumption)</td>
<td>x</td>
<td>x</td>
</tr>
<tr>
<td>O6 SLAs with datacenters (assumption)</td>
<td>x</td>
<td>x</td>
</tr>
<tr>
<td>O7 Policies on user access on systems company wide</td>
<td>x</td>
<td>x</td>
</tr>
<tr>
<td>O8 Roles &amp; least privileges (assumption)</td>
<td>x</td>
<td>x</td>
</tr>
<tr>
<td><strong>Policies and assumptions</strong></td>
<td></td>
<td></td>
</tr>
<tr>
<td>O9 Use of encryption for all communication</td>
<td>x</td>
<td></td>
</tr>
<tr>
<td>O10 Secure software development</td>
<td>x</td>
<td></td>
</tr>
<tr>
<td>O11 PKI, hashes and signatures for communication</td>
<td>x</td>
<td>x</td>
</tr>
<tr>
<td><strong>Use of principles within technology</strong></td>
<td></td>
<td></td>
</tr>
<tr>
<td>O12 User password has to be changed after first login</td>
<td>x</td>
<td>x</td>
</tr>
<tr>
<td>O13 Access control on FMS module</td>
<td>x</td>
<td>x</td>
</tr>
<tr>
<td>O14 Configuration is stored encrypted and signed by FMS</td>
<td>x</td>
<td>x</td>
</tr>
<tr>
<td>O15 Block ip and user after # failed logins for X time</td>
<td>x</td>
<td></td>
</tr>
</tbody>
</table>

*Figure 17: Mitigations of possible attacks*
We can see that all identified threats are covered by security objectives or are mitigated within the assumptions on the environment. Other remaining issues are discussed in the following section.

5.4. Remaining issues and other possible mitigations
Security is never totally reached, and depending on the effort, there will always be remaining issues. Some remaining issues we did not include in this security architecture because they are out of the scope of this project:

- DDOS attacks on the company network, FMS server or the devices themselves.
- Breaching of SLAs (e.g. unauthorized physical access to machines in an external datacenter).
- Misuse by employees or contracted companies who have access to the FMS.
- Use of specific network attacks like hacking routing devices and changing DNS servers, making MITM attacks possible.
- 0-day attacks or exploits on unpatched workstations
- Code flaws: programming is done by people, and people make mistakes. Even with code reviews, it is still possible that there are flaws or errors in programming code, which might become a security issue.
6. Implementation of the prototype

In this chapter the prototype based on the architecture is described. This prototype will be further used for evaluation in the next chapter.

We start with an overview of the prototype in packages and components, then we discuss the implementation decisions that were made. After that each part of the system is described in its own section.

6.1. Overview FMS implementation

Based on the architecture, we created the following component diagram that we want to implement in the prototype. It describes the different components, packages and relations between them:

![Diagram of components and packages in prototype]

*Figure 18: Components and packages in prototype*

The implementation of these components are further discussed in this chapter.
6.2. General implementation decisions

The technologies used for the implementation of the management station are mostly Django and Python. The FMS-service is a Python application. Also external Python-components have been used, like Tastypie for Django to implement the webservice.

For global validation we used a rule-system that works with pre- and post-conditions. For individual field validation we extended the current validation system of Django.

We implemented only a PULL connection to connect to the management station. Also we did not implement all measures described in the security architecture.

The reasons behind this decisions are described in this section.

6.2.1. Technology used

*Use of Django & Python*

We chose to use this framework and programming language because it is used a lot by developers at Fox-IT. This makes the management station easier adaptable by the developers. Also the way Django applications are made ("only one way to do it") and the wider used MVC pattern, makes it easier for others to understand. Another benefit of using Django is that its framework has some build in measures against possible attacks, e.g. from the OWASP top 10:

- Cross site scripting (XSS) protection
- Cross site request forgery (CSRF) protection
- SQL injection protection
- Clickjacking protection

It reduces the chances on these attacks by implementing some best practices in the Django framework. [35][36]

Django has a steep learning curve before one is able to implement a web-application, so it took time to get to know it and learn how to use it, but for this project it outweighed that with its benefits.

*Use of Tastypie package*

Tastypie is an open source webservice API framework for Django. It is based on the REST principle and includes an object model serializer that supports related objects. This can be used for the export of device types and configuration files in Json-format.

6.2.2. Validation

To set up global validation we chose to use logical rules on items, which can be further defined by propositions and implications. The individual field validation can be done by defining validators through the GUI Afterwards this information is used by the Django specific field validators.
6.2.3. Connection device to webservice
For the prototype, we only implemented the pull-connection with a webservice, because the focus is on the management station and it would take more time to implement the PUSH connection also. This could be extended without too much effort on the current architecture though.

6.2.4. Implementation of security concepts
We chose not to implement all measures defined in the security architecture from chapter 5: some of them are quite extensive and not interesting as part of this research. It would also take too much time to set up a complete environment with all the countermeasures. We focused on the implementation part on the management station software and the specific security functions of the FMS.

6.3. GUI layout
The GUI layout is the part of the web application that is visible to the end-user. This section describes how the application is split up in main parts, and how the URL is built and shown to the user. Furthermore we discuss here a more specific element within the GUI and how it was implemented: the tabbed display of items.

6.3.1. GUI parts
The GUI of the management station is split into 3 main parts:

- A user management part: here new users can be created, edited and passwords restored. Also groups can be managed and permissions to certain parts of the system set.
- A device control part: where all configuration and validation is done.
- Advice type module: new device type modules can be imported by an end-user and Fox-developers can create a device type module and export this.

Figure 19: screenshot of the GUI
6.3.2. URL layout
We tried to use a restful approach when designing the URLs. They are in the form of:
https://serveraddress/module/component/action/nameoftheproperty/
eg.:  https://serveraddress/devices/device type/edit/openvpn-server/

6.3.3. Tabbed items
Based on in which itemgroup items are placed in the device specification, tabs are generated where the configitems and its inputfields are placed in.

![Class diagram of the itemgroups](image)

Figure 20: Class diagram of the itemgroups

The self-reference on the ItemGroup class, makes it possible to have multiple levels of itemgroups, which translates in nested tabs in the GUI.

The following screenshot shows the transformation of itemgroups in a tabbed structure:
Figure 21: Configuration with different itemgroups (tabs) and validation errors


### 6.4. Device and Configuration

In this section we describe how the device configuration is implemented in the prototype. We start with an overview of the workflow for configuring a device by the end-user. After that we discuss how this is implemented by looking at the GUI and at more specific implementation details of functionalities for this part of the system.

#### 6.4.1. Workflow configuring a device

When the end-user wants to configure a device, the following steps are taken before it is send to the device:

![Diagram of the configuration process]

**Figure 22: Workflow of the configuration process**

The implementation of the validation process will be further described in section 6.6. The other parts in the next paragraph.
6.4.2. Implementation of device and configuration management

In this section we further discuss the implementation of the configuration and device management part of the system. We start with looking at the GUI, then review the general structure.

**View from GUI**

When a device is selected, it displays an overview with its current active configuration and values, where only one can put to active:

**Configurations of device**

**Active configuration**

<table>
<thead>
<tr>
<th>Item Key</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>username</td>
<td>3</td>
</tr>
<tr>
<td>protocol</td>
<td>https</td>
</tr>
<tr>
<td>network</td>
<td>FREELAN_SERVER_NETWORK</td>
</tr>
<tr>
<td>disable_host_verification</td>
<td>no</td>
</tr>
<tr>
<td>disable_peer_verification</td>
<td>no</td>
</tr>
<tr>
<td>enabled</td>
<td>no</td>
</tr>
<tr>
<td>https_proxy</td>
<td>2</td>
</tr>
<tr>
<td>host</td>
<td>1</td>
</tr>
<tr>
<td>public_endpoint</td>
<td>0.0.0.0 ::</td>
</tr>
<tr>
<td>password</td>
<td>4</td>
</tr>
</tbody>
</table>

**View active configuration**

View JSON configuration through webservice

View configuration as device config (download)

View configuration as device config (in window)

**All configurations**

<table>
<thead>
<tr>
<th>Config name</th>
<th>Active</th>
<th>Remove config</th>
</tr>
</thead>
<tbody>
<tr>
<td>config</td>
<td>True</td>
<td>x</td>
</tr>
</tbody>
</table>

**Figure 23: Overview of the device**

From this view the configuration can also be edited, new configurations added or view the configuration as JSON or device-specific layout.
**General structure**
The following class diagram displays each class, its attributes and relations:

![Class diagram of the device and config structure](image)

*Figure 24: class diagram of the device and config structure*
6.5. **Devicetype**

In this section about the devicetype component, we discuss the implementation of the devicetype specification. This specification Fox-developers can use to describe their security product, so that it can be configured through the management station software.

We first look at the general functions within this process, then look at the GUI on how this actually looks and after that we will go into more implementation details on how this was built. Here we also look at how the fms module and webservice are implemented.

6.5.1. **General functions**

In the following use case diagram the functions of the devicetype component is displayed:

![Use case diagram of the devicetype component](image)

**Figure 25: use case diagram of the devicetype component**

The Fox-developer designs the devicetype specification. It contains a configuration schema: configuration items, their itemgroups and validators and a configuration layout: middleware containing the knowledge for converting the configuration data by the end-user in a device specific format. When this is defined, the specification can be exported as
a json file. This specification can be included in the management software, or later imported manually by the end-user of the software and security device. After installing, the device is ready to be configured through the management station.

6.5.2. Device type specification from GUI

The devicetype specification consists out of a configuration schema, a configuration layout and global validation rules. The first two are described as viewed from the GUI in this section. The global validation rules will be further discussed in the next section.

**Configuration schema**

In the configuration schema the different configuration items and their specifics are defined:

![Screenshot for defining the device type's configuration schema](image)

Figure 26: Screenshot for defining the device type's configuration schema

Different items with their itemgroups (that will be converted in tabs for the end-user), valuetypes and individual validators can be defined here.
Figure 27: Screenshot for adding a new configuration item with its variables
**Configuration layout**

We looked at how current configuration files from all the devices are built, and extracted the following configuration specifics. With these specifics a device own configuration file can be build.

- **Default filename:**
  The default name of the exported configfile, for example: config.conf

- **Define character:**
  The define character is the character between the key and value in the configuration file readable by the device, for example: `=`

- **Comment character:**
  The comment character is the character that disables a key and value from the configuration file, for example: `#`

- **Delimiter character:**
  The delimiter character is the character between multiple values, for example: `,`

- **Behaviour empty values:**
  When a user doesn’t enter a value, what should be put in the configuration file?

- **Behaviour boolean values:**
  When a user enters a FALSE, what should be put in the configuration file?

- **Behaviour multiple keys:**
  When there's a double configuration key in a user configuration, how to handle the entered values in the configfile?

- **Device type:**

---

**Figure 28: Screenshot of the 'configuration layout' screen**
6.5.3. Implementation of devicetype specification

The following diagram displays the different classes and attributes that support the device type specification. We left the global validation rules out of this overview, they belong here but will be discussed in section 6.6.

![Class Diagram](image)

**Figure 29**: class diagram of the devicetype specification

The ConfigLayout class has attributes that refer to static variables, these are as follows:

```python
BEHAVIOUR_EMPTY_VALUES = (
    ('COMMENT', 'comment the key + default value'),
    ('LEAVE_KEY', 'leave the key out of the config'),
    ('ENTER_KEY', 'only enter the key'),
    ('ERROR', 'error message'),
)

BEHAVIOUR_BOOLEAN_VALUES = (
    ('ENTER_VALUE', 'Enter TRUE/FALSE'),
    ('COMMENT_KEY', 'comment the key+default value'),
)

BEHAVIOUR_MULTIPLE_KEYS = (
    ('MERGE_VALUES', 'samenvoegen van de values'),
    ('DOUBLE_KEYS', 'new line with same key and value'),
    ('ERROR', 'error message'),
)
```
6.5.4. Implementation of the device connector and module

The device module is a small Python application that connects to the webservice and retrieves its configuration file. It first connects to the server-URL on https, and authenticates then with its device-credentials.

For the webservice an extension of Django is used: Django Tastypie3. It is a RESTful webservice API framework for Django. It can serialize configuration objects to the configuration in the specified layout, this is by default in json.

6.6. Validation

The validation part is a large part of the system. We distinguished between individual and global validation of fields. They both can be defined within the devicetype specification section of the management station.

The individual field validation of configuration items, where only the value of one individual item is checked. Global validation rules can be defined that validate values of items in combination with other item-values.

In this section we will further describe how we implemented this and what choices were made.

6.6.1. Workflow validation

After a user has configured a device in the webclient, the configuration file is validated by the rules set up. First the individual fields are checked for correctness, after that the relations between the fields are checked for inconsistency. When errors occur, they are being displayed in the screen.

This is implemented as follows: after Submitting, first the individual item-validators are checked, after that the global validation rules. Because some of the global validation rules are allowed to be broken, there is a confirmation asked of the user before saving the configuration.

---

3http://tastypieapi.org/
Figure 30: validation process
6.6.2. Defining individual validations from the GUI
For the prototype we created a simple individual field-validator with 3 options. In a real version these should not be all in one form, but one type of a validator should be selected before filling in. This is more clear to the user and will exclude the chance of inconsistent input.

![Validation of config item: username (STRING)](image)

Figure 31: Screenshot for adding an individual validation to a configuration item

In the example in Figure 31 a regular expression is entered. When the end-user will enter a username, it will be checked when the form is submitted.

6.6.3. Defining global validation rules from the GUI
The global validation rules were implemented as follows:

First a name of the rule should be entered. When ‘tolerate breaking’ is checked, the rule can be broken by the user, but it will give a warning message. Otherwise, a configuration cannot be saved when the rule is broken.

In the next screen, multiple propositions and implications of the rule can be added to the rule.
Create rules

Figure 32: screenshot of step 1: add a validation rule

Here also the option for the possibility to break the rule by the user can be chosen.

After that propositions and implications on this rule have to be defined. There can be multiple propositions and implications, the propositions can be coupled by the connectives AND or OR.

Add propositions (if)

A configuration item have to be selected, the kind of value entered and optional a value. If it is not the first rule, a connective has to be entered as well (AND, OR)

Figure 33: adding propositions

Add implication (then)

Here the implications of the rule are defined. The fields are the same as for propositions, except for actions: a choice can be made to delete other fields from the form, and any entered value won't be saved.

If it is not the first implication, a connective has to be entered as well (AND, OR)
6.6.4. Implementation in Django

Django works by default with automatically created forms based on the attributes of the model. Our fields and validators are variable, defined by the Fox-developer. Therefore we couldn't use the Django Forms directly and had to build and override the form and validate methods within Django, to make it possible to work with our user-defined items and validators.

We created an extended forms with all the wanted items by overriding Django's ModelForm __init__ method and adding the needed fields with their default values, helptexts etc.

To be able to use the individual validators and checks for a correct valuetype, we could override the __clean__ method and run the additional checks. A method that runs the global validation rules is also fired from this method. If any problems occur, Django's ValidationErrors is raised and errors are displayed to the user.
6.7. **Authentication and user management**

We did not implement all authentication features we mentioned in the general architecture and security architecture from the last chapters in the prototype: we chose to not implement the PKI infrastructure, because it relies on external components that are not really interesting for this research. The other features we did implement will be discussed in this section.

6.7.1. **Authentication to the GUI**

For users and authentication we used the Django user authentication package. With this package user groups can be created that have certain permissions, after that users can be put into the groups. It has built in functions for user login and authentication which could be used within code of the management system.

**Permissions**

When a user is logged in, permissions can be checked before executing specific functions within the management system. This is done on different levels: on the view and the template level.

Example of a permission check on a view function:

```python
@permission_required('auth.add_user')
def DeviceOverview(request):
    devices = Device.objects.all()
    ...
```

Example of permission check in the template, if the user hasn’t access, certain links won’t be displayed. The views behind these URLs are also restricted.

```html
{% if perms.auth.add_user %}
<h3>Users</h3>
<a href="/users/">user overview</a><br/>
<a href="/users/groups/">group overview</a><br/>
{% endif %}
```
User groups

We created 2 different user groups for this prototype: a ‘technical administrator’ which can handle devices and configurations and an ‘fms admin’, who handles the users and roles on this system. The groups with specific rights can be extended through the Django admin interface by a ‘technical administrator’.

![Add group](image)

Figure 35: extending functionality - adding new groups with new rights

6.7.3. Use of encryption in communication

The only use of encryption we used within this prototype, is deploying the web application on a https webservice.
7. Validating the FMS

In this chapter we validate the design and implementation of the management system if it really is a solution to our research problem.

To be able to do this, we validate it by seeking answers to the following questions:

1. Is the architecture of this centralized management system secure (enough) to use and extend further by Fox-IT?
2. Is the prototype generic and extensible enough to support future possible systems without having to modify the system much?
3. Are the configurations done by users on this management station of a better quality than before they were using this configuration method?

We answer the first questions by looking at the security architecture of the FMS for meeting its security objectives and having a security expert at Fox-IT judge this architecture for completeness.

The second question will be answered by using a beforehand unknown open source VPN product that we will try to configure through the management software prototype.

The third question we will try to answer through logical reasoning.
7.1. Validating security

We validate the security of the management station by looking at the design and security architecture. We did not implement all security aspects in this design, so validation is about the design and architecture only.

A system cannot be validated for being completely ‘secure’. It can only be validated on actions that are taken to reduce the chances of an attack as much as possible. We validate the security of the design by looking at the different threats and possible attacks on that architecture and what measures are taken to reduce the risk of these threats happening. Figure 17: Mitigations of possible attacks in section 5.3 displays the identified threats and mitigations for the management station. When these mitigations are implemented correctly, we can say that the chances of the identified threats and attacks happening are smaller than when they aren't implemented.

We also had a security expert at Fox-IT check the security architecture of the management station. He had no remarks on the identified threats and countermeasures. Further he advised on how to create a security architecture document, as we had no examples of this (because of the sensitivity of these documents). He also made some remarks on improving the analysis and assumptions part of the security architecture.

We processed most of these remarks.

For the prototype, we used the Django framework which by default has mechanisms build in against different often used web attacks (like most of the OWASP top 10 attacks). We did not further test the prototype on security though, as it was meant to demonstrate the other aspects of this thesis project.

7.2. Validating extensibility

To validate the extensibility of the management software, we used open source VPN software (Freelan) and tried to manage it through our management station. We did not have knowledge of this application in the design and implementation phase of the management system. The test is described in Appendix 2.

The software could be managed within the FMS without software or design adaption of the FMS. We cannot in general claim that this makes the FMS extensible for all VPN software, but it is extensible for this specific beforehand unknown case.

Genericity is always limited to a certain level. For the FMS this is based on a limited amount of security devices. It is possible to extend the functionality of the FMS, but in a limited way. In this case, we can say that the management station is extensible and generic for devices that can run the device module which can control the device specific functionality from there. The module also has to be able to connect to the management station through a local network or VPN, use https and be able to retrieve data.

Also the prototype is set up through Django’s structure, and should be able to understand and adapt further by other Django developers with relative ease.
7.3. Validating new configurations

Are the configurations done by users on this management station of a better quality than before they were using this configuration method?

To validate if the new configuration process is better than the current one, we first look at how the management station impacts the individual devices, and then how it impacts a network with different devices. We look at the OpenVPN·nl and DataDiode case.

For OpenVPN·nl, when the imported device type specification with configuration items, with their item groups, default values, helptexts and validation rules are defined correct by the Fox-developers, the configurations created by the management station are of a better quality: faults can be detected before the configuration is sent to the device.

As for the DataDiode proxy servers, they already had a graphical user interface with some validation checks. The possibility to configure multiple proxies through 1 management station is not an option for this either, as it would compensate the security feature of the DataDiode itself: 2 networks would not be completely separated by the DataDiode anymore (but also by the management station). It has improvements for the inter-relations between different fields and the possibility for future versions of the DataDiode to use this management station instead of creating or supporting a new graphical user interface. It can be installed locally on a proxyserver, but also on another server within the same network.

When there are different Fox-IT devices on a single network (e.g. a DataDiode black proxyserver and OpenVPN devices), the configurations can compare values with each other through the use of global validation rules. In this case, rules can set up to detect conflicting values.

So we can say that the quality of the configurations at least improves when a device type specification is created and global validation rules are used. It gets checked before it is sent to the device itself, so the quality of the configuration is higher when it gets picked up by the device then before.
8. Conclusion

This thesis project is a design project for the Master Information Architecture. We needed to create an architecture and prototype of a centralized solution for a management station for Fox-IT's security products. It had to be secure, extensible and should improve the configuration process of these devices by their customers.

We came up with requirements and an architecture by looking at related literature and analyzing the environment where Fox-IT operates. The architecture was partly implemented in a prototype that focused on the configuration process and extensibility of the management station. The security aspects from the architecture were not completely implemented, as Fox-IT already has knowledge of this and implementing all aspects would take too much time.

The security of the management station was validated by creating a security architecture that covered various threats. The validity of this security architecture was checked by a security expert of Fox-IT. The prototype was validated on extensibility and configuration quality: the extensibility was tested by extending it on another open source VPN product, the configuration quality by logical reasoning on the quality of the configuration that gets sent to the device.

Reflections & limitations on this research

The intuitiveness of the management station could have been better validated if the product would be tested on actual end-users of products by Fox-IT. This is also the case for development iterations of the prototype in general. Because of the nature and extensiveness of the products, it wasn't possible to connect the prototype to a 'real' Fox-IT product that could be compared by a customer.

The writing process lasted a long time and even though Fox-IT had my research material and non-final thesis for some time, I feel that this finished version of the thesis might be currently outdated for the company.

During this research project I've learned how to handle a project more methodological. Also being able to view the professional, high-standard development process of high secure products at Fox-IT was very interesting and useful.
9. Recommendations

Even though the architecture and prototype of the management station answer the research question, we have recommendations on the management station that would improve its use. These are described in this chapter.

9.1. Extra functionality for the management station

These are functionalities that were partly beyond the scope of the thesis project, but would improve the FMS much if implemented:

- **Security**:
  - Use of **logging** to improve the Accountability of the system.
  - Use of **PKI**: It would be a good thing if there was a certificate authority module available to the FMS. When configuring devices, they need to have certificates.
- The ability to create **backups** of the configurations and settings of the whole management system and its devices
- Device **group settings**: all devices within groups have these settings
- **Global variables**: variables that can be entered within the configurations (e.g. %FMS_SERVER_IP)
- Databank / **FAQ** with common questions / problems that customers can look into.
- **Monitoring** functionality to view statuses of connected devices: e.g.: are their configurations received and implemented? Is it online?
- A template system: load default settings to devices
- An **auditor role** where settings can be viewed but not changed or crypto managers that can import keys in the system
  - The possibility to have an **auditor compare** its settings with templates to review security settings of a device

9.2. Improvements

The following points discuss room for improvement:

- A problem with a centralized solution is when a device is **misconfigured**, it might not be possible to **reconnect** to the network and so cannot be managed anymore. The system administrator has to reconfigure the device locally again, and disconnect it from the management station until the settings are right again. An improvement would be to implement a backup safety procedure (e.g. restore an older configuration file if it cannot connect with the new one).
- Configurations are now written to a **single** text based configuration **file**. It is possible that other devices would have more than one file with settings. It could bring new security risks when files can be written from the management station to the device though.
- **Bootstrapping**: make the bootstrapping process more easy: automatically configure device on first connection.
The implementation of 'global validation', where items can be checked in relation to another, only supports validation of fields within the same devicetype. A way to improve this is to have devicetype modules check for other devicetype modules when importing, and only importing global validation rules that impact both devicetypes.

9.3. Other recommendations
A quite new field where we did not look further into is model-based security testing. With this methodology a security model is created where the system is developed onto next. Security tests are generated automatically. It might be interesting for Fox-IT to research how this methodology could fit in its current development processes.
Bibliography


"Extensibility," wikipedia, [Online]. Available:


## Appendix 1: Vocabulary used

<table>
<thead>
<tr>
<th>Term</th>
<th>Definition</th>
</tr>
</thead>
<tbody>
<tr>
<td>API</td>
<td>Application Programming Interface</td>
</tr>
<tr>
<td>Back end</td>
<td>Restricted part of an application where end-users aren't allowed to. Usually used for configuration.</td>
</tr>
<tr>
<td>Bootstrapping</td>
<td>Initial loading of an application or a system</td>
</tr>
<tr>
<td>Configuration</td>
<td>Specific settings for a system</td>
</tr>
<tr>
<td>Data cleansing &amp; data enrichment</td>
<td>Improve the quality of data, usually automatically.</td>
</tr>
<tr>
<td>DDOS attack</td>
<td>Distributed Denial of Service: attack where a service can't handle all of the incoming malicious requests</td>
</tr>
<tr>
<td>Django</td>
<td>MVC framework for Python</td>
</tr>
<tr>
<td>Extensibility (software)</td>
<td>Programming principle that takes future growth into account</td>
</tr>
<tr>
<td>Fill device</td>
<td>A fill device is a form of key transfer device, that is used (most commonly by the military) for the distribution of cryptographic variables, such as crypto keys and frequency hopping tables</td>
</tr>
<tr>
<td>Firewall</td>
<td>System that shields another system or network from a potential harmful environment</td>
</tr>
<tr>
<td>Framework (software)</td>
<td>Platform with build-in functionality to reuse certain code more easily and build applications faster.</td>
</tr>
<tr>
<td>Freelan</td>
<td>Open source VPN software</td>
</tr>
<tr>
<td>Front-end</td>
<td>Part of an application that is accessible to end-users</td>
</tr>
<tr>
<td>Generic / genericity</td>
<td>General specification of functionality that can be specified later.</td>
</tr>
<tr>
<td>Global variables</td>
<td>Variable that is accessible throughout a programming code</td>
</tr>
<tr>
<td>GUI</td>
<td>Graphical User Interface</td>
</tr>
<tr>
<td>https</td>
<td>Encrypted protocol version of http</td>
</tr>
<tr>
<td>IDS</td>
<td>Intrusion Detection System</td>
</tr>
<tr>
<td>interoperability</td>
<td>Ability to have multiple systems work together.</td>
</tr>
<tr>
<td>JSON</td>
<td>Serialization standard that is currently popular to transfer data between different data services.</td>
</tr>
<tr>
<td>MITM attack</td>
<td>Man In The Middle attack: an attack where an outsider pretends to be the source and destination, intercepting and transferring all data.</td>
</tr>
<tr>
<td>MVC</td>
<td>Model View Controller: A current popular methodology to develop software where an abstraction is made to decouple application logic, structure and presentation of data.</td>
</tr>
<tr>
<td>NBV</td>
<td>Nationaal Bureau voor Veiligheidsverbinding</td>
</tr>
<tr>
<td>NLNCSA</td>
<td>English term for the (NBV) - an institute that helps the dutch government with the security of their data and infrastructure.</td>
</tr>
<tr>
<td>PKI</td>
<td>Public Key Infrastructure</td>
</tr>
<tr>
<td>Proof of Concept (POC)</td>
<td>Demonstration that a certain idea is feasible.</td>
</tr>
<tr>
<td>Pull connection</td>
<td>Data from a source is being requested for information by its requester in a timely interval</td>
</tr>
<tr>
<td>Push connection</td>
<td>Data from a source is being transferred automatically to its destination when data is updated.</td>
</tr>
<tr>
<td>Python</td>
<td>Programming language</td>
</tr>
<tr>
<td>REST / RESTFUL webservice</td>
<td>A software architecture for distributed systems, like the web.</td>
</tr>
<tr>
<td>Router</td>
<td>Device that connects systems or networks</td>
</tr>
<tr>
<td>SSL/TLS</td>
<td>Secure Socket Layer/Transport Layer Security: encryption protocols</td>
</tr>
<tr>
<td>Tastypie</td>
<td>A restful webservice API for Django</td>
</tr>
<tr>
<td>Term</td>
<td>Definition</td>
</tr>
<tr>
<td>------------------</td>
<td>-----------------------------------------------------------------------------</td>
</tr>
<tr>
<td>TCP/IP</td>
<td>Data transfer protocol</td>
</tr>
<tr>
<td>Unix / linux</td>
<td>Operating system, used often for servers</td>
</tr>
<tr>
<td>Variable (software)</td>
<td>A symbolic name in the memory of a program with a value that can be changed. Used for referencing in programming code.</td>
</tr>
<tr>
<td>VPN</td>
<td>Virtual Private Network: A method to connect a system or network to another physical network elsewhere over an unsecure public network.</td>
</tr>
<tr>
<td>Webservice</td>
<td>A service that’s available through the web, used to communicate and transfer data between systems.</td>
</tr>
</tbody>
</table>
Appendix 2: Extensibility test FMS

Validating extensibility and genericity
We looked for open source software that we could use to test the extensibility of our management system. We looked specifically for software whose main function is the encryption of data for secure communication, which exists on the border of the network (just like the systems we used as starting point for this system.

We looked specifically at software based communication encryption, because these are the only variants which could be used for free. We found the following 2 possible candidates:

- Freelan [http://www.freelan.org/]: VPN software without a graphical interface. Configurations are created and stored in one plain text file. It has a service that needs to be started, after that the configuration file is loaded and a connection is created. The software is available for Windows and Linux.

We chose to use Freelan, because it was better documented.

Freelan configuration through FMS
To be able to manage the Freelan client through the FMS, we did the following:

1. We installed Freelan on a Windows machine and created certificates for it.
2. We created in the FMS a new device type: Freelan:

Add devicetype:

Name: freelan [Next step - configuration items]
3. We defined the needed items (we used the sample configuration file as input). We did not define help texts and validation rules to help the user, because this is to demonstrate that the client can be supported by the FMS. It is possible to define the other items too though. We did use the tabbed layout, as there is some sort of distinction made in the configuration files too.

![Figure 38: One of the needed items being added](image-url)
4. After that the layout of the configuration file could be defined, so the FMS knows how to generate the file:

**Figure 39: List of items**

**Figure 40: Defining the layout of the config file**
5. After that the device could be configured by an end-user:

![Configuration Screen]

**Figure 41: Part of the generated configuration screen**

6. Extra to the normal configuration, authentication data has to be defined for the device to authenticate to the webserver.

7. To be able to retrieve the configuration done by the end-user, the device has to install (if not pre-installed) and configure the FMS module on the device. This has to be done once. The module contains:
   a. Choice of mechanism used to retrieve the config (PUSH/PULL)
   b. a URL where the configuration can be found on the FMS
   c. the credentials to authenticate
   d. the directory where the config file should be put in.
   e. the os-command how to restart the specific of the application (in this case freelan).

8. After that the module can be (auto)started and connect to the FMS to retrieve its configuration. It gets written to the dir and the service restarted with the new config.
Problems run into

We did run into some problems which are not directly related to the extensibility, but should be mentioned:

- We did not implement a functionality in the management station that creates certificates as a certificate authority, so it can create certificates that can be used for encryption of the communication (SSL/TLS). These certificates are stored locally on the device and are referenced from the configuration file. When configuring a device through the FMS, one of its items is the name and location of the certificate file on the device itself. You have to know this specific information on the FMS to be able to make it connect to the VPN network. This is not handy. A solution for this problem would be to have the FMS also create and distribute these certificates and send them with the configuration file to the device.

- Another problem we ran into was the bootstrapping problem. If the FMS is only accessible within the local network (which is the safest option), a device has to be configured for the first time within this network so it can access the FMS server. When it has received its configuration, it can first create the connection through the VPN and access the FMS later on too.

We also mention these points in the chapter 8 - Conclusion, as recommendations for improvement.