Institutionen för datavetenskap
Department of Computer and Information Science

Final thesis

Security concerns regarding connected embedded systems

by

Jon Mårdsjö

LIU-IDA/LITH-EX-A--13/011—SE

2013-02-28
Final Thesis

Security concerns regarding connected embedded systems

by

Jon Mårdsjö

LIU-IDA/LITH-EX-A--13/011—SE

2013-02-28

Supervisor: Anna Vapen
Examiner: Nahid Shahmehri
Copyright

The publishers will keep this document online on the Internet – or its possible replacement – for a period of 25 years starting from the date of publication barring exceptional circumstances.

The online availability of the document implies permanent permission for anyone to read, to download, or to print out single copies for his/hers own use and to use it unchanged for non-commercial research and educational purpose. Subsequent transfers of copyright cannot revoke this permission. All other uses of the document are conditional upon the consent of the copyright owner. The publisher has taken technical and administrative measures to assure authenticity, security and accessibility.

According to intellectual property law the author has the right to be mentioned when his/her work is accessed as described above and to be protected against infringement.

For additional information about the Linköping University Electronic Press and its procedures for publication and for assurance of document integrity, please refer to its www home page: http://www.ep.liu.se/.

© Jon Mårdsjö
Abstract

Embedded systems have been present in our daily lives for some time, but trends clearly show a rise in inter-connectivity in such devices. This presents promising new applications and possibilities, but also opens up a lot attack surface. Our goal in this thesis is to find out how you can develop such interconnected embedded systems in a way that guarantees the three major components of information security: Confidentiality, Integrity and Availability. The main focus of security is networked security. In this thesis, a dual approach is taken: investigate the development process of building secure systems, and perform such an implementation. The artifacts produced as byproducts, the software itself, deployment instructions and lessons learned are all presented. It is shown that the process used helps businesses find a somewhat deterministic approach to security, have a higher level of confidence, helps justify the costs that security work entails and helps in seeing security as a business decision. Embedded systems were also shown to present unforeseen obstacles, such as how the lack of a motherboard battery clashes with X.509. In the end, a discussion is made about how far the system can guarantee information security, what problems still exist and what could be done to mitigate them.

Keywords: Embedded security, network security, risk management, untrusted domains, interconnected systems, Internet of Things (IoT).
Acknowledgement

I’d like to take the opportunity to thank everyone who supported me in this thesis work. At the office at Semcon a lot of people have helped with insight, support, understanding and patience. They are too many to mention so I’ll just name a few. My supervisor at Semcon, Johan Sandgren. My manager at semcon, Johan Kristensson. My surrounding co-workers; Bo, Dinesh, Jonas and Christoffer to name a few.

From the university side, my supervisor Anna Vapen has been a huge help in writing this. Also my examiner Nahid Shahmehri for providing insight and direction along the way.

I’d also like to thank my sister Alva Mårdsjö for graciously helping me with accommodation in Göteborg (not to mention having patience with my morning mood!), and to my parents for unending long-distance support.
Table of Contents

Acknowledgement ........................................................................................................ vi

1. Introduction ............................................................................................................... 14
   1.1. Background ........................................................................................................ 14
   1.2. Questions .......................................................................................................... 14
   1.3. Scenario ............................................................................................................ 14
   1.4. Scope ................................................................................................................ 15

2. Theoretical framework ............................................................................................... 16
   2.1. Sources ............................................................................................................. 16
   2.2. Cryptography .................................................................................................... 16
       2.2.1. Cryptographic hashes .............................................................................. 16
       2.2.2. Strength .................................................................................................... 17
   2.3. Certificates ....................................................................................................... 17
       2.3.1. Chain of Trust ......................................................................................... 17
   2.4. Authentication ................................................................................................. 17
       2.4.1. Basics ....................................................................................................... 17
       2.4.2. The process of authentication ................................................................. 18
       2.4.3. Challenge-Response ................................................................................ 19
   2.5. Risk Management ............................................................................................. 20
       2.5.1. Artifact ..................................................................................................... 20
       2.5.2. Activity ..................................................................................................... 20
       2.5.3. Misuse cases ............................................................................................. 20
       2.5.4. Security requirements .............................................................................. 21
       2.5.5. Risk analysis ............................................................................................ 21
       2.5.6. Static analysis ............................................................................................ 22
       2.5.7. Attack trees ............................................................................................... 22
2.6. Annual Loss Expectancy ................................................................. 22
2.7. Risk Management Framework ....................................................... 23
  2.7.1. Understand the Business Context ............................................... 23
  2.7.2. Identify and link the Business and Technical Risks ........................ 24
  2.7.3. Synthesize and Rank the Risks .................................................. 24
  2.7.4. Define a Risk Mitigation Strategy .............................................. 24
  2.7.5. Fix the Problems and Validate the Fixes ..................................... 24
3. Method ............................................................................................... 25
  3.1. Overview ....................................................................................... 25
  3.2. Scenario ....................................................................................... 25
  3.3. Risk management framework ....................................................... 25
  3.4. Waterfall ..................................................................................... 26
  3.5. Combining RMF with Waterfall .................................................... 26
4. Planning .............................................................................................. 27
  4.1. Scenario ....................................................................................... 27
    4.1.1. System overview .................................................................. 27
    4.1.2. Business Context ................................................................. 28
  4.2. System usage ............................................................................... 28
  4.3. Process .......................................................................................... 30
  4.4. Technological survey .................................................................... 31
    4.4.1. Link layer ............................................................................. 32
  4.5. Hardware platform ....................................................................... 37
  4.6. Presentation layer ......................................................................... 37
5. Risk management ............................................................................... 37
  5.1. Assets .......................................................................................... 37
  5.2. Security requirements ................................................................... 39
5.3. Attack trees ........................................................................................................... 40
5.4. Security risks .......................................................................................................... 42
  5.4.1. Registering a fake backend .................................................................................. 42
  5.4.2. Registering a fake welder ..................................................................................... 43
  5.4.3. Altering configuration or software on gateway or leaf physically ....................... 43
  5.4.4. Eavesdropping or performing Man-in-the-Middle on a link ............................... 43
  5.4.5. Altering welder data on a gateway or leaf physically ......................................... 43
  5.4.6. Reading welder data from a gateway or leaf physically ...................................... 44
  5.4.7. Reading welder data from a backend ................................................................. 44
  5.4.8. Disrupting the delivery of alarms ....................................................................... 44
  5.4.9. Summary ............................................................................................................... 44
5.5. Mitigations .................................................................................................................. 45
  5.5.1. Reg. fake backend, #1 ......................................................................................... 45
  5.5.2. Eavesdropping/MitM, #4 ..................................................................................... 45
  5.5.3. Reading welder data from backend, #7 .............................................................. 46
  5.5.4. Altering configuration or software on gateway or leaf physically, #3 .............. 46
  5.5.5. Disrupting the delivery of alarms, #8 ................................................................. 46
  5.5.6. Summary ............................................................................................................... 47
5.6. Architecture alterations ............................................................................................ 47
  5.6.1. Certificates ........................................................................................................... 47
  5.6.2. Architectural requirements ................................................................................... 48
  5.6.3. Physical requirements .......................................................................................... 48
6. Implementation .............................................................................................................. 49
  6.1. General implementation layout ................................................................................ 49
  6.2. XMPGate .................................................................................................................. 50
  6.3. XMPLeaf .................................................................................................................. 50
6.3.1. Welder model ........................................................................................................... 50
6.4. XMPBack ..................................................................................................................... 51
  6.4.1. Generated graphs ................................................................................................. 51
6.5. Key scheme ............................................................................................................... 54
6.6. Cryptography ............................................................................................................. 55
  6.6.1. Signing .................................................................................................................. 56
  6.6.2. Encryption ............................................................................................................ 56
6.7. File layout ................................................................................................................. 57

7. Deployment .................................................................................................................... 58
  7.1. Deployment issues .................................................................................................... 58
    7.1.1. Memory management ....................................................................................... 58
    7.1.2. Mitigations ......................................................................................................... 59
  7.2. Configuration management ...................................................................................... 60
  7.3. Signing code ............................................................................................................ 61
  7.4. Startup .................................................................................................................... 61
  7.5. Time management ................................................................................................... 61

8. Risk management, round two ....................................................................................... 62
  8.1. Identify and link the business and technical risks .................................................... 62
  8.2. Synthesize and rank the risks ................................................................................ 64
    8.2.1. Registering a fake backend ............................................................................. 64
    8.2.2. Registering a fake welder ............................................................................... 65
    8.2.3. Altering configuration or software on gateway or leaf physically .................... 65
    8.2.4. Eavesdropping or performing Man-in-the-Middle on a link ............................. 65
    8.2.5. Altering welder data on a gateway or leaf physically ........................................ 66
    8.2.6. Reading welder data from a gateway or leaf physically ..................................... 66
    8.2.7. Reading welder data from a backend ............................................................... 66
8.2.8. Disrupting the delivery of alarms/status updates/data ................. 67

8.2.9. Denial of service ................................................................. 67

8.2.10. Summary ........................................................................... 67

8.3. Define a risk mitigation strategy .................................................. 68

8.3.1. Register fake backend, #1 .................................................. 68

8.3.2. Disrupt delivery of alarms/status/data, #8 ......................... 68

8.3.3. Denial of service, #9 .............................................................. 68

8.3.4. Eavesdropping/MitM, #4 ...................................................... 69

9. Discussion .................................................................................. 69

9.1. How can we identify the threats and risks involved in creating interconnected embedded systems? ......................................................... 69

9.2. What is a reasonable level of risk mitigation? ............................ 70

9.3. What kinds of architectures are able to mitigate these risks to a reasonable level? ................................................................. 70

9.4. How can we implement such an architecture in the real world? ........................................................ 70

9.5. Security best practices .............................................................. 71

9.6. Scenario .................................................................................. 73

10. Conclusion ................................................................................. 74

11. Future work .................................................................................. 75

12. Bibliography ............................................................................... 77

Appendix A The CIA of information security ..................................... 80

12.1. Confidentiality ........................................................................ 80

12.2. Integrity .................................................................................. 80

12.3. Availability ............................................................................. 80

Appendix B Information security terminology .................................. 81

12.4. Policy ...................................................................................... 81

12.5. Mechanism ............................................................................. 81

12.6. Assurance ................................................................................ 81
1. Introduction

1.1. Background

Semcon has worked with embedded devices for a long time. However, they foresaw an expanding market for interconnected embedded devices. They wanted to be able to remotely control, configure and monitor the status of devices. In addition to new business potential, it also opened up a lot of uncertainty regarding security. In this thesis, we sought to explore these new security concerns to identify risks present, and ways to handle them.

So the main issue is to go from a world of single-node embedded systems to a world of multi-node embedded systems.

We also wanted to stay pretty close to reality, to avoid this being a purely theoretical practice. Because of this, we formed a scenario together with Semcon; a scenario in which Semcon is approached by a customer to design a system with certain requirements and business goals.

After deciding on a scenario, we dove deeper into the security- and deployment concerns it would imply. Both on a theoretical level with risk management analyses, and on a practical level with an implementation. Our goal was to see how well we could conform to the security demands set out in the architecture, not necessarily to have it perform the exact same task as the scenario suggests.

We recognized that security is not a feature, but an emergent property of how a system operates. In order to achieve this property, we worked deliberately according to a development process that assured a certain level of Confidentiality, Integrity and Availability. Both the process and the results are covered in this thesis report.

1.2. Questions

We have distilled the background into a set of questions. These questions form the basis of what this thesis tries to accomplish.

- How can we identify the threats and risks involved in creating interconnected embedded systems?
  - What is a suitable threat model? What is a suitable risk set?
- What is a reasonable level of risk mitigation?
- What kinds of architectures are able to mitigate these risks to a reasonable level?
- How can we implement such an architecture in the real world?

We returned to these questions at the end of the thesis, to see what questions we managed to respond to.

1.3. Scenario

The scenario we chose for this thesis is a welding scenario. In this scenario, Semcon is approached by another company that operates welding machinery. In the welding apparatus, there are embedded chips that monitor different data (amount of welding fuel left, status of the welder etc.) and that control what tasks the welder performs. Currently, if something breaks
down at the factory floor, there is no way to gain insight, other than to physically approach the machinery.

This means a consultant will have to be dispatched which could take days, or at the very least hours. In a factory scenario, every hour of standstill creates huge costs for everyone involved. If the systems were embedded and networked, it would be possible to diagnose from a distance, update orders remotely or even get a warning before something breaks down. The remote location in question could be either in a coordination office at the location, or at a headquarters location far away.

The company wanted to be able to operate these embedded chips from a remote location. Operating from a remote location means two things:

i) Getting timely and correct data on a number of status variables
ii) Sending instructions to the welding apparatus

The main issue is not to have to physically go to each embedded system individually to review status or change its instructions. Networking a system like this will open up a lot of business opportunities as mentioned, but also some security issues. Those security issues (and their solutions) are the focus of this thesis. But in order to weigh technical risks against each other, the business context must be kept in mind.

### 1.4. Scope

We sought to find a scenario, and a system that could perform the task given by the scenario. But in order to create a complete piece of work, a few limitations had to be made. It was our intention to focus on the process of creating this architecture as well as the architecture itself. As far as possible, we wanted to stay out of implementation details. The hardware internals of the systems would at this stage largely be regarded as black boxes, with a few exceptions where the hardware specifics cannot be disregarded.

In the cases where we implemented systems, we sought to do so in as high a level as possible, where a commercial application might try to do it from a lower level to keep costs down. We used hardware that was able to run Debian Linux, namely Raspberry Pi and BeagleBone, which meant we could leverage the components of those systems instead of having to build everything from scratch. For a real large-scale commercial application, hardware costs can be of significant importance, so staying as close to bare minimums may be desirable.
2. Theoretical framework

A large part of the theoretical framework has been included in the form of appendices. In order to understand the parts left here you will need to be familiar with the terms used in information security. The most important ones are listed in Appendix A and Appendix B. A lot of the terms are words recurring in other parts of software development, but the meanings are sometimes specific to information security. 12.13 is also relevant as part of the theoretical framework; it lists an industry-standard set of best practices.

2.1. Sources

For the initial part, we first studied the books used as course material in the relevant courses. During this part, we focused on just reading on a lot of background we knew we would need for example the risk analysis. This was also to prepare for evaluating the scenarios.

After that, we approached the literature on a more agile basis, looking up information as needed. Before deciding on a system architecture, we would turn to the recent literature to find the current state-of-the-art for different solutions regarding security. Here, there literature was more cutting-edge than the previous course literature. We had access to DiVA for research articles from LiU, to CPL for research from Chalmers and to IEEE Xplore for research from conference proceedings and magazines. In particular, we used IEEE Xplore to find to find information from a magazine called Security and Privacy.

The literature for the implementation was a combination of the Semcon documentation of similar earlier projects that they have performed, documentation from suitable programmable embeddable systems and sources from literature mentioned previously.

2.2. Cryptography

2.2.1. Cryptographic hashes

Cryptographic hashes is a family of mathematical functions that serve a central role in cryptography. They are sometimes called one-way-hashes. These hashes are used in other areas than cryptography. For instance, given a large file, a hash can be computed to represent the file. After a network transfer, the hash can be re-computed. If the computation arrives at the same number as prior to the transfer, then the file is considered consistent. This is called a checksum or message digest (Bishop, 2004).

For a hashing function to be considered strong enough to be usable in cryptographic situations, it needs fulfil a few additional requirements:

1. Given a digest, it should be difficult to compute the original value.
2. Given a message, it should be difficult to find another message that would give the same digest. This is called a collision.
3. It needs to be fast.

The input space of these hash methods is theoretically unlimited and only limited by the realities of what a computer can handle. The output space is rather limited. Different hash methods have different length. Crc32 for instance has an output space of 32 bits, while SHA-
256 has an output space of 256 bits. Therefore, finding a collision for a Crc32 hash by pure brute force is far quicker than doing the same for SHA-256.

### 2.2.2. Strength

The word *difficult* in the requirements hints that it is not completely impossible to compute original values or find collisions. One must always find a suitable level of security depending on the application. It used to be that MD5 was considered a cryptographic hash, but researchers have found methods to calculate collisions in faster and faster ways. It should not be used as a cryptographic hash (Ming, Mao, 2009).

Since computer resources available to both attackers and others constantly increase, this measure of strength must be updated now and then to keep up.

### 2.3. Certificates

Certificates are a way of representing identity (see 12.13) within computing. It relies on digital signatures to form a chain of trust. One can never be entirely sure someone is who they claim to be over a network, but you can prove that they are who they say they are according to someone else. The validity of a certificate must be signed by a trusted entity.

Certificates are based on asymmetric cryptography, where you have a public key and a private key. The public key is included in the certificate as public knowledge, and the private key is held by the certificate owner. Thus, a certificate can be used to encrypt data that only the certificate holder can decrypt (Bishop, 2004).

#### 2.3.1. Chain of Trust

Certificates are signed by Certificate Authorities, or CA for short. The trust on the validity of a certificate must always be based on trust in the reputability of the CA. A CA may sign different levels of certificates, depending on how these certificates are issued. For instance, in 1996 VeriSign Corporation had a number of different levels. Level 1 meant they had authenticated a user’s email address. Level 3 meant they had performed a background check via a third party into the matter. Depending on what level of assurance a certificate needs, a reasonable issuance policy must be used.

It is only reasonable to have different levels of assurance in different organizations. In a military organization, a certificate used to identify a co-worker must have a very high level of assurance. At the same time, a certificate used to identify a user registering for a university course would be a different matter.

### 2.4. Authentication

Authentication is the concept of verifying the identity of a subject. It is very central to security. Authorization and audit are completely dependent on authentication being performed correctly. Successfully pretending to be someone else is the beginning of many attacks (Bishop, 2004).

#### 2.4.1. Basics

There are basically four different things that a subject can show to prove who they are.
- Something the subject knows.
  Such as a password.
- Something the subject has.
  Such as a smart card.
- Something the subject is.
  Biometrics; DNA, fingerprints.
- Where the subject is.
  Such as inside the server room.

These all have their strengths and weaknesses, and good security typically relies on a combination of the different types. Combining two types is called two-factor authorization and is typically a better way to improve security than making the one-factor authorization stronger (Schneier, 2005). For instance, Linköping University requires a code along with the identification card to enter the building at night, and recommends these to be stored separately (Linköping University, 2011).

A big weakness with biometric authentication (aside from the practicalities) is that it cannot be replaced. A lost password (something you know) can be changed. Something the user has can be replaced. Something the user is cannot be altered after an attacker has created a working forgery.

### 2.4.2. The process of authentication

Information based authentication can be said to consist of five components (Bishop, 1991):

1. The set A of authentication information subjects use to prove their identities.
2. The set C of complementary information that the system stores to validate authentication information.
3. The set F of complementary functions that generate all C from all A.
4. The set L of authentication functions that verify identity. Given an A and a C, should give “true” or “false”.
5. The set S of selection functions that enable a user to create or alter A or C.

Example of a standard salted and hashed password system, where H is a cryptographic hashing function:

1. The set A: username and password from the user.
2. The set C: a salt and a hashed password per user.
3. The set F: the set of functions $F_{user}(x): H(x+salt_{user})$
4. The set L: the set of functions $C_{user} = F_{user}(A_{user})$
5. A function to let the user change $A_{user}$.

In order to keep authentication secure, some of this information needs to be secret. There are two ways of doing this: one is to keep secret the information of A, B or F. The other is to prevent access to the authentication mechanism L. For example, lock out login attempts over the network to make the system unreachable even for a user that has all information needed for authentication.
2.4.3. Challenge-Response

A big problem with passwords is that once they are known, they are easy to re-use. Eavesdropping on an authentication session can therefore reveal passwords and/or enough other data to allow for a replay-attack\(^1\). A mitigation technique to that is a challenge response.

When asking for a login, a challenge nonce is given by the server. It is a temporary one-use random value. The client will perform a function with this value and respond with the value to the server. The server will perform the same function on the value given and match it with the response. Here, the function itself is the secret that proves identification. It can be any kind of transformation function. Then, only the nonce and the response will be sent as network traffic, and the password remains secret.

\(^1\) Recording authentication data and replaying it to appear as the same identity as the recorded one.
However, this technique does nothing to reduce the risk of having eavesdroppers. Instead, the focus is moved to the transformation function. A common way of using it is having a cryptographic hash (see 2.2.1) as the transformation function and letting the nonce be a combination of the current time and a random value. To avoid replay attacks, the same nonce must never be used twice.

### 2.5. Risk Management

Security is an emergent property of a system, not a feature. This is like how “being dry” is an emergent property of being inside a tent in the rain. The tent only keeps you dry if the poles are stabilized, vertical, and able to support the weight of wet fabric; the tent also must have waterproof fabric (with no holes) and be large enough to protect everyone who wants to remain dry. ... So, although having poles and fabric is important, it’s not enough to say, “the tent has poles and fabric, thus it keeps you dry!” (Hope, et al., 2004)

In a project developing any kind of information processing system, it is important to build security in during the development. Some of the mechanics needed to implement your policy and enforce your requirements might be so integral to the project, that they need to be there from the start (Bishop, 2004). The rework needed to add security in later is inestimable at the project planning phase, but can be expected to be greater than the work needed to build security in.

Since security is such an integral part of the process, there are many process frameworks aimed at helping developers achieve a high level of trustworthiness in projects. Some artifacts and activities are common to those frameworks, and will be covered here.

#### 2.5.1. Artifact

An artifact is a tangible by-product of a development activity. It can be just about anything, but is commonly some kind of report, code or other document.

#### 2.5.2. Activity

An activity is something you do as part of a software project. Its output can be varied; some activities produce a better understanding of what you’re trying to achieve. Some activities produce artifacts to be used later in the process. And some activities produce something less tangible, like maintaining consumer reputation.

#### 2.5.3. Misuse cases

Use cases and misuse cases are a common artifact in security engineering. It seeks to detail a list of valid “use cases” for a system, and to define a set of “misuse cases”. There are two basic types of entities:

- Actors
- Use case

For every service the system is to provide, a use case is present. For instance, in a restaurant some of the use cases would be “eat food”, “order food”, “pay for food”, “cook food”, “steal payment”. The use cases all have either positive or negative connotations, depending on if the
use case is a desirable one or not. The positive ones are called use cases, while the negative ones are called misuse cases. Each of the cases can be performed by the different actors acting upon the system. In the restaurant example, these actors could be “waiter”, “customer”, “thief” and “cook”. The waiter, customer and cook are connected to the positive use cases, while the thief is connected to the misuse cases. The diagram will show what connection these actors have on the use cases of the system and how the use cases relate to each other (Hope, et al., 2004).

Positive actors can also be connected to misuse cases in the case of accidents. For instance, leaving the restaurant without paying could be done either deliberately by a thief or by sheer forgetfulness.

Misuse cases will generally be created early on in the project, to serve as a background when finding the requirements of the system. Misuse cases are sometimes subdivided into abuse-cases and misuse-cases, where abuse cases signify purposeful abuse and misuse cases occur due to accidents.

2.5.4. Security requirements

In the early stages of the software development process, you make a list of requirements for the finished system to fulfil. These requirements can be based off of the use- and misuse-cases created earlier. There are both functional and non-functional requirements. Functional ones are on the form “the system should have a function that lets restaurant clients pay”. They answer the question “what should the system do?”.

Non-functional requirements describe the characteristics of the system, rather than the functionality. They deal in characteristics such as availability, maintainability, price etc. It is, however, important to refrain from describing how these characteristics are to be implemented. We are still early in the process here, and want to avoid locking ourselves into a solution already.

If there are some architectural demands that we know of already, we can also include them as requirements.

2.5.5. Risk analysis

Risk analysis is an integral part of the risk management process. It deals with potential future events, and their consequences. In essence, we seek to find out what threats the system will face, and what the negative impacts would be. Creating a completely secure system is of course impossible, but we can find out what risks are possible to deal with, and which ones are not worthwhile. In the end, security is a business decision, and risks can be dealt with in that way. Maximizing return on investment means that you first deal with the low-mitigation-cost, high-risk-cost threats. Risks that are unlikely to happen, cause little damage if they do and would cost a lot to mitigate get a low priority, and are unlikely to get fixed (Paco Hope, 2005).

A risk is, as mentioned previously, a combination of several components. It consists of a threat (an unwanted event), a vulnerability, an exposure and a consequence. To mitigate a threat, we can either reduce the chance that it will happen, or reduce the consequence should it happen.
Threat modelling is a part of risk management, a part that happens quite early in the process. In it, you decompose the system and identify what assets and components it has. After that, you identify and categorize threats to each asset or component (Davis, et al., 2004).

2.5.6. Static analysis

Static analysis is a method of code analysis. It analyses a program without running it, by just reading the code and finding patterns that are prone to create security related defects. This is especially important in programming languages that are prone to cause code defects, such as C. Static analysis can often find a lot of false positives (something that looks wrong, but actually works fine), which sometimes deter people from doing it or trusting it. It has also been argued that these false positives point to bad programming patterns that should be avoided as they are prone to fail (Zheng, et al., 2006).

2.5.7. Attack trees

Attack trees are a way of modeling the paths possible toward a threat (an unwanted event). You start off the unwanted event as a root node for a tree, and build the possible paths from there. The reason for doing so is to be able to prioritize different risks, finding the easiest path to the threats and dealing with them first. Each leaf nodes in the tree should be given some kind of weight to be able to assess its difficulty/likelihood for an attacker to perform. Non-leaf nodes inherit their weight from the cheapest child. Here is an example, modeling the different paths to get a safe open (Schneier, 1999):

\[
\text{Open Safe} \\
\text{NSE/¥20K} \\
\text{Pick Lock} \\
\text{NSE/¥30K} \\
\text{Learn Combo} \\
\text{NSE/¥20K} \\
\text{Cut Open Safe} \\
\text{NSE/¥10K} \\
\text{Install Improperly} \\
\text{NSE/¥100K} \\
\text{Find Written Combo} \\
\text{NSE/¥50K} \\
\text{Get Combo From Target} \\
\text{NSE/¥20K} \\
\text{Threaten} \\
\text{NSE/¥60K} \\
\text{Blackmail} \\
\text{NSE/¥100K} \\
\text{Eavesdrop} \\
\text{NSE/¥80K} \\
\text{Bribe} \\
\text{NSE/¥20K} \\
\text{Listen to Conversation} \\
\text{NSE/¥20K} \\
\text{Get Target to State Combo} \\
\text{NSE/¥40K} \\
\]

2.6. Annual Loss Expectancy

ALE is a simple way of weighing risks against each other. In order to understand how important it is to deal with a risk, two variables are taken into account: risk of occurrence and consequence of occurrence. In ALE these are represented as ARO and SLE. ARO stands for Annual Rate of Occurrence and signifies how often you expect a negative event to occur if left unmanaged. SLE stands for Single Loss Expectancy and signifies how much it will cost each time the event happens. These are multiplied together to form ALE: \( \text{ALE} = \text{ARO} \times \text{SLE} \).
ALE for a risk can be expressed as “how much does it cost us annually to keep this risk unmanaged?” (Shimonski, 2004).

2.7. Risk Management Framework

The model we will be looking at closer is called simply RMF, Risk Management Framework. It is a framework that consists of five activities (McGraw, 2005):

1. Understand the Business Context
2. Identify and link the Business and Technical Risks
3. Synthesize and Rank the Risks
4. Define the Risk Mitigation Strategy
5. Carry out Fixes and Validate

It is meant to be performed in an iterative fashion, returning to step 2 after completing step 5. Notice that the framework does not specify specific tasks or artifacts, but rather the overarching philosophy of what you should be doing in those steps.

There are different ways to interpret how to use these steps, and they will often change from project to project. Also, some more veteran analysts will sometimes skip steps depending on the situation; something might have come up that is known to affect steps 2 and 5, so skipping performing 3 and 4 might be valid.

![Figure 1: Risk management framework overview (McGraw, 2005)](image)

2.7.1. Understand the Business Context

Central to the risk management framework is how it relates to business. We have to understand the scenario, and what the system’s place in that scenario is. What actors (positive/negative) are there? What events are positive, and which are to be regarded as threats? Which of them are abuse cases and which are just misuse? Understanding the business context will also let you create two different kinds of risks: technical risks and business risks.
2.7.2. Identify and link the Business and Technical Risks

Here you will commonly be doing something like use-cases and abuse-cases will help you find threats (unwanted events). This will provide the foundation for the risks. One must always weigh the risks and costs of mitigation to the return on investment. Central to this stage of the analysis is to map technical risks via business risks to business goals.

2.7.3. Synthesize and Rank the Risks

Here we rank risks depending on which technical risks are most important to the business goals. Via the mapping through business risks, we can find what technical risks are most harmful to the system. The output artefact from this stage should be a list of all risks and their priority. If a risk is very expensive to fix but does not provide substantial risk mitigation, it might have a low priority. What terms these are weighed in depends on the scenario, but common metrics are money, development time, risk likelihood, risk impact and risk severity.

2.7.4. Define a Risk Mitigation Strategy

Using the technical risks defined and ranked earlier, we must find a coherent risk mitigation strategy. It must take into account the business context, and the impact it will have upon it. In addition to the costs found earlier to rank how expensive the risks were, we must here find a measure of how expensive the fixes are to perform. The strategies may also have an effect on other risks in the set, or introduce new ones altogether. We should also look ahead and find verification strategies, to verify whether the risks really have been mitigated, and to what degree.

2.7.5. Fix the Problems and Validate the Fixes

Here we implement the risk mitigation strategy defined earlier. We should also try to measure how well the fixes have been implemented in terms of how well they take care of risks. Also unexpected effects, such as fixes introducing new risks should be mentioned here.
3. Method

3.1. Overview

The first step will be a literature study. This will be to lay a foundation of information security essentials that the rest of the thesis will be based on. During this study, we will draw information mainly from earlier course books and other sources of foundation material. The study will also lay a foundation for the risk analysis and threat modelling that is to be performed on one of the scenarios mentioned below.

After that, we will have a structured brainstorming to find out what different scenarios could be relevant for the thesis. This will be based both on the demands of the client company and a judgement of what is relevant for the thesis. We will not take into account to what extent this can be implemented and realized within this thesis at this stage, as we anticipate that certain limitations will have to be made either way.

One of these scenarios will be chosen as the scenario to focus the latter parts of the theoretical exercise on. After describing the scenario in more detail, we will perform a resource identification, risk analysis and threat modelling. Using this information, we will find a suitable level of system security. An architecture will be designed to achieve the level of system security found to be suitable.

3.2. Scenario

At the start of every software project is the ever important planning phase. When aiming for a secure system, it is even more so. To align the desires of Semcon and the university, we held a brainstorming session. The direct results of the session are available as Appendix A. A short description of the scenario was produced, as detailed in the introduction. We dive deeper into it in section 4.

3.3. Risk management framework

The risk management framework was interpreted thus:

1. Understand the Business Context
   a. Describe scenario, find business goals
   b. Create a suitable system architecture
2. Identify and link the Business and Technical Risks
   a. Identify key assets in the architecture
   b. Create security requirements from the business goals
   c. Draw attack trees with attacks on the assets as top nodes
3. Synthesize and Rank the Risks
   a. Tie the results of step 2 together into a list of security risks
   b. Perform ALE on the security risks
   c. Rank the risks according to how expensive they are
4. Define the Risk Mitigation Strategy
   a. For the top scoring risks, find a risk mitigation strategy
5. Carry out Fixes and Validate
a. Implement the mitigations found in 4. If necessary, make alterations to the fixes.
b. Validate the fixes; make sure that the risk really is mitigated to expected degree, and find out what new risks they have introduced.

What the risks and strategies will be differ between the different parts of the process; they will affect different artefacts. One iteration of RMF was performed during the design phase of the project, where the risk management will examine the use case, business goals and architecture artefacts.

After a first implementation of the architecture, we ran RMF one iteration more, to add new technical risks that have become apparent during implementation. In some cases the reality of implementation forced unexpected architecture changes that affected risks, and sometimes the deeper knowledge garnered from implementation lead us to risks previously not thought of.

We followed the RMF framework and produced artifacts explained in the theory section. In the planning section of the thesis, we include these artifacts: Scenario description, Business context, use cases, business and technical risks and a system architecture.

3.4. Waterfall

RMF was used for the security process, and in parallel a waterfall-like process was run for the implementation. Some steps went together with RMF and waterfall, and some steps were run in isolation from the other process. This is what our waterfall process looked like:

- Requirements analysis
- Design
- Implementation
- Deployment
- Maintenance

3.5. Combining RMF with Waterfall

Requirements analysis went hand in hand with the first step of RMF, understand the business context. They are very similar and produce about the same artifacts. After that we did steps two through five of RMF, while the waterfall process was on hold. Upon completion, we returned to waterfall, and found that the design step had also largely been covered in the first RMF step. So we finalized design and performed implementation and deployment.

After that we returned to run another iteration of RMF, steps two to five again. Each iteration of RMF will serve to improve the software, and was seen as a continuation of the waterfall process as well. A sort of iterative maintenance.
4. Planning

This section represents a combination of step one of waterfall and step one of RMF. Since they have so much in common, they have been joined together in the same section. We also made a bit of implementation planning. Not thoroughly, but just enough to produce artifacts to perform our first iteration of RMF on later.

4.1. Scenario

As mentioned in section 1.3, a welding scenario was chosen as scenario for this thesis.

In this scenario we had a number of security requirements. Some them arose from the added attack vectors that networking creates, and some were inherent to the nature of the business case. When exposing such systems to networks, we gain a significant benefit from being able to aggregate the operation of many systems, but this benefit applies to both benevolent operators and malignant attackers.

4.1.1. System overview

![System overview diagram]

In this system we have three different kinds of systems: backend, gateway and leaf. The backends (denoted B1, B2 in Figure 2) are the systems collating the data. A typical scenario would have one backend at the site for the local operators, and one backend at a remote site for data analysis. Backends can be placed on different domains. The links Li1 between the gateway and a remote backend were considered out of operating the company’s control and therefore untrusted. The backends needed to be able to send configuration data to the leaf nodes as well.

The gateway (denoted G in Figure 2) is the center hub of the system; it keeps track of the data flow, the leaves and the backends. Much of the security is focused on this device.
The leaves (denoted Le1, Le2, Le3 in Figure 2) are the systems embedded in the individual welders. Pieces of sensory equipment in the welders are connected to the leaves, which in turn pass the data on to the gateway. The leaves are unaware of the rest of the system topology, and are homogenous “dumb” nodes. Leaves are connected to the gateway through links denoted Li2.

We will regard the backends as just a server in this case. In a real scenario, much of the business use of the system would probably come out of how these are used, but we are focused on the technical aspects that allow the system to function.

We begin our first iteration of RMF with “Understand the business context”.

4.1.2. Business Context

In this business, Semcon is the provider of the system. In a real business context, both the backend and the frontend would probably be important selling points of the system. In this case, we chose not dive too deeply into the features (centralized collation, direction etc.) of the backend system. But we made sure that the frontend system was technically capable of achieving the three goals stated in the scenario description.

These were the business goals of the system:

i) Ensure a quick warning system when something breaks down (or when possible, before)
ii) Minimize costs of sending technicians to the physical location
iii) Help business become more agile by centralizing the management of welding apparatus
iv) To fit within companies’ IT policies, infrastructure and other constraints

Another important note is that Semcon would not be in control of either the system or the environment where it resides. That means that technology had to be as feasible as possible within a target company’s IT infrastructure. We could not expect every client company to allow any kind of obscure IT policy or technical IT layout. Therefore, we had to make reasonable assumptions on what generally reasonable target IT infrastructure looked like.

4.2. System usage

Normal system usage was modeled in the use-case diagram named Figure 3.
For the use-cases we found four types of actors in the system: the operator, the backend, the gateway and the welder. These have been explained in the Scenario description. The basic services explained in the business context were fulfilled by these use cases.

There were three basic types of data transactions in this system:

- **Periodic data transfer**
  Bulk welding data that is not needed quickly would be transferred every now and then, at a set interval.

- **Immediate weld-initiated data transfer**
  Some data originating from events taking place at the welding-side of the system were to be reported as quickly as possible, such as temperature alarms and shortage of welding material.
• Backend-requested data
Some data would be requested by an operator interaction, such as viewing current backends. Note that backends were able to request data either from the gateway or from individual nodes.

4.3. Process
This is how different parts of the system would enable a few of the business goals.
Figure 4: Periodic welder data sequence diagram

Figure 5: Setting welder config sequence diagram

4.4. Technological survey

In order to realize the system, we needed to find what hardware and software to use in different parts of the system. We used terms from the standardized OSI model (Tranter, et al., 2007). We began with finding hardware for the link layer.
4.4.1. Link layer

One technology that was disregarded was RuBee. It might be relevant, but was deemed out of scope.

Data links were evaluated on these criteria:

<table>
<thead>
<tr>
<th>ID</th>
<th>Parameter</th>
<th>Description</th>
<th>Weight</th>
</tr>
</thead>
</table>
| 1  | Implementation cost        | Hardware and wiring cost                                                    | 3:low  
                                          |                                            | 2:medium  
                                          |                                            | 1:high  |
| 2  | Throughput                 | Mean bit rate                                                               | 3:high  
                                          |                                            | 2:medium  
                                          |                                            | 1:low  |
| 3  | Physical clutter           | How much clutter (cables/routers etc.) does the method create               | 3:minimal  
                                          |                                            | 2:medium   
                                          |                                            | 1:much  |
| 4  | Responsiveness             | Delay in transmitting data. This includes the factor of stability.          | 3:quick  
                                          |                                            | 2:medium  
                                          |                                            | 1:unreliable |
| 5  | Technical risk             | A measure of the technical stability of the method.                         | 3:stable  
                                          |                                            | 2:medium   
                                          |                                            | 1:emerging |
| 6  | Open source                | How much of the architecture is proprietary/closed source?                  | 3:all open  
                                          |                                            | 2:most open  
                                          |                                            | 1:proprietary |
| 7  | Distance                   | Is distance cross-domain or short-range?                                   | 3:cross domain  
                                          |                                            | 2:cross domain with IT policy support  
                                          |                                            | 1:short range |

Inspiration for criteria and values were from a US Army report (Dattathreya, 2009). High weight-numbers were good. The distance-parameter: “cross domain with IT policy support” meant that cross domain traffic was possible, but it relied on the company’s help by letting data through their system. Security was an integral part of this project, but was not chosen as a criterion since we tackled that on another OSI layer.
### Gigabit Ethernet (802.3ab)

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Details</th>
<th>Ranking</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Cost</strong></td>
<td>Relatively cheap</td>
<td>3</td>
</tr>
<tr>
<td><strong>Throughput</strong></td>
<td>1 Gbyte/s</td>
<td>3</td>
</tr>
<tr>
<td><strong>Physical clutter</strong></td>
<td>Cabling and routers</td>
<td>1</td>
</tr>
<tr>
<td><strong>Responsiveness</strong></td>
<td>High responsiveness</td>
<td>3</td>
</tr>
<tr>
<td><strong>Technical risk</strong></td>
<td>Mature</td>
<td>3</td>
</tr>
<tr>
<td><strong>Open source</strong></td>
<td>Public standard</td>
<td>3</td>
</tr>
<tr>
<td><strong>Distance</strong></td>
<td>Cross domain with IT support</td>
<td>2</td>
</tr>
</tbody>
</table>

### Wireless Ethernet (802.11 family)

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Details</th>
<th>Ranking</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Cost</strong></td>
<td>Relatively cheap</td>
<td>3</td>
</tr>
<tr>
<td><strong>Throughput</strong></td>
<td>2Mbit/s to ~300Mbit/s</td>
<td>3</td>
</tr>
<tr>
<td><strong>Physical clutter</strong></td>
<td>Routers to extend range</td>
<td>2</td>
</tr>
<tr>
<td><strong>Responsiveness</strong></td>
<td>Depends on local circumstances</td>
<td>2</td>
</tr>
<tr>
<td><strong>Technical risk</strong></td>
<td>Mature</td>
<td>3</td>
</tr>
<tr>
<td><strong>Open source</strong></td>
<td>Public standards</td>
<td>3</td>
</tr>
<tr>
<td><strong>Distance</strong></td>
<td>Short-range (can be extended to several hundred metres, but cross domain requires cabling)</td>
<td>1</td>
</tr>
</tbody>
</table>
**Zigbee**

Zigbee is a specification on many OSI-levels, mainly on the application layer to provide a high level abstraction. It has some built-in security measures. There is a focus on long battery life and having a very small power footprint (Zigbee Alliance, 2007). It can also be used routerless in a self-organizing clutterless mesh network, but network overhead for routing will quickly decrease throughput and responsiveness.

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Details</th>
<th>Ranking</th>
</tr>
</thead>
<tbody>
<tr>
<td>Cost</td>
<td>Cheap in high volume production.</td>
<td>2</td>
</tr>
<tr>
<td>Throughput</td>
<td>250kbit/s = 31.25kbyte/s</td>
<td>1</td>
</tr>
<tr>
<td>Physical clutter</td>
<td>Embeddable</td>
<td>3</td>
</tr>
<tr>
<td>Responsiveness</td>
<td>Depends on local circumstances</td>
<td>2</td>
</tr>
<tr>
<td>Technical risk</td>
<td>Mature, but depends partly on commercial interests</td>
<td>2</td>
</tr>
<tr>
<td>Open source</td>
<td>Free membership for specification, but an involved certification process.</td>
<td>1</td>
</tr>
<tr>
<td>Distance</td>
<td>Short-range, between 10 and 75 metres (cross domain requires cabling)</td>
<td>1</td>
</tr>
</tbody>
</table>

**GPRS/W-CDMA**

Ordinary telephony standards can also be used for general-purpose data transfer. Standards are released by 3GPP on GPRS (3GPP, 2012) and W-CDMA.

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Details</th>
<th>Ranking</th>
</tr>
</thead>
<tbody>
<tr>
<td>Cost</td>
<td>Relatively expensive</td>
<td>2</td>
</tr>
<tr>
<td>Throughput</td>
<td>54-114kbit/s (GPRS) / 5.76mbit/s (HSUPA)</td>
<td>1 / 2</td>
</tr>
<tr>
<td>Physical clutter</td>
<td>Embeddable</td>
<td>3</td>
</tr>
<tr>
<td>--------------------------</td>
<td>----------------------</td>
<td>----</td>
</tr>
<tr>
<td><strong>Responsiveness</strong></td>
<td>Shaky. Depends on local circumstances. GPRS widely available, but sensitive to disturbance.</td>
<td>1</td>
</tr>
<tr>
<td><strong>Technical risk</strong></td>
<td>Mature</td>
<td>3</td>
</tr>
<tr>
<td><strong>Open source</strong></td>
<td></td>
<td>3</td>
</tr>
<tr>
<td><strong>Distance</strong></td>
<td>Long-range</td>
<td>3</td>
</tr>
</tbody>
</table>

**Assumptions**

We made some assumptions about the data rate required for the welders:

<p>| | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>Welder robots</td>
<td>20</td>
</tr>
<tr>
<td>Monitored variables</td>
<td>5</td>
</tr>
<tr>
<td>Sampling interval(seconds)</td>
<td>60</td>
</tr>
<tr>
<td>Data points per minute</td>
<td>100</td>
</tr>
<tr>
<td>Data point + timestamp size bytes</td>
<td>128</td>
</tr>
<tr>
<td>Kbytes data per day</td>
<td>18432</td>
</tr>
<tr>
<td>Kbytes/day after Zlib compression</td>
<td>6144</td>
</tr>
<tr>
<td>Kbytes/day after Zlib compression per robot</td>
<td>307,2</td>
</tr>
</tbody>
</table>

We wanted to reserve bandwidth to have the ability to send software updates in the future. For this, we made the assumption that a typical software update is 10 Megabytes in size.

Regarding alarms, we assumed that a delivery time of 30 seconds is tolerable. This is only imposed on backends in the same domain as the welders.

**Conclusions**

We found at least two different kinds of links in our system: the intra-domain link and the inter-domain link. For the inter-domain link we had a few requirements:

- We must be able to transfer all the data.
- Must be able to send software updates.
- High responsiveness would be good, but is not mission-critical.

Based on this, either Gigabit Ethernet or W-CDMA is suitable. GPRS does not allow for the transfer of the software updates in a reasonable time span. With 54 kbit/s, 10 megabytes would take $10000 / (54/8) = 10000 / 6.7kbyte/s = 1492$ seconds, which is 25 minutes. This assumes good network conditions.
Conclusion: either Gigabit Ethernet or W-CDMA if possible for the inter-domain link.

Except traversing domains, intra-domain link had the same requirements that the inter-domain link had. The scenario also imposed a few requirements:

- Data must be delivered quickly (within 30 seconds)
- Physical clutter should be minimal

These weights were applied to weigh the alternatives:

Cost, throughput, physical clutter, responsiveness, technical risk, open source, distance

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Weight</th>
</tr>
</thead>
<tbody>
<tr>
<td>Cost</td>
<td>10%</td>
</tr>
<tr>
<td>Throughput</td>
<td>20%</td>
</tr>
<tr>
<td>Physical clutter</td>
<td>25%</td>
</tr>
<tr>
<td>Responsiveness</td>
<td>25%</td>
</tr>
<tr>
<td>Technical risk</td>
<td>10%</td>
</tr>
<tr>
<td>Open source</td>
<td>10%</td>
</tr>
<tr>
<td>Distance</td>
<td>0%</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Weight</th>
</tr>
</thead>
<tbody>
<tr>
<td>Cost</td>
<td>10%</td>
</tr>
<tr>
<td>Throughput</td>
<td>20%</td>
</tr>
<tr>
<td>Physical clutter</td>
<td>25%</td>
</tr>
<tr>
<td>Responsiveness</td>
<td>25%</td>
</tr>
<tr>
<td>Technical risk</td>
<td>10%</td>
</tr>
<tr>
<td>Open source</td>
<td>10%</td>
</tr>
<tr>
<td>Distance</td>
<td>0%</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Weight</th>
</tr>
</thead>
<tbody>
<tr>
<td>Total</td>
<td>100%</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Cost</th>
<th>Throughput</th>
<th>Physical clutter</th>
<th>Responsiveness</th>
<th>Technical risk</th>
<th>Open source</th>
<th>Distance</th>
<th>TOTAL</th>
</tr>
</thead>
<tbody>
<tr>
<td>Gigabit Ethernet</td>
<td>3</td>
<td>3</td>
<td>1</td>
<td>3</td>
<td>3</td>
<td>3</td>
<td>2</td>
</tr>
<tr>
<td>Wireless Ethernet</td>
<td>3</td>
<td>3</td>
<td>2</td>
<td>2</td>
<td>3</td>
<td>3</td>
<td>1</td>
</tr>
<tr>
<td>Zigbee</td>
<td>2</td>
<td>1</td>
<td>3</td>
<td>2</td>
<td>2</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>GPRS</td>
<td>2</td>
<td>1</td>
<td>3</td>
<td>1</td>
<td>3</td>
<td>3</td>
<td>3</td>
</tr>
<tr>
<td>WCDMA</td>
<td>2</td>
<td>3</td>
<td>3</td>
<td>1</td>
<td>3</td>
<td>3</td>
<td>3</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Cost</th>
<th>Throughput</th>
<th>Physical clutter</th>
<th>Responsiveness</th>
<th>Technical risk</th>
<th>Open source</th>
<th>Distance</th>
<th>TOTAL</th>
</tr>
</thead>
<tbody>
<tr>
<td>Gigabit Ethernet</td>
<td>0.3</td>
<td>0.6</td>
<td>0.25</td>
<td>0.75</td>
<td>0.3</td>
<td>0.3</td>
<td>0</td>
</tr>
<tr>
<td>Wireless Ethernet</td>
<td>0.3</td>
<td>0.6</td>
<td>0.5</td>
<td>0.5</td>
<td>0.3</td>
<td>0.3</td>
<td>0</td>
</tr>
<tr>
<td>Zigbee</td>
<td>0.2</td>
<td>0.2</td>
<td>0.75</td>
<td>0.5</td>
<td>0.2</td>
<td>0.1</td>
<td>0</td>
</tr>
<tr>
<td>GPRS</td>
<td>0.2</td>
<td>0.2</td>
<td>0.75</td>
<td>0.25</td>
<td>0.3</td>
<td>0.3</td>
<td>0</td>
</tr>
<tr>
<td>WCDMA</td>
<td>0.2</td>
<td>0.6</td>
<td>0.75</td>
<td>0.25</td>
<td>0.3</td>
<td>0.3</td>
<td>0</td>
</tr>
</tbody>
</table>

With these weights, we decided to look into Gigabit Ethernet and Wireless Ethernet. It should be noted that if the weight of “throughput” was diminished, Zigbee quickly became the
winner. It was the requirement to be able to push software updates of 10MB in size to the leaves that puts Zigbee in a bad spot.

Conclusion: Use Wireless Ethernet (802.11) or Gigabit Ethernet for the intra-domain link.

4.5. Hardware platform

At this stage there were not many requirements on the hardware platform, other than that it had to support the link layer. The implementation was to be performed on a BeagleBoard and two Raspberry Pis, which are open source (both hardware and software) general purpose computers. They can run a complete Linux-stack.

Since the hardware is open source, it would be possible to strip away parts that are unwanted or just not needed late in the process. For now, the cost is low enough for our academic endeavors.

4.6. Presentation layer

The presentation layer is part of the OSI model as well. It is part of the data layer, and has the function of data representation, encrypting and decrypting and converting data models to an interoperable form. For this project we reviewed the protocol XMPP, which is an open set of standards for instant messaging, remote presence and general routing of XML data (XMPP Standards Foundation, 2012). We found it to be a good match for the presentation layer.

We had not yet assessed whether we can use the built-in security functions in XMPP. We decided to evaluate whether they fit in our scenario and are sufficiently trustworthy. If were are not, we would just use XMPP as a transport layer, and make use of other cryptography libraries on top of that. In our model, there would be an XMPP server on the gateway, and client nodes installed on the backends and leaves.

XMPP is backed by among others Google, who used it on its Wave platform. The Facebook chat can be used via XMPP. Google talk can be used via XMPP. There are a number of popular clients, such as pidgin. There are also a number of different APIs, SDKs and such for servers, clients and components.

We decided to use Openfire by ignite realtime, and a client SDK called Smack by the same organization.

5. Risk management

Following the risk management framework we had now completed step one “Understand the business context”. We had also come along some way on step two, “Identify and link the business and technical risks”. We continued on that step here:

5.1. Assets

Here we decomposed the system into its basic assets. This was necessary in order to find what threats (unwanted events) existed against those assets. We also wrote a rough estimate on how bad it would be in our business context if either C, I or A were compromised for each information asset.
<table>
<thead>
<tr>
<th>ID</th>
<th>Name</th>
<th>Description</th>
<th>Business impact if compromised</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Information</td>
<td>Information Assets</td>
<td></td>
</tr>
<tr>
<td>1.1</td>
<td>Weld data</td>
<td>The welder data stored on the leaves</td>
<td>C: Low I: Low A: Medium</td>
</tr>
<tr>
<td>1.2</td>
<td>Weld data</td>
<td>The welder data stored on the gateway</td>
<td>C: Medium I: Medium A: Medium</td>
</tr>
<tr>
<td>1.3</td>
<td>Weld data</td>
<td>The welder data stored on the backend</td>
<td>C: High I: High A: Medium</td>
</tr>
<tr>
<td>1.4</td>
<td>Alarms</td>
<td>Alarms sent to support personnel</td>
<td>C: Low I: Medium A: High</td>
</tr>
<tr>
<td>1.5</td>
<td>Software</td>
<td>The software and configuration running on gateways</td>
<td>C: Low I: High A: High</td>
</tr>
<tr>
<td>1.6</td>
<td>Software</td>
<td>The software and configuration running on leaf</td>
<td>C: Low I: High A: High</td>
</tr>
<tr>
<td>1.7</td>
<td>Private keys</td>
<td>Private keys on gateways</td>
<td>C: High I: Low A: High</td>
</tr>
<tr>
<td>1.8</td>
<td>Private keys</td>
<td>Private keys on the leaves</td>
<td>C: High I: Low A: High</td>
</tr>
<tr>
<td>1.9</td>
<td>Private keys</td>
<td>Private keys on backends</td>
<td>C: High I: Low A: High</td>
</tr>
<tr>
<td>2</td>
<td>Hardware</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
5.2. Security requirements

Our security requirements came from both the business context and the assets that the system uses. They were to be used to find security risks against the system, and were listed in this table:

<table>
<thead>
<tr>
<th>ID</th>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>2.1</td>
<td>Welder data confidentiality</td>
<td>Welder data should only be received by legitimate parties.</td>
</tr>
<tr>
<td>2.2</td>
<td>Alarm confidentiality</td>
<td>Alarms should only be received by legitimate parties.</td>
</tr>
<tr>
<td>2.3</td>
<td>Welder data integrity</td>
<td>It should not be possible to alter welder data, or add fake data.</td>
</tr>
<tr>
<td>2.4</td>
<td>Alarm integrity</td>
<td>It should not be possible to alter alarms, or create fake ones.</td>
</tr>
<tr>
<td>2.5</td>
<td>Alarm availability</td>
<td>Alarms should be available at domain-local backends within 30 seconds.</td>
</tr>
<tr>
<td>2.6</td>
<td>Welder data availability</td>
<td>The software and configuration running on gateways.</td>
</tr>
<tr>
<td>2.7</td>
<td>Network</td>
<td>Only legitimate backends should be able to add, remove and view other backends.</td>
</tr>
<tr>
<td>2.8</td>
<td>Network</td>
<td>Only legitimate leaf nodes should be able to register with the gateway.</td>
</tr>
<tr>
<td>2.9</td>
<td>Software</td>
<td>It should only be possible to view and alter configurations from legitimate backends.</td>
</tr>
<tr>
<td>2.10</td>
<td>Software</td>
<td>It should not be possible to alter the gateway- or welder software in transit or when stored on the target devices.</td>
</tr>
<tr>
<td>2.11</td>
<td>Configuration</td>
<td>Only legitimate backends should be able to send software and configurations.</td>
</tr>
<tr>
<td>------</td>
<td>---------------</td>
<td>--------------------------------------------------------------------------------</td>
</tr>
<tr>
<td>2.12</td>
<td>Configuration</td>
<td>It should not be possible to alter the gateway- or welder configuration in transit or when stored on the target devices.</td>
</tr>
</tbody>
</table>

### 5.3. Attack trees

Attack trees are another tool to help find security risks that do not arise obviously from the security requirements. Trees have a threat as the top node, and leaves represent attacks that could be launched to make that threat occur.

A few trees were been created to find such risks. Since the trees for different threats would look similar, only a few specific threats were covered. We chose one for each of C, I and A to cover most attack paths in three trees.

Figure 6 describes the possible attacks to make the threat “Steal welder data” happen. The different labels were marked with a boolean value to differentiate the ones that require some kind of physical access from the ones which are possible to perform from a remote location. Using this figure, we were able to find security requirement 2.9 and 2.10.

Figure 6 shows us the ways an attacker can implant or alter data in the system. It is in some ways similar to the previous one, but notably adds the security requirement about only allowing legitimate welders join the network (2.8).
Figure 6: Attack tree: steal welder data

Figure 7: Attack tree: Implant/Alter data
Figure 8: Attack tree: Disrupt alarms

5.4. Security risks

The next step in RMF was “Synthesize and Rank the Risks”. We used ALE to find which risks were of highest importance, which means that each risk had to be given two rankings: one “Annualized Rate of Occurrence” and one “Single Loss Expectancy”. This means “how often can it be expected to occur per year and how much does it cost each time?”. Our security risks looked like this:

5.4.1. Registering a fake backend

Num: 1

**Impact:** Registering a fake backend to a gateway is a way to start several attacks. If someone is able to register a fake backend, they would be able to passively listen to get all welder data and alarms. They would also be able to send software and configuration data, which is potentially disastrous, and would be able to affect all assets.

**Vulnerability:** No authentication required to add backends.

**Vectors:** Via either Li2 or Li1, posing as a legitimate backend, or by altering the gateway configuration.
5.4.2. Registering a fake welder

Num: 2

**Impact:** Registering a fake welder is a way to start a few attacks. If someone achieves this, they would be able to implant fake data and fake alarms. This could affect the trustworthiness of the system and its alarms, potentially leading to operators ignoring real alarms.

**Vulnerability:** No authentication required to add a welder.

**Vectors:** Via either Li2 or Li1, posing as a legitimate welder, or by altering the gateway configuration.

5.4.3. Altering configuration or software on gateway or leaf physically

Num: 3

**Impact:** If someone is able to alter configuration or software on a gateway or a leaf by means of physical access, they can deal a lot of damage. The welders themselves could be at risk, and everything implied by the section 5.4.2 as well (esp. with regards to implanting fake data). The manufacturing results could also be at risk. Since there are a lot of people that have physical access to the devices, they can be considered very exposed.

**Vulnerability:** Physical access. Data stored unsigned and in plaintext.

**Vectors:** If an agent has physical access to the devices, they alter the data as it resides on the SSD. If the device has means of physical access such as USB (which is the case with BeagleBone), that is also an option.

5.4.4. Eavesdropping or performing Man-in-the-Middle on a link

Num: 4

**Vectors:** An ill-meaning agent eavesdropping or performing Man-in-the-Middle on a link, either by being physically present to perform it on Li2 or Li1 or remotely to do it on Li1 could cause some harm as well.

**Vulnerability:** No encryption or signing of data during transfer.

**Impact:** If eavesdropping is possible, they can gather the data flowing through the system, which means alarms, welding data, configurations and software. If they can do Man-in-the-Middle, they can alter all the aforementioned data as well, which would imply all damages mentioned by 5.4.3 as well. A well-crafted Man-in-the-Middle could also potentially be harder to detect than the physical disturbance needed to perform 5.4.3.

5.4.5. Altering welder data on a gateway or leaf physically

Num: 5

**Vectors:** If an agent has physical access to the devices, they alter the data as it resides on the SSD. If the device has means of physical access such as USB (which is the case with BeagleBone), that is also an option.
Vulnerability: Physical access. Data stored unsigned and in plaintext.

Impact: If they can do that, they can manipulate the data before it is being sent to the backend or gateway. This could lead to damages to various assets: the welder data will be damaged, the welder itself might not be treated with the right kind of maintenance or repair and trustworthiness of other data may decline.

5.4.6. Reading welder data from a gateway or leaf physically

Num: 6

Vectors: If an agent can access the devices by the same means mentioned in section 5.4.5, reading data from the SSD is a simple task.

Vulnerability: Physical access. Data stored unsigned and in plaintext.

Impact: Altering the running program to output its data in another form than the standard one is also possible, by means of reverse-engineering.

5.4.7. Reading welder data from a backend

Num: 7

Impact: If an attacker can access a backend, they can read the welder data. The backend is where the data will be presented to the end-user, so it has to be available in a non-encrypted format.

Vulnerability: Physical access. Data stored unsigned and in plaintext.

Vectors: A backend can be accessed either remotely via something like SSH, or physically if local security measures are inadequate. This risk pertains to legitimate backends being accessed in an illegitimate way.

5.4.8. Disrupting the delivery of alarms

Num: 8

Impact: If an attacker can disrupt the delivery of alarms for a period of time, they can keep operators in the dark regarding the status of the welding apparatus.


Vectors: Assuming the local backend is not connected insecurely to an external network, this attack has two possible vectors: local physical disruption and remotely overloading the gateway. Local physical disruption is very simple in this case: just pull a power plug.

5.4.9. Summary

We gathered the listed risks in this table and created AROs and SLEs for them. AROs and SLEs are by nature very hard to get right, and we have gone with what we believe to be reasonable numbers. Those numbers would have a big impact on how risk is managed in the
project. The AROs took into account how easy the attack is to perform and how desirable it would be to an attacker.

Figure 9 shown below is a summary of the risks we took into account, sorted on the highest ALE. This showed us clearly which risks were necessary to take deal with first.

<table>
<thead>
<tr>
<th>Risk</th>
<th>#</th>
<th>ARO</th>
<th>SLE</th>
<th>ALE (ARO*SLE)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Reg. fake backend</td>
<td>1</td>
<td>10</td>
<td>2 000 000 kr</td>
<td>20 000 000 kr</td>
</tr>
<tr>
<td>Eavesdropping/MitM</td>
<td>4</td>
<td>5</td>
<td>2 000 000 kr</td>
<td>10 000 000 kr</td>
</tr>
<tr>
<td>Reading welder data from backend</td>
<td>7</td>
<td>5</td>
<td>500 000 kr</td>
<td>2 500 000 kr</td>
</tr>
<tr>
<td>Altering config/software on gate</td>
<td>3</td>
<td>5</td>
<td>500 000 kr</td>
<td>2 500 000 kr</td>
</tr>
<tr>
<td>Disrupt delivery of alarms</td>
<td>8</td>
<td>6</td>
<td>200 000 kr</td>
<td>1 200 000 kr</td>
</tr>
<tr>
<td>Reg. Fake welder</td>
<td>2</td>
<td>5</td>
<td>50 000 kr</td>
<td>250 000 kr</td>
</tr>
<tr>
<td>Alter welder data physically</td>
<td>5</td>
<td>2</td>
<td>30 000 kr</td>
<td>60 000 kr</td>
</tr>
<tr>
<td>Reading welder data physically</td>
<td>6</td>
<td>2</td>
<td>30 000 kr</td>
<td>60 000 kr</td>
</tr>
</tbody>
</table>

Figure 9: Risk table, sorted on ALE

5.5. Mitigations

Step four in RMF is “Define the risk mitigation strategy”.

Using the summary above, we selected a set of risks to mitigate. We could either mitigate the likelihood of occurrence (which affects ARO), or the effect if it does occur (which affects SLE). Both factors weigh into ALE, which can be thought of as a yearly cost of keeping this risk unmitigated. If a mitigation would be found to be more costly than ALE, it would not be worth pursuing. We selected the highest ranked risks according to their ALEs and found mitigations.

5.5.1. Reg. fake backend, #1

We cannot do much about the vector or impact. But we can decrease the vulnerability. We can require a backend to be authorized in order to add another backend. We must require all requests to add backends to be signed to verify origin and contents. That means we must share a set of public keys between the nodes of the network. That way we can authenticate identity. If we could set up some kind of shared secret, then that would be a good way too, but because of the physically exposed position of some nodes secrecy is hard to guarantee.

5.5.2. Eavesdropping/MitM, #4

We cannot do much about the vector in this case either. But we can alter the vulnerability a lot. If we make sure to encrypt all data packets before sending them, eavesdropping should be impossible. If we set up a proper data connection with per-session key exchange we should be able to prevent replay attacks as well. And with digital signing of the data contents MitM should be unable to alter origin or contents without it being visible.

This mitigation will take care of problem #1 as well.
5.5.3. Reading welder data from backend, #7

As for physical access, we can recommend companies to keep their local backends in a room where only authorized operators have access. We can also encrypt data before it is stored on the backends. That will make it harder, but not impossible, to get the data.

Since the program must be able to show the data to legitimate operators, it has to be unencrypted in the memory of the backend at some point. We could make sure to run the backends on a trusted computing platform, but that is beyond the scope of this thesis.

Limiting data access could also be done with standard operating system means. In Linux, that means changing the chmod flags on database files, so that only operators can read them.

We propose two mitigations: for non-physical security: make sure the database file is readable only by the operators who need to read the data. For physical security, make sure the room is accessible only to verified operators, and install a security camera to deter attackers.

5.5.4. Altering configuration or software on gateway or leaf physically, #3

This one is kind of tricky. We can make it harder to accomplish, but we cannot limit the impact. Every time the configuration or software is changed through legitimate means, we can sign the new version digitally with the local private key, or with the backend private key responsible for the software version. That will ensure that the files’ integrity is complete.

The one key piece of data that should resist tampering or reading is the private key. It is used to guarantee the integrity of both origin and contents of data coming from the leaf or gateway. If it falls into the wrong hands, an attacker can seem to be a trusted network node. There are a few solutions to improve the physical security of key pieces of information like this, such as the YubiKey (Yubico, 2012) or RSA SecurID (EMC Corporation, 2012).

Only with a secure trusted computing environment can we be sure that our safeguards will not be sidestepped, manipulated or plain removed. Trusted computing is, once again, deemed beyond the scope of this thesis.

The mitigation suggested is therefore the use of a smart card to store the private key, and signing of binaries and configurations to avoid alterations.

5.5.5. Disrupting the delivery of alarms, #8

If an agent has physical access to either a gateway, the originating leaf or the backend showing alarms, then disrupting the system is fairly easy. All that is required is to pull the power plug. To guard against this, some kind of UPS (uninterruptible power supply) system could be employed. That is considered overkill and beyond the scope of this thesis. Another way is to limit access at the location, or to have security cameras monitor the equipment.

Another vector is the external link. If it can be used to overload the gateway, then service could be interrupted for a while. This attack can be an annoyance, but is unreliable for the attacker.
A third attack vector is the ability to cause radio interference, severing the WLAN link. If the factory has a noisy radio environment, interference can be a real problem. In such cases, WLAN will have to be replaced by shielded cabling.

Depending on how the application turns out, there can be some simple ways of causing the gateway to perform high intensity calculations, which would put it out of play. But that is something that must be analyzed later in the process, when there is actual software to test. An example of a similar attack would be the slowloris attack that came to light a couple of years ago, which tricked webservers into using up too much system resources and become unstable (Hansen, 2009).

5.5.6. Summary

The fixes were performed on the artifacts earlier produced. They mainly forced new constraints on the architecture, especially in regards to cryptography. Some of the fixes were on the form of recommendations for the company installing the hardware. This was to deal with purposeful sabotage or at-location-hacking.

5.6. Architecture alterations

We summarized and combined the mitigations in the form of architecture alterations. They consisted of three parts: certificates, architectural requirements and physical requirements. They were as follows:

5.6.1. Certificates

In the new architecture, it has become important to deal with identity. A part of that is to either introduce a system of shared secrets or public/private key pairs. A standardized way of doing this is certificates. We propose a certificate chain like this:

- Semcon Root CA
  - Gateway CA
    - Gw1.Semcon.com
    - Gw2.Semcon.com
  - Backend CA
    - Be1.Semcon.com
    - Be2.Semcon.com
  - Leaf CA
    - Le1.Semcon.com
    - Le2.Semcon.com
    - Le3.Semcon.com
  - Jar CA
    - Jar cert

If such a chain is used, every node in the network will be pre-seeded with all of the mentioned CA’s and also a cert along with that cert’s private key. That way, each node would be able to authenticate other nodes in the network and sign their own messages.
5.6.2. Architectural requirements

Using the aforementioned certificate structure, we can use SASL over TSL to provide a secure transport client to server. That will help with most of our mitigations. But one thing is not covered yet: mitigation #3. For that, we need to sign software and configurations as they are sent from the backend. XMPP does not provide a finished system for that, so we have to implement it.

What it means is that before sending out a configuration or software, it must be signed by the backend. After being sent to its target device, that device will have to unpack the software/configuration and compare it to the signature. This self-check has to occur every time the system starts up. The signature and the certificate used to create it also have to be stored on the target device.

5.6.3. Physical requirements

In the second version of the architecture, a few new requirements have been put on physical security. Since the devices will be operating inside an environment where a lot of people have access, we need to consider the fact that a lot of physical attacks could circumvent software mechanisms. The most important requirements are as follows (in order of importance):

1. If the location has a very noisy radio environment, use shielded CAT-5 cables instead of WLAN.
2. Use a secure hardware to store identifying data (private keys). I.e. a smart-card.
3. Keep local backends in a room where only operators have access.
4. Install security cameras to deter tampering.
5. To prevent service loss when power fails, put all nodes on a UPS.

In order to limit the scope of this thesis, the physical requirements have to remain theoretical. But they were nevertheless as important according to the ALE.
6. Implementation

The software implementation for the scenario in this thesis was dubbed XMPLary. It consists of four different pieces of software. These are called “backend”, “gateway”, “common” and “leaf”, or by their project names “xmpback”, “xmpgate”, “xmpcommon” and “xmpleaf”. Development has been done in java in eclipse. During the development phase, testing was done as three separate processes located on the same computer, instead of single processes on three different computers.

Each of the four pieces of software will be described here.

6.1. General implementation layout

The implementation of XMPLary focused largely on message passing and database management. Upon starting an instance of XMPLary, the code will register a set of strategies for dealing with incoming messages. These strategies look a bit different, but most of them implement a sort of observer pattern. But instead of registering listener objects, they register listener classes. In turn, these classes also implement the command class of the command pattern.

It might seem a bit messy at first, but what it accomplishes is that upon receiving a message, all strategies will be asked if they want to handle that message and if they do, a command will be added to the command queue. The reasons for implementing the command pattern are threefold: unified logging and exception management, thread management and queue management.

Unified logging was important for debugging and implementation purposes. Unified exception management was also largely a debugging and implementation issue, but also something mandated by the XMPP library Smack (exceptions should not be raised in the packet listener thread, and code running in it should be short-lived). Thread management was important due to data consistency issues. SQLite and ormlite are very sensitive to concurrency and were prone to crash if accessed and/or altered concurrently; moving all command execution to an XMPCommandRunner thread had the bonus of alleviating any such issues. Queue management was important because of the risk of buffer overruns unless commands are processed in the right order.

Every different command has been given a specific priority, which mandates what place it will take in the queue. This can, however, be altered at run-time for commands that have variable priority.

Message passing is focused around the framework XMPP, as mentioned previously. In practice, this means that Smack played a big part in how the code is laid out. It also means that the way the user ("operator") interacts with the programs is also done via XMPP. Interaction is done in two ways: a chat channel, and a direct conversation between operator and backend.

The chat channel is called YOLO and is used to output some status messages and to announce presence; the latter in the form of a user list.
There is a list of the message types involved in the system in Appendix F. A list of the commands running on the XMPCommandRunner thread is available in Appendix G. The third party software used to make this program a reality is available in Appendix H, which also lists the licenses they are used under.

### 6.2. XMPGate

The gateway is the central node of the XMPLary network. It acts as a switchboard, routing messages with other nodes as target. The only exception is the operator, who communicates directly with the xmpback. It registers to the server with the name “gateway” and should be unique on the server. The gateway has a list of currently registered backends and leaves.

Leaves will send data points to the gateway when new data is available, in a way that assumes a high-speed low latency network. These data points will be gathered on the gateway database, and periodically sent out to all registered backends. Some kinds of messages, for new status updates and alarms, are multicast from the leaves via the gateway. That means that the leaves are unaware of the backends, but specifies the gateway as the message’s intended target. Other messages, such as responses to GetDump requests, are sent to a specific target backend.

Since encryption is done node-to-node, not end-to-end, the gateway needs to be able to decrypt all messages coming to it.

XMPGate will receive data points in a somewhat continuous stream from the leaves and gather them up. At a set interval they will be packed together and sent to the registered backends. This was dubbed *Periodic Dispatch*.

### 6.3. XMPLef

The XMPLef program is what is supposed to run on the welder nodes. In order to emulate a welder, it also runs a WeldingThread that runs a state machine simulation of a welder.

#### 6.3.1. Welder model

Since we have only a passing knowledge of the intricacies of welding, we have constructed a model of a welder with a few variables. The point is not to emulate the inner workings of an actual welder, but to produce a data set similar in shape and size to what a welder would do. The data will be used as a sort of place holder, to make sure the system could fulfill the scenario.

The model will update variables at least once every 10 seconds, and will go between a set of different states. For every tick, these variables will be updated:

- Temperature (°C)
- Fuel drain (litres/hr)
- Fuel remaining (litres)
- Weld speed (mm/sec)
- Cheeseburgers consumed (/hour)
- Voodoo Magic (souls/hour)
The model does it best to simulate the variables. Most of them should be self-explanatory, but cheeseburgers changes every few hours or so, to emulate a switch of supervisor personnel. Voodoo Magic is probably not involved in a real implementation, but used as a placeholder to be able to draw graphs of interesting behavior.

A simple state machine is running the model. States available:

- Running
- Refueling
- Coolingdown
- Stopped
- Await_Refuel

An illustration of how the variables interact can be found in section 6.4.1.

6.4. XMPBack

The backend. Is currently run on the same computer as the operator, which does not have to be the case. Serves as both a passive node that receives and saves a backlog of data points/alarms/status updates, and an active node that the operator can interact with. It can respond to some commands from the operator. These commands are listed in Appendix E.

6.4.1. Generated graphs

Here is a set of graphs generated by the DumpData method. These are meant to exemplify how the system reacts when not refilled with new fuel. To build these graphs, the data points of two welders that have been left running for about 20 minutes were used. They are called leaf-20 and leaf-35. It is possible to get data from all connected leaves or from just one, depending on if you want to get an overview or study an instance closer.
6.5. Key scheme

XMPLary uses a set of X509v3 certificates for keys. During compilation and build, each instance of the program gets an appropriate key that identifies them and the type of node they
act as. We have used the open source program XCA to create a CA and certificates to use, just like we described in 5.6.1. Here is a picture from the program.

Figure 10: XCA Certificate management

Certificates are laid out as previously discussed, with the addition of a “XMPLary jar CA” and a “XMPLary Jar” jar. They are explained in section 7.3.

6.6. Cryptography

In 12.18 we can see the raw message format of an alarm:

```json
{"iv":"hPNOGaXdw1lJHN2uKPF0QA==","response_to_id":0,"contents":"3F1lR3kkLJg83TV0j4kijlJpMoMJH2D6ku3TBRYEVZL8ggy6fn6Re79/VN7xNczQVlNDEIFedBC7JrNd5+t0Yq+iDTmJ8oUcj2dXWuIqAiQIf5pxAIqSp0UIZ4r+Fvaz","origin":"leaf-20","original":2957,"target":"backend-3","type":"Alarm","key":"WrO/Dlvm2wR6lTrXxi19OtwdQTlpBoAhbid9VBnDd+q7+/IC627WUN4ZtW3fVRQLXD0hYTia6n37QEkaxTnnkswjljBhHS0E5H6luMOPPEEly2nWTDUFPbIGGFoIf6mMmTq86ZcPofbwqEzO4c8JKXnY3hviIH2k00By3Hqkm0=","signature":"3311d322d1ae2953a4e003c24eaba9b056ec48f107a4098c25d32782dd1e0d8ed98e7ba064b620b0629b95c96d42776809c3763c42ce0719b81efff415281eb3e0cd741079a7aaaee699eb5c7122ad39f3944716ea7750f476ec6924bfb345bf4a965be1a9bba465b95e79fa8f3fe8dc1ad451ca9780d64cc0f421d4ec02"}
```

Here the message is encrypted and signed. The encryption affects four fields and the signing only one. The simplest one is signing, so we’ll explain that first.
6.6.1. Signing

Signing is done end-to-end. That is to say, if for instance leaf-20 wants to send a message to backend-3, it will sign its message with its own private key, and let backend-3 verify it with leaf-20’s public key. Signing is done as a typical RSA asymmetric signature. We can see in XMPMessage.java#sign:

```
signature = XMPCrypt.sign(contents + origin.getName() + getOriginalId());
```

Where the name would be “leaf-20” and originalId would be an integer that is unique for that message and node. A reason for including the Id is to make every signature unique and avoid replay attacks. Another reason is to avoid data duplication.

There are a few things to note about this. One is that the receiving party will need a few pieces of information to verify the message: the unencrypted contents, the origin name, the original Id and the public key of the sender.

When a node notices that it lacks the required public key to verify a signature, it will request for a handshake to be performed. The handshake is a simple RequestRegistration command, answered with a Registration command.

For signing, we use the method “SHA256WithRSAEncryption” provided by BouncyCastle.

6.6.2. Encryption

Encryption is done node-to-node. That is to say, if leaf-20 sends backend-5 a message, it will be sent via the gateway, which decrypts it and then encrypts it before passing it along. Mainly, the reason for this is that decryption is a requirement for verification.

The encryption scheme used is a two-step process. At its core the scheme is an asymmetric encryption. The problem is, however, that the asymmetric encryption used does not take as input information larger than the key used. There are ways to get around this, of course, but another problem is that it can become prohibitively slow, especially for the low-end hardware targeted by this application. To alleviate this, we use a symmetric encryption for the potentially large pieces of data in the contents field.

So our encryption process looks like this:

- Create a random one-time key
- Encrypt contents symmetrically with the key
- Encrypt the key asymmetrically with the target receiver’s public key
- Send message along with the encrypted key

And on the other end, decryption looks like this:

- Receive message along with encrypted key
- Decrypt the key asymmetrically with my private key
- Decrypt contents of the message symmetrically with the key
For asymmetric encryption we use RSA with OAEP padding, and for symmetric encryption AES in CBC mode with PKC5 padding with a 256 bit key. The IV for the block chaining is generated randomly for each message, and is sent along as well.

6.7. File layout

The XMPLary project consists of five folders: certs, xmpback, xmpcommon, xmpgate and xampleaf.

The certs folder contains everything pertaining to certificates. That means an exported version of the CA certs and EE certs, the EE cert’s private keys, and the XCA database. XCA databases are secured with a password, in this case “starwars”. We are unsure how strong this password protects the file; it is safe to assume the file should be kept as privately as possible despite this protection.

Xmpback, xmpgate and xampleaf each hold a java project responding to each node type each. They share a similar structure, even if the code running differs. There is a “src” folder that holds the source code, a “bin” folder generated by eclipse during development, a “build” file that holds the output of the build if it has been run, a “files” folder for auxiliary files and one or more xmpbuild.xml files.

The “files” folder gets populated during build and run, but should at the very least contain a “conf.properties” to be able to run the program.

In xmpcommon all the files shared between the projects exist. It has a “lib” folder with all the 3rd party libraries imported by the runnable projects. The “files” folder is also interesting, with the log configurations, “run.sh” and keystore files, which will be explained in section 6.5.

The files have been made available at https://github.com/lulzmachine/xmplary for posterity.
7. Deployment

The hardware we used to deploy XMPLary was one BeagleBone and two Raspberry Pis. They are both similar pieces of hardware, used as a sort of middle ground between a full-blown computer and an embedded chip. BeagleBones are typically used as a development environment, before building a stripped-down production chip such as something based on Olimex. The BeagleBones and Raspberry Pis have ARM-hf processors around 700 MHz and 256mb RAM each.

In the architecture laid out in the planning phase, we would have WiFi between the systems at the factory and WCDMA to connect to the separate domain. In order to streamline development, an Ethernet cable with a standard-configuration switch has been used. The hardware systems lack WiFi out-of-the-box, although this could probably have been achieved with a USB WiFi modem. The same goes for WCDMA.

We were also unable to get a hold of, and set up, some kind of hardware solution for keeping secrets. As mentioned in the end of the first iteration of RMF, we suggest a USB smart card to keep sensitive information secret even though deployment is done in a hostile environment. As it stands, an attacker could simply pull the SD card from our systems and read/alter data freely.

The gateway and accompanying XMPP server was put on the BeagleBone, and each of the Raspberry Pis have been seeded with an instance of the xmpleaf program. They are able to run many instances of xmpleaf each, but only one has been put as an autostart cron job.

The realities of deploying software also brought forth a few obstacles such as configuration management and auto starting.

This is also part of the thesis report that gets most bogged down in practicalities and technicalities.

7.1. Deployment issues

To paraphrase Helmuth von Moltke, *no plan survives contact with deployment*. Deploying XMPLary was no exception. The main problem was getting the programs to run on such slim hardware. We devised ways to debug and troubleshoot, which led us to new mitigations.

7.1.1. Memory management

Both BeagleBone and Raspberry Pi provide 256 megabytes of memory. On this, we want to fit a Debian operating system, the Openfire server and XMPLary. Early tests showed us that this was unlikely to happen.

There are mainly two ways of measuring occupied memory: VIRT and RES. VIRT stands for Virtual and is the sum of how much memory the process currently uses and has reserved for future use. RES stands for Resident is a more accurate description of how much memory the process is actually using (Sumanariu, 2010).

Upon starting, Openfire has about 352mb VIRT reserved, and uses about 47mb. When trying to start XMPLary on top of that, XMPLary got eaten by Linux’ out-of-memory killer.
XMPLary, it turns out, was not perfect either. An inherent problem of living within java’s VM is that you have no direct control of how memory is managed. The only control you really have it is the ability to set a maximum memory allowance for the whole VM. The rest; allocations, deallocations, levels of caching etc. are at best deducible and at worst unpredictable. Initially, xmpgate would allocate 408mb VIRT. This does not have to be a problem as you can allocate more virtual memory than what is physically accessible, but the 188mb allocated as RES definitely is.

Normally XMPLary is run on the openjdk-7-jdk package, which does not have the jmap tool to get heap dumps. Thankfully Oracle just released a FX ARM Preview version of java able to run on the ARM-hf architecture of our embedded systems (Oracle, 2013). Using this, we were able to get a heap dump to analyze (using jmap). It turns out the heap only took about 5.6mb, which led us to confusion regarding what really used up all the space. It is also unclear if it actually just used 5.6mb on oracle’s VM or it is just the misrepresentation inherent in measuring memory used “inside” a VM and “outside” the VM. But since we wrote XMPLary to target openjdk, we did not investigate this further.

Running profiling on windows gave us a dump like this:

![Memory Statistics](image)

It pointed a finger clearly toward the database.

**7.1.2. Mitigations**

The solution to the server problem was to switch to another server called prosody. As prosody was available in aptitude, the decision to switch was not hard. After some configuration we had the same kind of setup with a much slimmer memory footprint: 7mb VIRT and 3.7mb reserved upon startup. It is hard to measure memory due to a lot of complications with shared libraries et cetera, but the difference was large enough to make up for any such uncertainty.

When it comes to the problem with XMPLary, it was still unclear whether the problem was ormlite or SQLite, but we hoped that switching SQLite to another database would help. Since
ormlite abstracts the database layer to a great extent, switching SQLite to HSQLDB was relatively straightforward, and only required slight implementation changes to how serialized types are saved into the database.

It helped: running an XMPLary instance after changing from SQLite to HSQLDB and trimming the java launch parameters uses about 44mb RES with an empty database. The packaged jar also went from 35mb to around 28mb, since sqlite-jdbc comes with a lot of native drivers that we cannot use for ARM-hf anyway.

7.2. Configuration management

Handling configurations and building XMPLary into a deployable solution also became an issue we had to solve. In our case, we needed to have around 7 instances to test with at a time, and moving files and rebuilding by hand each time became unfeasible. We therefore made a set of ant-scripts to automate the process. It was also expanded to handle writing configuration files automatically.

The ant-script is split into two parts: one in xmpcommon and one part for each instance to be built. We’ll use a config used for the gateway to exemplify. It is available in Appendix L.

The common one is also available there. It will generate the jar file, copy the relevant certificates and private keys, generate a configuration file, remember what the current build’s number is and copy a few auxiliary files. It will also sign the jar file, to make any subsequent tampering detectable.

Also, to make sure XMPLary is run with the right settings, a “run.sh” shell file was created. It lives in xmpcommon/files/run.sh, is copied automatically by the build script and looks like this:

```
#!/bin/bash
/usr/lib/jvm/java-7-openjdk-armhf/bin/java -Xms128m -Xmx128m -XX:PermSize=64m -XX:MaxPermSize=64m -jar jar/xmplary.jar
```

The logging system also has a configuration file, which is copied into the build directory by the script. It puts all logging messages of level INFO or higher into a log file, and everything with a level of ERROR or higher in the console as well. It looks like this:

```
log4j.rootLogger=DEBUG, CA, SO
log4j.appender.CA=org.apache.log4j.RollingFileAppender
log4j.appender.CA.File=files/log.log
log4j.appender.CA.MaxFileSize=30MB
log4j.appender.CA.MaxBackupIndex=4
log4j.appender.CA.layout=org.apache.log4j.PatternLayout
log4j.appender.CA.layout.ConversionPattern=%d{HH:mm:ss.SSS} [%t:%5p] (%F:%L) %n => %m%n
log4j.appender.SO=org.apache.log4j.ConsoleAppender
log4j.appender.SO.Threshold = ERROR
log4j.appender.SO.layout=org.apache.log4j.PatternLayout
```
The config file of the prosody server is available as Appendix K. Notable features: the security measures of XMPP have been deactivated entirely. The “muc” module has been activated to accommodate the YOLO chat room.

7.3. Signing code

As mentioned, jars are signed after creation. That means that the jar gets a cryptographic hash assigned to it that corresponds to its contents. Using the java tool “jarsigner –verify”, it is possible to determine whether the jar file has been tampered with or not. For the signing to work, a certificate had to be created and compiled into a format that java understands; i.e. a keystore. Java comes with a tool to manage keystores – called keytool – that can easily generate self-signed certificates. But since we already had a CA chain set up, we wanted to use it. So we exported a chain of certificates leading down to the “XMPLary Jar” certificate, along with its private key into a PKCS12 package. This is important since keytool can treat PKCS12 packages as keystores, but is unable to import private keys and key chains in any other sensible way. We detailed how this process looks in practice in Appendix J.

7.4. Startup

Debian, the operating system on all our systems, comes with a system called anachron, which runs cron jobs. That is to say, lists of programs set to execute recurringly. In order to start XMPLary, we simply added a line in the crontab like this:

```
@reboot Debian cd /home/Debian/build && ./run.sh >> runlog.log
```

Prosody, our XMPP server, is started a bit earlier than crontab execution. Its startup is governed by its presence in /etc/init.d/ and /etc/rcX.d which was automatically configured by aptitude.

7.5. Time management

Time management was an unforeseen and unsolved issue we encountered. At its core, the problem occurs because of the lack of reliable hardware clocks on the Raspberries (Chamberlain, 2012) and BeagleBones (Coley, 2012). Normally overcome with internet connectivity and automatically updating clocks, our system fails to solve the problem.

Mainly the issue is that the systems wake up thinking that the current date is somewhere around the year 2000, and while this can be altered with “sudo date –s 2013-01-31” there is no hardware clock to make the change permanent. Even if we could overcome this obstacle on some devices, the system depends on all nodes to be able to work with the certificates.

Our certificates were initially set to expire within a couple of ears from 2013, but we had to make concessions for our deployment. Our solution was simply to create new “generous” certificates with a life span from 1980 to 2040. Another solution would be to have a reliable time counter for the gateway and put an auto-update system like NTP on the systems.
8. Risk management, round two

Implementation and deployment concludes our linear waterfall-like realization stretch. In order to show in practice what we said earlier about risk management being an iterative process, we returned to RMF to perform a second round. Due to time constraints, however, the mitigations proposed by it will not be implemented but instead left as future work.

8.1. Identify and link the business and technical risks

Here we need to update relevant artifacts. Assets and security requirements belong to the first step of RMF, which should only be done once. That makes sense, since they are unchanged. In some cases assets will have been changed by implementation details, but not in our case. Attack trees, however, could stand to be refined at this point. Here are the updated versions:

![Attack tree: Implant/Alter data version 2](image-url)

Figure 11: Attack tree: Implant/Alter data version 2
Figure 12: Attack tree: Disrupt alarms version 2
8.2. Synthesize and rank the risks

In order to identify risks we need to do four things here:

- Start off with the risks mentioned earlier
- Change them, depending on how well mitigations have been implemented
- Add new risks that have been encountered or introduced in implementation
- Search the artifacts for new risks

We’ll start off with the earlier ones, modified depending on how implementation of the system and of the mitigations has been performed. After that, a few new ones might be introduced. Parts of the risks that have been mitigated are struck through, and parts added appear as *italicized.*

8.2.1. Registering a fake backend

Num: 1
**Impact:** Registering a fake backend to a gateway is a way to start several attacks. If someone is able to register a fake backend, they would be able to passively listen to get all welder data and alarms. They would also be able to send software and configuration data, which is potentially disastrous, and would be able to affect all assets.

**Vulnerability:** No authentication required to add backends. Authentication with a valid X509 certificate required. Certificate chain is not, however, verified.

**Vectors:** Via either Li2 or Li1, posing as a legitimate backend, or by altering the gateway configuration database.

8.2.2. Registering a fake welder

Num: 2

**Impact:** Registering a fake welder is a way to start a few attacks. If someone achieves this, they would be able to implant fake data and fake alarms. This could affect the trustworthiness of the system and its alarms, potentially leading to operators ignoring real alarms.

**Vulnerability:** No authentication required to add welders. Authentication with a valid X509 certificate required. Certificate chain is not, however, verified.

**Vectors:** Via either Li2 or Li1, posing as a legitimate welder, or by altering the gateway configuration database.

8.2.3. Altering configuration or software on gateway or leaf physically

Num: 3

**Impact:** If someone is able to alter configuration or software on a gateway or a leaf by means of physical access, they can deal a lot of damage. The welders themselves could be at risk, and everything implied by the section 5.4.2 as well (esp. with regards to implanting fake data). The manufacturing results could also be at risk. Since there are a lot of people that have physical access to the devices, they were considered exposed.

**Vulnerability:** Physical access. Data stored unsigned and in plaintext. Data is stored unsigned and in plaintext, but software is signed. Signature consistency is not, however, verified.

**Vectors:** If an agent has physical access to the devices, they alter the data as it resides on the SSD. If the device has means of physical access such as USB-tty (which is the case with BeagleBone), that is also an option.

8.2.4. Eavesdropping or performing Man-in-the-Middle on a link

Num: 4

**Vectors:** An ill-meaning agent eavesdropping or performing Man-in-the-Middle on a link, either by being physically present to perform it on Li2 or Li1 or remotely to do it on Li1 could cause some harm as well.
**Vulnerability:** No encryption or signing of data during transfer. Data is encrypted node-to-node and signed end-to-end. The X509 certificate involved in encryption should belong to a specific certificate chain, but this is not enforced.

The communication backend to operator is done over XMPP unencrypted at the moment. Even if backend and operator are located on the same workstation, communication is done via the XMPP server.

**Impact:** If eavesdropping is possible, an attacker can gather the data flowing through the system, which means alarms, welding data, configurations and software. If they can do Man-in-the-Middle, they can alter all the aforementioned data as well, which would imply all damages mentioned by 5.4.3 as well. A well-crafted Man-in-the-Middle could also potentially be harder to detect than the physical disturbance needed to perform 5.4.3.

### 8.2.5. Altering welder data on a gateway or leaf physically

**Vectors:** If an agent has physical access to the devices, they alter the data as it resides on the SSD. If the device has means of physical access such as USB (which is the case with BeagleBone), that is also an option.

**Vulnerability:** Physical access. Data stored unsigned and in plaintext.

**Impact:** If they can do that, they can manipulate the data before it is being sent to the backend or gateway. This could lead to damages to various assets: the welder data will be damaged, the welder itself might not be treated with the right kind of maintenance or repair and trustworthiness of other data may decline.

### 8.2.6. Reading welder data from a gateway or leaf physically

**Vectors:** If an agent can access the devices by the same means mentioned in section 5.4.5, reading data from the SSD is a simple task.

**Vulnerability:** Physical access. Data stored unsigned and in plaintext.

**Impact:** Altering the running program to output its data in another form than the standard one is also possible, by means of reverse-engineering.

### 8.2.7. Reading welder data from a backend

**Impact:** If an attacker can access a backend, they can read the welder data. The backend is where the data will be presented to the end-user, so it has to be available in a non-encrypted format.

**Vulnerability:** Physical access. Data stored unsigned and in plaintext.
**Vectors:** A backend can be accessed either remotely via something like SSH, or physically if local security measures are inadequate. This risk pertains to legitimate backends being accessed in an illegitimate way.

**8.2.8. Disrupting the delivery of alarms/status updates/data**

Num: 8

**Impact:** If an attacker can disrupt the delivery of alarms for a period of time, they can keep operators in the dark regarding the status of the welding apparatus.

**Vulnerability:** Dependency on stable electricity. Sensitivity to radio interference. Dependency on other network nodes. *No automatic re-connection, which means a temporary disturbance is enough. No verification of message delivery.*

**Vectors:** Assuming the local backend is not connected insecurely to an external network, this attack has two possible vectors: local physical disruption and remotely overloading the gateway. Local physical disruption is very simple in this case: just pull a power plug.

**8.2.9. Denial of service**

Num: 9 (new)

**Impact:** An attacker can permanently overflow one of the nodes in a typical denial of service-attack.

**Vulnerability:** All messages that an XMPLary client reads or writes is saved into a database. Sending random text messages to a client will make the database too heavy to be runnable by the embedded system.

**Vectors:** Either over the XMPP network or physical tampering.

**8.2.10. Summary**

Again, the risks are summarized and ranked using ALE. AROs and SLEs have been updated to reflect the new situation. Figure 14 shown below is a summary of the risks we have taken into account, sorted on the highest ALE.

<table>
<thead>
<tr>
<th>Risk</th>
<th>#</th>
<th>ARO</th>
<th>SLE</th>
<th>ALE (=ARO*SLE)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Reg. fake backend</td>
<td>1</td>
<td>7</td>
<td>2 000 000 kr</td>
<td>14 000 000 kr</td>
</tr>
<tr>
<td>Denial of service</td>
<td>9</td>
<td>8</td>
<td>1 000 000 kr</td>
<td>8 000 000 kr</td>
</tr>
<tr>
<td>Eavesdropping/MitM</td>
<td>4</td>
<td>4</td>
<td>2 000 000 kr</td>
<td>8 000 000 kr</td>
</tr>
<tr>
<td>Disrupt delivery of alarms/status</td>
<td>8</td>
<td>15</td>
<td>500 000 kr</td>
<td>7 500 000 kr</td>
</tr>
<tr>
<td>Reading welder data from backend</td>
<td>7</td>
<td>10</td>
<td>500 000 kr</td>
<td>5 000 000 kr</td>
</tr>
<tr>
<td>Altering config/software on gate</td>
<td>3</td>
<td>4</td>
<td>500 000 kr</td>
<td>2 000 000 kr</td>
</tr>
<tr>
<td>Alter welder data physically</td>
<td>5</td>
<td>2</td>
<td>400 000 kr</td>
<td>800 000 kr</td>
</tr>
<tr>
<td>Reg. Fake welder</td>
<td>2</td>
<td>4</td>
<td>50 000 kr</td>
<td>200 000 kr</td>
</tr>
<tr>
<td>Reading welder data physically</td>
<td>6</td>
<td>2</td>
<td>30 000 kr</td>
<td>60 000 kr</td>
</tr>
</tbody>
</table>

Figure 14: Risk table, sorted on ALE, version 2
8.3. Define a risk mitigation strategy

We’ll choose the top four risks and find a mitigation strategy for them. Some of the risks remain the same as in the earlier iteration, while a few are new. The reason is twofold: the mitigations performed have not been perfectly implemented, and in implementation new vulnerabilities have been found and/or introduced. Overall, the ALE of the project has been increased, while the top offenders have decreased. It is hard to say clearly how good or bad this is.

8.3.1. Register fake backend, #1

Registering a fake backend remains a top scorer for ALE. Even though it was present in the last iteration of RMF and therefore had mitigations suggested, these have not been implemented perfectly. The main reason is that the vulnerability that pulled the ARO up still exists. While authentication is required, the only requirement is that it is done with a valid X.509 certificate. Anyone could craft a self-signed certificate and add a backend.

Mitigations: The fix for this is obvious: upon receiving a request to add a new backend, verify that the certificate offered really is an EE certificate from the “XMPLary Backend CA” chain. Also perform this check on system startup, to purge the database of malicious backends.

8.3.2. Disrupt delivery of alarms/status/data, #8

The main reason for number eight’s appearance on the top four is the large vulnerability. Currently the system is prone to sending data to offline nodes without verifying that messages have truly been delivered. The system does not try to reconnect when experiencing temporary network outages, which is also a problem. So to summarize: the main problem here was accidental threats, not intentional threats.

Mitigations: The presentation layer used between the nodes must be more reliable. Adding a message delivery verification mechanism would be needed. There are different way to do this; either manually or by using some of the extensions of XMPP (Isode, 2011) (XMPP Standards Foundation, 2011). It also needs support in the XMPLary code, such as automatically reconnecting to the server and some kind of re-try-delivery-command.

8.3.3. Denial of service, #9

Number nine is a new risk that has become apparent during implementation. It might seem very similar to number eight, but with the important difference of being a permanent outage. Since the embedded systems where the system resides are very weak (especially in RAM, which is something java loves to use up), bloating the database is very easy. Further work is needed to make sure XMPLary can be kept alive as a long-running reliable system. Even without a dedicated attacker, the everlasting stream of produced data along with the fact that all data is saved will invariably lead to this threat occurring.

Mitigations: XMPLary will have to automatically prune old records from the database, in order to maintain a small memory footprint. For instance removing messages from the database when they have been transmitted and acknowledged, and removing data points from leaves when they have been transmitted. Of course this introduces a new problem: when to
remove something and when to keep it. For instance if a backend connects to a gateway, it could hope to get a backlog of old data points. But if the gateway lives on a small embedded system, it is not able to keep them around for a long time. There are others issues with this as well, which is why we will leave this as future work.

8.3.4. Eavesdropping/MitM, #4

MitMs and eavesdroppers are by nature rather hard problems to get rid of. Cryptography is usually seen as the main protection against this threat, which is probably true. But just protecting your data with encryption is not enough if it is not done correctly. For instance in Xmplary there is a public key infrastructure that supports the exchange of public keys to encrypt and sign data, but it lacks a good way to enforce trust. An attacker could easily create a self-signed certificate and claim to be a valid node; the rest of the system has no reason to doubt her.

**Mitigations:** Add a way to enforce consistency of the certificate chain. This might be easier said than done, in part due to the fact that there is no hardware clock; all certificates in the chain must be verifiable. As it stands, the age-checking of certificates can be expected to be erratic. Also a system for certificate revocation should be implemented.

9. Discussion

In the beginning we set out to answer a few questions:

- How can we identify the threats and risks involved in creating interconnected embedded systems?
  - What is a suitable threat model? What is a suitable risk set?
- What is a reasonable level of risk mitigation?
- What kinds of architectures are able to mitigate these risks to a reasonable level?
- How can we implement such an architecture in the real world?

The methods and scope were still a bit vague at the time but we knew we wanted to create a system, with all the problems and solutions that entailed. Scope was a bit defensively expressed, saying we will stay away from implementation as far as possible, seeing as implementation tends to be a time sink if not managed. But as it turned out we found a lot of time to make something useful.

We believe we have answered the questions posed in the beginning. If not in black and white in the report yet, then at least between the lines of the java code and risk management. We have striven to be more precise here, however.

9.1. How can we identify the threats and risks involved in creating interconnected embedded systems?

By using a combination of tools. Throughout the thesis we have worked actively to find problems, and solve them to as good a degree as possible. Identifying assets, drawing attack trees, writing requirements, writing code, understanding best practices and reading abstract theories have all helped. One thing we would have loved to do (but which was deemed out of scope early on) is to have time for some penetration testing. But the project is still in an early
enough state that we can find obvious security problems without testing, so this has not been an actual hindrance.

We were surprised by how accurate the initial risk set was. We had half expected it to seem naïve after getting our hands dirty in the implementation. Between RMF 1 and RMF 2 we did implementation, and in the second round of RMF the risks remained largely the same, with the addition of only one new risk and modifications of about five others.

The bulk of the risks came writing security requirements from the information assets we found on the system. This is probably something many people do intuitively, thinking “I have login credentials. How can I keep them secure?”, but doing it more extensively by giving them scores for C, I and A is probably a very good thing to do. It does not have to take a lot of time, but gave us insight enough to flesh out most of our risks.

9.2. What is a reasonable level of risk mitigation?

It depends. We never did define a “goal” to try to reach with RMF, but we could clearly see where we were with ALE. What a reasonable level means depends on how many rounds of RMF you have time to go through. You do not have to decide in the beginning of your project what your security will look like in 6 months, but keep the process iterative and you can decide a month or so in advance where you want to stop.

We kept two different processes running throughout or process: a linear waterfall model for implementation, and an iterative risk management process. If you were to use an iterative model for implementation, such as SCRUM, it is likely that you can tie together RMF and SCRUM in a neat way. SCRUM also avoids answering “what is a reasonable level of done” too early, but pushes for an answer at the end of each iteration.

If you take the costs produced by ALE at face value, you could use them to calculate precisely how much you should work on mitigations before it becomes more expensive than beneficial.

9.3. What kinds of architectures are able to mitigate these risks to a reasonable level?

Carefully weighing the options and adapting to the realities of deployment is probably a general enough answer. For some parts of the architecture (such as software layout), experience and knowledge of software patterns aided. For others (such as link layer), an evaluation technique used by the US Army was the answer.

9.4. How can we implement such an architecture in the real world?

We feel we this has been explained as well. The two parts of this question, “how can we implement such an architecture” and “in the real world” have each been given a main section of the thesis. We actually did not expect the second half of the question to take as much effort as it turned out to do, and still it should be remembered that this is the rosy soft real world of development boards and fully-fledged general purpose Debian Linux. Not the harsh real world of uptime requirements, hand-compiled kernels, radio wave interference and obscure embedded distributions of not-even-Linux.
9.5. Security best practices

In the theory section we laid out a set of security best practices that outline general sound principles of secure software development. While they have not been taken into account directly during development, we took some time to look back of them now. In general, we have followed most of them, most notably

- Securing the weakest link. ALE helps us do this very effectively.
- Least privilege. With our clearly defined node roles, no node can do things that are outside their role.
- Least Common Mechanism. With a lot of the code being shared, mechanisms remain common.
- Never assuming that your secrets are safe. Since our system is to be deployed in an unsecure location, we never trusted that a system based on shared secrets could work.
- Psychological Acceptability. All the system requires is for certificates to updated once every few years.

Things that have not been followed very well:

- Defense in Depth. For instance, we turned XMPPs security features off, relying on only our own (with good reasons). In a future work we could add more redundant levels.
- Reluctance to Trust. Should be have more time this would have been overcome. But for now, a valid X509 certificate is all it takes to authenticate.
- Complete Mediation. We have a handshake (called ‘registration’) mechanism when a connection is established. This is later cached. Perhaps it should be wiped on restart or after some time has passed.
9.6. Scenario

We believe the system we created fulfills the scenario to a fairly high degree. It performs both of the technical requirements, and then some. We can send and receive alarms, status updates (which were not even part of the initial scenario) and data points. The system also allows for setting variables individually on the welders, and for getting debug data from them.

It is kind of lacking in the reliability department. A lot of the reliability-related problems are still of such a fundamental nature that they can be classed as bugs rather than security-related problems. They are however successfully addressed by RMF, which means treating them as security problems is enough. Since reliability is a subclass of Availability, we feel that this should be sufficient.

We think that it is even a good thing, since it puts other security problems such as lack of integrity on the same level. If we were to treat reliability differently from others, there might be a risk that software bugs would be thought of as must fix while the others would be thought of as would be nice to fix. Which would lead to security being “added later”. As we discussed in theory, security is not a feature, but an emergent property of a system. So treating all these problems on the same level probably helps us improve the system as a whole.

We have not fulfilled the security to such a degree that it can be sold, but that was never the intention. The intention was to create a system that could fit in a real-world scenario, but that used placeholder values for emulating welder data. We feel that we have done this.
10. Conclusion

We have dreamt up and implemented a system from scratch, with security as one of the main goals. It took us through a lot of processes, from determining scope, learning about security theory, to coming up with a scenario, to risk management, to project management, further on to architecture evaluation, implementation of a system in software, implementation on different kinds of hardware, through deployment and another iteration of risk management. It has been a long and interesting ride.

The expectations put on us from different sources were differing. The university focused a lot on the processes required to build secure hardware, while the company focused on specific mechanisms. Our report became something of a link between these two worlds. Instead of answering just “what processes should one use” or “what mechanisms should one use”, we answered a question somewhere in between; something like “What processes should one use, and what mechanisms will they provide?”.

We have shown that security can be something other than a gut feeling, that there can be a reliable method. The human element has to be there to make judgments along the way, but having a process helps.

It has also been made clear how a process can be used to build security in from the start. Since security is an emergent property of a system and not a feature, adding it later can be burdensome. We have shown that this methodical process can provide something resembling determinism when building security in.

Another important aspect is the business aspect. The whole reason for this thesis was that companies wanted to create networked embedded systems, and that this created a lot of uncertainty regarding security. We have shown a method that provides a business-level justification for the security process. With the costs of keeping risks being balanced with the costs of mitigating them, it has been made clearer what the reason for spending effort really is, and thus helping in finding a reasonable level of effort to spend on it.
11. Future work

The risk management process has clearly shown us a possible avenue for future work: keep on running RMF until satisfied with the results. At a point not too far into the future, testing should become a part of RMF, in order to find new weaknesses. For instance penetration testing and fuzz testing could be introduced in a future RMF iteration.

If the project were to be continued for long, that would suggest that a real-world purpose has been found for XMPLary. In that case, deployment, requirements and testing for the site would be an important part. For instance, the viability of the link layer in the face of radio interference, reliability of long-running XMPLary instances, audit and physical tampering defenses could be things to examine.

One part of the software system that was disregarded was the ability to send software updates. We reserved bandwidth for it, and actually implemented most of it in the software. But we were unable to get it reliable enough to include in the thesis. A piece of future work could be investigating why it was so unreliable, and to do something about it.

XMPP has been used as a basis for all inter-process and inter-system communication in this thesis. There are many alternatives to this, for instance JMS, MQTT etc. It would be interesting to examine the pros and cons of other technologies.

Other more specific mechanisms that would be interesting to examine are:

- Firewalls
- Switches with hard-coded MACs
- Uninterruptible Power Supplies
- Gateway on a real big computer, serving NTPdates
- Self-pruning systems
- A “backup” type node?
- SSH security (passwords, PGP certificates?)
- Pluggable levels of security
- Integrating operator and backend into one system
12. Bibliography

3GPP, 2012. 3GPP TS 22.060, Sophia-Antipolis Cedex: 3GPP.


Appendix A The CIA of information security

Information security is divided into three basic components, and at its core it strives to provide these three services for an information based system (Bishop, 2004).

12.1. Confidentiality

Confidentiality is usually the first thing that comes to mind when discussing information security. It is the service of keeping certain information secret from other entities. The need to keep information secret comes from all forms of computer use, which is to say governmental use, industry use and personal use. It can be accomplished with an array of means, such as access control mechanisms and cryptographic mechanisms.

It can be important to keep both data and the existence of data confidential. It is also important to note that confidentiality mechanisms rely on other agents to for instance hold encryption keys, hold sensitive data or be reliable in enciphering. Therefore, assumptions and trust are relevant to confidentiality mechanisms.

To exemplify, while a student must be allowed to see their grades (along with the school and relevant institutions), it should not be publically visible to just anyone feeling curious. Any implementation of this must rely on the student trusting the school or a third party to hold the data confidential.

12.2. Integrity

Another service that information security seeks to provide is integrity. It measures how much trust you can place in the trustworthiness of data or the origin of data. A very basic version of providing this service is to ensure the validity of signatures. Forging signatures on documents will break the trustworthiness of the origin of a piece of data. Modifying a contract after it has been agreed upon and signed will break the trustworthiness of the contents of the document. Just as in the case of confidentiality, evaluating the measure of integrity relies on trusting other parties.

There are two kinds of integrity mechanisms: prevention mechanisms and detection mechanisms. The first one is there to prevent a breach of integrity, and the second kind is only concerned with detecting whether integrity has been compromised. For instance, requiring user authentication and authorization prior to letting a user edit data is prevention. Adding something like a cryptographic signature and confirming its validity is a detection mechanism.

In the school grade example, editing the grades in an unauthorized way will create a breach of integrity. Having a high degree of integrity relies on both the student and other institutions trusting the school to not tamper with the grades.

12.3. Availability

The last security service is called availability. It has to do with making sure that data or resources are available when the user needs it. In some cases, this is much more important than the other services. A public-facing governmental website can be of high importance to the public and must therefore be available at all times, but there is no need to keep the information on it confidential.
The mechanisms involved in maintaining confidentiality and integrity are quite dissimilar to the ones maintaining availability. In fact, they are at odd ends with each other. For instance, removing encryption and cryptographic signing could alleviate the load on a system’s computing resources, letting it handle more users.

In the school grade example, maintaining availability means making sure the grade information is available at all times. For instance, when students are applying for universities, it can be very important that the systems providing grade data do not become overloaded and fail.

Appendix B Information security terminology

There are a number of other terms used a lot within information security. They may appear self-explanatory, but this is what they mean in the context of security (Bishop, 2004).

12.4. Policy

If you consider a system to be a finite state automaton that has a number of states, a policy is a set of rules that defines what states are allowed and what states are disallowed. It will make a separation between different groups of states, to define within which realms the system state is allowed to be. A secure system is a system that starts off in an allowed state and never enters a disallowed state.

12.5. Mechanism

A mechanism is a tool or resource that makes sure the system stays within the allowed set of states. It is the actual implementation of the policies into the system. A policy must always be written with possible mechanisms in mind, to avoid creating rules that are not enforceable. For instance, if a system has the policy to not allow guests to edit the passwd-file, it has to have an operating system-level mechanism that can enforce the need to have write-rights to edit a file, and a mechanism for applying that rule to the passwd-file.

12.6. Assurance

Assurance is the practice of establishing trustworthiness in the system. What that means is that you make an assurance that the mechanisms you have implemented in the system conform to the policy laid out.

12.7. Threat

A threat is a possible unwanted event. Threats can be taken care of by either ensuring that the event cannot occur, or making sure that the event would cause no harm. Threats can be either intentional or accidents. By safeguarding against intentional attackers, we can be sure that unintentional attacks – accidents – are unlikely to happen.

12.8. Asset

An asset can mean many things, but in our case it will be used in the sense of an “information asset” unless otherwise stated; for instance login credentials. In a more general sense of the word, it can be expanded to include computer equipment, services provided etc.
12.9. Risk

Risk is a combination of other components: Risk = Threat * Vulnerability * Asset. A vulnerability is an exploitable security hole. From the asset we can estimate how much negative impact a risk would have if it occurs.

12.10. Authentication

Authentication is the concept of verifying the identity of a subject. It is necessary for many security functions, such as audit and authorization. It is commonly implemented as logging in with credentials such as username and password. We will return to this concept in more depth later.

12.11. Authorization

Authorization is the concept of connecting together an identity with a set of rights. If a user has been authenticated into the system, they should be given rights depending on the user. A user can be given rights depending on different properties. In some systems, like UNIX, this generally implies the identity of the user and the group the user belongs to (admin/guest etc.). Also, RBAC (Role-Based Access Control) is an alternative where you need to perform a specific task depending on your role in a system (developer/system operator etc.). These roles can be added and dropped depending on current task, and a user can have different roles (as opposed to groups).

There are two important basic types of authorization: access control lists and capabilities. In a computer file system, every file has an access control list. There is a defined set of operations, for instance read/write/truncate/delete/append, and a set of subjects. For instance, if a log file had the property (Lisa, append), that would mean that Lisa could append to the log file. No other properties, such as read are implied.

Where an access control list is a per-object list, the capabilities list is a per-subject list. For instance it can contain entries such as (reboot.sh, restart server) or (lp.exe, use connected printers).

There is commonly a user account with all available rights granted, in the UNIX world known as root or user id 0.

12.12. Audit

Auditing is the act of reviewing a system’s states after the fact. Audit is intimately tied with non-repudiation. It does not deal with prevention, but rather detection. To make sure auditing is possible, extensive log-keeping of important system events is usually used.

12.13. Identity

As in any human endeavor, identity is a very central concept. In computer science it is important to know the identity of a subject in order to assign rights to different resources. A principal is a unique entity (a human, a specific server, an embedded chip) (Bishop, 2004). An identity specifies a principal. It is common for computer systems to begin each interaction
with an authentication. The implementation details differ, but the reason is that the system wants to bind a principal with the system’s own concept of identity.

It is not possible to completely bind a principal to a specific system identity. In some cases this is not even desired, as one person can have different user accounts for different roles. This is in order to reduce privileges as much as possible for the task at hand.

In many systems, the user has not just one but a set of identities at the same time. For instance, in the UNIX world, you commonly have a real UID, an effective UID, a group UID etc. The real UID is the actual user id that the user logged in as, and the effective UID is prone to change. There is a set of programs with capabilities (see 12.11) to change the current effective UID. Typically this will be done for a specific task. For instance, the ping command requires a set of rights that the typical user does not have. So it has been given the capability to alter the user’s effective UID during execution (and will change it back upon completion) (University of Illinois, 1997).

This temporary change of effective UID is a potential security risk, since it often entails elevated privileges. For example, if a user can achieve a buffer overflow and take control of execution inside the ping command, they will have superuser privileges (IBM, 2000).

Appendix C Best practices

There are a few best practices when designing secure software. First introduced by Jerome Zaltzer and Michael Shroeder in 1975, these practices by themselves are not a complete method to create perfect software, but can serve as a guiding light (Saltzer, 1975). Since their inception, they have been revised in minor ways by different teams, but have largely stood the test of time (Davis, et al., 2004).

- **Securing the Weakest Link.** Find the weak spots in your system and secure them first.
- **Defense in Depth.** Design your system in layers of security. Make sure that a single point of failure does not expose more parts of your system then it has to.
- **Failing Securely.** If and when a system fails, it should revert to a safe default state. Ensure the default is to deny access. Always check return values for failures.
- **Least Privilege.** Privilege should only be escalated to the levels required, no higher.
- **Separation of Privilege.** A system should have different kinds of privilege depending on the task at hand. Switch between situational keys rather than a “master key”.
- **Economy of Mechanism.** Keep your system as simple and small as possible.
- **Least Common Mechanism.** Make sure subjects do not share the same mechanism for trust. E.g. having all users in the same group is not a sufficient differentiation.
- **Reluctance to Trust.** Assume that the system’s environment is insecure. Trust no external system/code/input/people unless they have been properly validated.
- **Never Assuming that your Secrets are Safe.** Never rely on “security by obscurity”, but design openly and assume that an attacker can disassemble or get your source code.
- **Complete Mediation.** Always check access permissions when a subject requests a resource. Do not cache permissions.
- **Psychological Acceptability.** Your security must be accepted by its users. A bothersome mechanism that does not appear to provide any security will be bypassed by its users.
• *Promoting Privacy.* Privacy is often a security concern, and needs to be promoted on many levels of an organization. Users respond very negatively to their personal data being leaked.

**Appendix D Brainstorming session**

The brainstorm session produced a set of keywords categorized into a few key categories. Many of these words have their natural place in RMF and will appear naturally in the process, and some words are things we have to take into account in different parts. Those that required further explanation have been elaborated upon. The keywords come in part from us and from Semcon. Some keywords are in Swedish. It should be noted that they are in no form binding, but merely suggestions or discussion points.

**Teknik**

- Zigbee
- WLAN
- GPRS
- X.509, Certificat
- Batteri
- Sensornät
- Adhoc
- Peer 2 Peer networks
- GPRS

**Business**

- Business case
- Policy
- Slippa resa
- Organisation
- IT-avdelning
  - Inte säkert att lokala policies tillåter extern kommunikation

**Connectivity**

- High data quantity
- Low data quantity
- Periodic transfer (of data)
- Real Time connected
- Sladdlöshet
- Network infrastructure

**Standarder**

- Internetteknik
  - (https/http/ssl/IPK etc.)
- Open source
- Banksäkerhet
- Terminologi

**Scenario**

- Smarta hem
- Svetsrobot
- Remote SWDL
- Infotainment
- Remote diagnostics
- On site diagnostics & SWDL
- Offline/online
- Backend/Backoffice
- Remote
  - Övervakning
  - Styrning
- Pizza
- Tjuvlarm

**Säk. Process**

- Script-kiddies
- Industrispionage
- Actors
- Gateways
- Domän
- Pentest
- Uptime
- Intrusion detection
  - alert

**Security-krav**

- Tillgänglighet
- Integritet
- Hemlighållande
- Tillit
- Safety class of collected/transferred data
- Safety class of connected device

**Systemkrav**

- Trådlöst
- Sladdlöshet
- Realtime connected
- Uptime
# Appendix E Operator commands

From the private chat between operator and backend:

<table>
<thead>
<tr>
<th>Name</th>
<th>Input</th>
<th>Output</th>
</tr>
</thead>
<tbody>
<tr>
<td>!Help</td>
<td></td>
<td>Displays a list of commands</td>
</tr>
<tr>
<td>!GetDump</td>
<td>&lt;node name&gt;</td>
<td>Prints heap dump from the node</td>
</tr>
<tr>
<td>!SetWelderConfigVar</td>
<td>&lt;leaf node name&gt; &lt;var name&gt; &lt;value&gt;</td>
<td>Sets a config var on the node</td>
</tr>
<tr>
<td>!GetWelderConfig</td>
<td>&lt;leaf node name&gt;</td>
<td>Prints the welder config from the leaf</td>
</tr>
<tr>
<td>!ListNodes</td>
<td></td>
<td>Prints a list of the nodes the backend is aware of</td>
</tr>
<tr>
<td>!DumpData</td>
<td>&lt;variable name&gt; [leaf node name]</td>
<td>Produces a graph of the node data points. For variable names, see section 6.3.1</td>
</tr>
<tr>
<td>!RequestDataPoints</td>
<td></td>
<td>Asks the gateway for a list of data points, rather than waiting for the periodic update</td>
</tr>
<tr>
<td>!Unregister</td>
<td></td>
<td>Unregisters self from the</td>
</tr>
<tr>
<td>Gateway Command</td>
<td>Description</td>
<td></td>
</tr>
<tr>
<td>-----------------</td>
<td>-------------</td>
<td></td>
</tr>
<tr>
<td>!Echo &lt;text&gt;</td>
<td>Echoes a text back to the operator</td>
<td></td>
</tr>
<tr>
<td>!Hello</td>
<td>“Y halo that mister &lt;operator name&gt;”</td>
<td></td>
</tr>
<tr>
<td>!IsRegistered</td>
<td>Asks the gateway whether the backend is registered or not</td>
<td></td>
</tr>
<tr>
<td>!Register</td>
<td>Registers the backend with the gateway</td>
<td></td>
</tr>
<tr>
<td>!GetStatusList  [leaf node name]</td>
<td>Lists all status changes received from a leaf</td>
<td></td>
</tr>
<tr>
<td>!GetAlarmsList  [leaf node name]</td>
<td>Lists all alarms from a leaf</td>
<td></td>
</tr>
</tbody>
</table>
Appendix F Message Types

Here is a list of the message types involved in the system. Note that messages going from a leaf to a backend and vice versa are still routed via the gateway. Also, in the case of multicast messages, the leaf is unaware of the backends listening. The gateway will then send the message on to every backend that is registered.

<table>
<thead>
<tr>
<th>Name</th>
<th>Contents</th>
<th>Direction</th>
</tr>
</thead>
<tbody>
<tr>
<td>Alarm</td>
<td>Alarm message and origin</td>
<td>leaf-&gt;multicast</td>
</tr>
<tr>
<td>WelderStatus</td>
<td>Status message and origin</td>
<td>leaf-&gt;multicast</td>
</tr>
<tr>
<td>IsRegistered</td>
<td>True/false</td>
<td>backend-&gt;gateway</td>
</tr>
<tr>
<td></td>
<td></td>
<td>gateway-&gt;backend</td>
</tr>
<tr>
<td>Register</td>
<td>Certificate</td>
<td>Any</td>
</tr>
<tr>
<td>Unregister</td>
<td></td>
<td>backend-&gt;gateway</td>
</tr>
<tr>
<td>RegistrationRequest</td>
<td></td>
<td>Any</td>
</tr>
<tr>
<td>GetWelderConfig</td>
<td></td>
<td>backend-&gt;leaf</td>
</tr>
<tr>
<td></td>
<td></td>
<td>leaf-&gt;backend</td>
</tr>
<tr>
<td>SetWelderConfigVar</td>
<td>Variable name, variable value</td>
<td>backend-&gt;leaf</td>
</tr>
<tr>
<td></td>
<td></td>
<td>leaf-&gt;backend</td>
</tr>
<tr>
<td>DataPoints</td>
<td>List of XMPDataPoints</td>
<td>leaf-&gt;gateway</td>
</tr>
<tr>
<td></td>
<td></td>
<td>gateway-&gt;backend</td>
</tr>
<tr>
<td>RequestDataPoints</td>
<td></td>
<td>backend-&gt;gateway</td>
</tr>
<tr>
<td>DumpRequest</td>
<td></td>
<td>backend-&gt;gateway</td>
</tr>
<tr>
<td></td>
<td></td>
<td>backend-&gt;leaf</td>
</tr>
<tr>
<td>DumpResponse</td>
<td>Contents of the heap dump</td>
<td>leaf-&gt;backend</td>
</tr>
<tr>
<td></td>
<td></td>
<td>gateway-&gt;backend</td>
</tr>
<tr>
<td>Raw</td>
<td></td>
<td>Operator-&gt;backend</td>
</tr>
</tbody>
</table>
Appendix G Commands

Levels of priority for the XMPCommandRunner command queue:

<table>
<thead>
<tr>
<th>Name</th>
<th>Order</th>
</tr>
</thead>
<tbody>
<tr>
<td>LOW</td>
<td>1</td>
</tr>
<tr>
<td>NORMAL</td>
<td>2</td>
</tr>
<tr>
<td>HIGH</td>
<td>3</td>
</tr>
<tr>
<td>INCOMING</td>
<td>4</td>
</tr>
<tr>
<td>SEMIURGENT</td>
<td>5</td>
</tr>
<tr>
<td>URGENT</td>
<td>6</td>
</tr>
</tbody>
</table>

The commands runnable by the XMPLary programs:

<table>
<thead>
<tr>
<th>Name</th>
<th>Priority</th>
<th>Purpose</th>
<th>Project</th>
</tr>
</thead>
<tbody>
<tr>
<td>DumpResponse</td>
<td>NORMAL</td>
<td>Respond to DumpRequest messages. Sends a DumpResponse message.</td>
<td>xmpcommon</td>
</tr>
<tr>
<td>IncomingDataPoints</td>
<td>LOW</td>
<td>Parses a DataPoints message, inserting the datapoints into the database.</td>
<td>xmpcommon</td>
</tr>
<tr>
<td>IsRegistered</td>
<td>NORMAL</td>
<td>Responds to a IsRegistered message.</td>
<td>xmpcommon</td>
</tr>
<tr>
<td>LoggerReceiver</td>
<td>LOW</td>
<td>Reads a message and outputs a human-readable version of it to the log system.</td>
<td>xmpcommon</td>
</tr>
<tr>
<td>MessageDispatchCommand</td>
<td>Depends on message type</td>
<td>Prepares a message for dispatching. Signs if appropriate, and encrypts if appropriate. Also checks if the target node is registered sending node. If it is not, queues a</td>
<td>xmpcommon</td>
</tr>
<tr>
<td>Command</td>
<td>Priority</td>
<td>Description</td>
<td>Module</td>
</tr>
<tr>
<td>----------------------------------------------</td>
<td>-----------</td>
<td>-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------</td>
<td>--------------</td>
</tr>
<tr>
<td>RequestRegistrationCommand</td>
<td>URGENT</td>
<td>In order for encryption to work, it needs to have the target’s public certificate.</td>
<td>xmpcommon</td>
</tr>
<tr>
<td>Register</td>
<td>URGENT</td>
<td>Parses a Register message, and inserts the given certificate file into the database.</td>
<td></td>
</tr>
<tr>
<td>RequestRegistrationCommand</td>
<td>SEMI-URGENT</td>
<td>Sends a message of type RegistrationRequest to target node.</td>
<td>xmpcommon</td>
</tr>
<tr>
<td>RespondToRegistrationRequest</td>
<td>URGENT</td>
<td>Responds to a message of type RegistrationRequest by sending a Register message.</td>
<td>xmpcommon</td>
</tr>
<tr>
<td>ReTryDelivery</td>
<td>NORMAL</td>
<td>Re-try delivery of previously undeliverable messages. (For instance due to lacking public certificate) (This is defunct).</td>
<td>xmpcommon</td>
</tr>
<tr>
<td>ReValidateCommand</td>
<td>NORMAL</td>
<td>Re-try validation of previously un-validatable messages. (For instance due to lacking public certificate) (This is defunct).</td>
<td>xmpcommon</td>
</tr>
<tr>
<td>UnpackAndReceiveMessage</td>
<td>INCOMING</td>
<td>The inversion of MessageDispatch. Reads a message, unpacks it, decrypts if necessary, verifies if necessary. Sends it on to the receive handlers.</td>
<td>xmpcommon</td>
</tr>
<tr>
<td>Unregister</td>
<td>NORMAL</td>
<td>Unregisters the sending node from the target node’s list.</td>
<td>xmpcommon</td>
</tr>
<tr>
<td>GatewayRegister</td>
<td>NORMAL</td>
<td>Register a backend with the gateway. Upon registration, searches the database for datapoints that this backend probably lacks and puts it on a send queue. Also schedules a SendAlarmBacklog command.</td>
<td>xmpgate</td>
</tr>
<tr>
<td>Function</td>
<td>Priority</td>
<td>Description</td>
<td>Module</td>
</tr>
<tr>
<td>---------------------------------------</td>
<td>----------</td>
<td>-----------------------------------------------------------------------------</td>
<td>--------</td>
</tr>
<tr>
<td>MulticastToBackends</td>
<td>NORMAL</td>
<td>Receives messages that are to be multicast to backends (such as alarms and status changes)</td>
<td>xmpgate</td>
</tr>
<tr>
<td>SendAlarmBacklog</td>
<td>LOW</td>
<td>Schedules the sending of the whole alarm backlog to a new backend</td>
<td>xmpgate</td>
</tr>
<tr>
<td>SendDataPoints</td>
<td>NORMAL</td>
<td>Schedules the sending of datapoints to a backend. The datapoints are split up into messages of 240 data points per message.</td>
<td>xmpgate</td>
</tr>
<tr>
<td>SwitchboardCommand</td>
<td>NORMAL</td>
<td>Acts as a switchboard between leaves and gateways. Forwards messages.</td>
<td>xmpgate</td>
</tr>
<tr>
<td>RunPeriodicUpdate</td>
<td>LOW</td>
<td>Scheduled by the Periodic Updates Thread, checks if there are datapoints to pack up and send.</td>
<td>xmpgate</td>
</tr>
<tr>
<td>AlarmReceiver</td>
<td>HIGH</td>
<td>Receives an Alarm message and sends a warning about it to the operator</td>
<td>xmpback</td>
</tr>
<tr>
<td>OperatorInputCommand</td>
<td>NORMAL</td>
<td>An abstract class that wraps the classes handling operator inputs. They are described under Appendix E.</td>
<td>xmpback</td>
</tr>
<tr>
<td>WeldStatusReceiver</td>
<td>NORMAL</td>
<td>Similar to AlarmReceiver.</td>
<td>xmpback</td>
</tr>
<tr>
<td>SendAlarmCommand</td>
<td>HIGH</td>
<td>Sends an alarm message</td>
<td>xmpleaf</td>
</tr>
<tr>
<td>SendStatus</td>
<td>HIGH</td>
<td>Sends a status update</td>
<td>xmpleaf</td>
</tr>
<tr>
<td>SendWelderConfig</td>
<td>NORMAL</td>
<td>Responds to a GetWelderConfig by sending its config file</td>
<td>xmpleaf</td>
</tr>
<tr>
<td>SendWeldingDatapoints</td>
<td>NORMAL</td>
<td>Packs and sends datapoints to</td>
<td>xmpleaf</td>
</tr>
<tr>
<td>Method</td>
<td>Status</td>
<td>Description</td>
<td></td>
</tr>
<tr>
<td>---------------------</td>
<td>--------</td>
<td>-----------------------------------------------------------------------------</td>
<td></td>
</tr>
<tr>
<td>SetWeldConfigVar</td>
<td>NORMAL</td>
<td>Responds to a SetWeldConfigVar message by setting a variable in its config file</td>
<td></td>
</tr>
</tbody>
</table>
| UpdateWelderValues  | NORMAL | Ticks the simulation of a welder. Updates variables and changes states when necessary.
Appendix H 3rd party software used

<table>
<thead>
<tr>
<th>Name</th>
<th>Type</th>
<th>Purpose</th>
<th>License</th>
</tr>
</thead>
<tbody>
<tr>
<td>Smack</td>
<td>Java API</td>
<td>XMPP client API</td>
<td>Apache v2</td>
</tr>
<tr>
<td>Openfire</td>
<td>XMPP server</td>
<td>XMPP server</td>
<td>Apache v2</td>
</tr>
<tr>
<td>BouncyCastle</td>
<td>Java API</td>
<td>Cryptography API</td>
<td>MIT X11</td>
</tr>
<tr>
<td>vtCrypt</td>
<td>Java API</td>
<td>BouncyCastle wrapper</td>
<td>LGPL v3 and Apache v2</td>
</tr>
<tr>
<td>Joda-time</td>
<td>Java API</td>
<td>Date/Time API</td>
<td>Apache v2</td>
</tr>
<tr>
<td>Log4j</td>
<td>Java API</td>
<td>Logging API</td>
<td>Apache v2</td>
</tr>
<tr>
<td>Commons-logging</td>
<td>Java API</td>
<td>Logging API (because of a dependency)</td>
<td>Apache v2</td>
</tr>
<tr>
<td>Ormlite</td>
<td>Java API</td>
<td>ORM (database manager)</td>
<td>Open Source License</td>
</tr>
<tr>
<td>Sqlite-jdbc</td>
<td>Java API</td>
<td>Database</td>
<td>Apache 2</td>
</tr>
<tr>
<td>XCA</td>
<td>CA manager</td>
<td>Set ut certificate chain</td>
<td>BSD Revised</td>
</tr>
<tr>
<td>Tail for Win32</td>
<td>Log viewer</td>
<td>To read log files produced by xmplary</td>
<td>GPL</td>
</tr>
<tr>
<td>Ant</td>
<td>Build tool</td>
<td>Build executables</td>
<td>Apache v2</td>
</tr>
<tr>
<td>Prosody</td>
<td>XMPP server</td>
<td>XMPP server, lightweight</td>
<td>MIT X11</td>
</tr>
<tr>
<td>HSQLDB</td>
<td>Java API</td>
<td>Database (to replace sqlite)</td>
<td>BSD</td>
</tr>
</tbody>
</table>
Appendix I Example printout

12.14. Getting system status

(10:40:40) lulzmachine: !GetDump leaf-20
(10:40:41) backend-3@se-got-nb-0084/Smack: Ok let me send request...
(10:40:45) backend-3@se-got-nb-0084/Smack: Received message of type DumpResponse. Verified: true. (o->f->t). (leaf-20->gateway->backend-3) Contents: "WeldingThread" Id=20 TIMED_WAITING
at java.lang.Thread.sleep(Native Method)
at se.localhost.xmplary.xmplaf.WeldingThread.sleep(WeldingThread.java:68)

"XMPCommandRunner" Id=19 RUNNABLE
at sun.management.ThreadImpl.dumpThreads0(Native Method)
at sun.management.ThreadImpl.dumpAllThreads(Unknown Source)
at se.loclcalhost.xmplary.common.commands.DumpResponse.execute(DumpResponse.java:28)
at se.localhost.xmplary.common.XMPCommandRunner.consume(XMPCommandRunner.java:62)
at se.localhost.xmplary.common.XMPCommandRunner.run(XMPCommandRunner.java:49)

"Smack Listener Processor (0)" Id=18 WAITING on java.util.concurrent.locks.AbstractQueuedSynchronizer$ConditionObject@55a6c368
at sun.misc.Unsafe.park(Native Method)
- waiting on java.util.concurrent.locks.AbstractQueuedSynchronizer$ConditionObject@55a6c368
at java.util.concurrent.locks.LockSupport.park(Unknown Source)
at java.util.concurrent.locks.AbstractQueuedSynchronizer$ConditionObject.await(Unknown Source)
at java.util.concurrent.LinkedBlockingQueue.take(Unknown Source)
at java.util.concurrent.ThreadPoolExecutor.getTask(Unknown Source)
at java.util.concurrent.ThreadPoolExecutor$Worker.run(Unknown Source)

"Smack Keep Alive (0)" Id=16 TIMED_WAITING
at java.lang.Thread.sleep(Native Method)
at java.lang.Thread.run(Unknown Source)

"Smack Packet Reader (0)" Id=15 RUNNABLE (in native)
at java.net.SocketInputStream.socketRead0(Native Method)
at java.net.SocketInputStream.read(Unknown Source)
at com.sun.net.ssl.internal.ssl.InputRecord.readFully(Unknown Source)
at com.sun.net.ssl.internal.ssl.SSLSocketImpl.readRecord(Unknown Source)
- locked java.lang.Object@37670cc6
at com.sun.net.ssl.internal.ssl.SSLSocketImpl.readDataRecord(Unknown Source)
at com.sun.net.ssl.internal.ssl.AppInputStream.read(Unknown Source)
- locked com.sun.net.ssl.internal.ssl.AppInputStream@15b57dcb
at sun.nio.cs.StreamDecoder.readBytes(Unknown Source)
...

"Smack Packet Writer (0)" Id=14 WAITING on java.util.concurrent.ArrayBlockingQueue@68e86f41
at java.lang.Object.wait(Native Method)
- waiting on java.util.concurrent.ArrayBlockingQueue@68e86f41
at java.lang.Object.wait(Object.java:485)
12.15. Setting and getting welder configuration

Getting:

(10:46:24) lulzmachine: !getwelderconfig leaf-20
(10:46:24) backend-3@se-got-nb-0084/Smack: Ok let me send request...
(10:47:05) backend-3@se-got-nb-0084/Smack: Received message of type GetWelderConfig.
Verified: true. (o->f->t). (leaf-20->gateway->backend-3) Contents: <?xml version="1.0" encoding="UTF-8" standalone="no"?>
<!DOCTYPE properties SYSTEM "http://java.sun.com/dtd/properties.dtd">
<properties>
<comment/>
<entry key="fiskar">23</entry>
<entry key="fiskers">23</entry>
</properties>

Setting:
12.16. Example printout from periodic data point dispatch:

<in the debug channel>

(10:47:12) backend-3@se-got-nb-0084/Smack: Received message of type SetWelderConfigVar. Veriﬁed: true. (o->f->t). (leaf-20->gateway->backend-3) Contents: <?xml version="1.0" encoding="UTF-8" standalone="no"?>
<!DOCTYPE properties SYSTEM "http://java.sun.com/dtd/properties.dtd">
<properties>
<comment/>
<entry key="CPM">40</entry>
<entry key="fiskar">23</entry>
<entry key="fiskers">23</entry>
</properties>

12.17. Printout for alarms

<to operator>


12.18. Receiving alarms

Sequence of events on backend after receiving alarm (from xmpback/files/log.log):

09:57:20.706 [XMPCommandRunner: INFO] (XMPCommandRunner.java:61) => Running command: [Q/P][0/3]:
se.loclhost.xmplary.common.commands.UnpackAndReceiveMessage
09:57:20.707 [XMPCommandRunner: INFO] (UnpackAndReceiveMessage.java:34) => Attempting to parse message
["iv":"hPNOGaXdwl1HN2uKPFOQA==","response_to_id":0,"contents":""]
The command AlarmReceiver will send this message to operator:

(09:57:21) backend-3@se-got-nb-0084/Smack: Received message of type Alarm. Verified: true. (o->f->t). (leaf-20->gateway->backend-3) Contents: {"message":"Someone dropped a cheeseburger on the circuit board."}
Appendix J Creating a keystore

(XMPLary_Jar.p12 is a PKCS12 file with private key and public cert chain)

$ keytool -importkeystore -deststorepass starwars -destkeypass starwars -destkeystore keystore.jks -srckeystore XMPlary_Jar.p12 -srcstoretype PKCS12 -srcstorepass starwars -alias "XMPlary Jar"

In the ant-file:
<signjar jar="${buildfolder}jar/xmplary.jar" alias="XMPLary Jar" storepass="starwars" keystore="../xmpcommon/files/keystore.jks" />

To verify:
$ jarsigner.exe -verify jar/xmplary.jar
jar verified.
Appendix K Prosody config

--------- Server-wide settings ---------
-- Settings in this section apply to the whole server and are the default settings
-- for any virtual hosts

admins = { }
modules_enabled = {
    -- Generally required
    "roster": -- Allow users to have a roster. Recommended :)
    "saslauth": -- Authentication for clients and servers. Recommended if you want to log in.
    "tls": -- Add support for secure TLS on c2s/s2s connections
    "dialback": -- s2s dialback support
    "disco": -- Service discovery

    -- Not essential, but recommended
    "private": -- Private XML storage (for room bookmarks, etc.)
    "vcard": -- Allow users to set vCards
    "privacy": -- Support privacy lists
    "compression": -- Stream compression (Debian: requires lua-zlib module to work)

    -- Nice to have
    "legacyauth": -- Legacy authentication. Only used by some old clients and bots.
    "version": -- Replies to server version requests
    "uptime": -- Report how long server has been running
    "time": -- Let others know the time here on this server
    "ping": -- Replies to XMPP pings with pongs
    "pep": -- Enables users to publish their mood, activity, playing music and more
    "register": -- Allow users to register on this server using a client and change passwords
    "adhoc": -- Support for "ad-hoc commands" that can be executed with an XMPP client

    -- Admin interfaces
    "admin_adhoc": -- Allows administration via an XMPP client that supports ad-hoc commands
    "admin_telnet": -- Opens telnet console interface on localhost port 5582

    -- Other specific functionality
    "bosh": -- Enable BOSH clients, aka "Jabber over HTTP"
    "httpserver": -- Serve static files from a directory over HTTP
    "groups": -- Shared roster support
    "announce": -- Send announcement to all online users
    "welcome": -- Welcome users who register accounts
    "watchregistrations": -- Alert admins of registrations
    "motd": -- Send a message to users when they log in

    -- Debian: do not remove this module, or you lose syslog
    -- support
    "posix": -- POSIX functionality, sends server to background, enables syslog, etc.
}
modules_disabled = {
    -- "presence": -- Route user/contact status information
    -- "message": -- Route messages
    -- "iq": -- Route info queries
    -- "offline": -- Store offline messages
}
allow_registration = true;
daemonize = true;
pidfile = "/var/run/prosody/prosody.pid";

-- These are the SSL/TLS-related settings. If you don't want
-- to use SSL/TLS, you may comment or remove this
--ssl = {
--  key = "/etc/prosody/certs/localhost.key";
--  certificate = "/etc/prosody/certs/localhost.cert";
--}

authentication = "internal_plain"
-- Debian:
-- Logs info and higher to /var/log
-- Logs errors to syslog also
log = {
-- Log files (change 'info' to 'debug' for debug logs):
  info = "/var/log/prosody/prosody.log";
  error = "/var/log/prosody/prosody.err";
-- Syslog:
  { levels = { "error" }; to = "syslog"; }
}

disallow_s2s = true
VirtualHost "arm"
VirtualHost "192.168.204.249"

Component "conference.arm" "muc"
Include "conf.d/*.cfg.lua"
Appendix L XMPLary build config

The project specific one (used for xmpgate):

```xml
<?xml version="1.0" encoding="UTF-8" standalone="no"?>
<project name="xmpbuild" default="create_run_jar">
  <!--this file is pretty awesome -->
  <!--ANT 1.7 is required -->
  <target name="setup" description="Set up the individual project settings">
    <property name="certname" value="XMPlary_Gateway_Generous" />
    <property name="buildfolder" value="build" />
    <property name="entry" value="se.lolcalhost.xmplary.xmpgate.GatewayMain" />
    <property name="name" value="gateway" />
    <property name="type" value="gateway" />
    <property name="FullName" value="XMPlary node gateway" />
    <property name="database" value="database.db" />
  </target>

  <import file="../xmpcommon/files/common_ant.xml" as="nested"/>

  <target name="create_run_jar" depends="nested.common_stuff"/>
</project>
```

Here is the common one. It will generate the jar file, copy the relevant certificates and private keys, generate a configuration file and copy a few auxiliary files. It will also sign the jar file, to make any subsequent tampering detectable.

```xml
<project>
  <target name="setup" />
  <target name="genconf" description="Generate the conf.properties file">
    <property name="Domain" value="arm" />
    <property name="Address" value="192.168.204.249" />
    <property name="RoomDomain" value="conference.arm" />
    <property name="room" value="yolo" />
    <property name="certfile" value="certs/${certname}.crt" />
    <property name="keyfile" value="keys/${certname}.pem" />
    <property name="pass" value="pass" />

    <mkdir dir="${buildfolder}files/"/>
    <touch file="${buildfolder}files/conf.properties"/>
    <echoproperties destfile="${buildfolder}files/conf.properties">
      <propertyset>
        <propertyref name="room"/>
        <propertyref name="name"/>
        <propertyref name="certfile"/>
        <propertyref name="Domain"/>
        <propertyref name="Address"/>
        <propertyref name="RoomDomain"/>
        <propertyref name="type"/>
        <propertyref name="keyfile"/>
        <propertyref name="FullName"/>
        <propertyref name="database"/>
        <propertyref name="pass"/>
      </propertyset>
    </echoproperties>
</target>
```
<target name="common_stuff" depends="setup.genconf" description="Invoke all the common stuff">
  <copy file="../xmpcommon/files/run.sh" todir="${buildfolder}" />
  <copy file="../xmpcommon/files/run.bat" todir="${buildfolder}" />
  <copy file="../xmpcommon/files/log4j.properties" todir="${buildfolder}files/" />
  <copy file="../certs/certs/${certname}.crt" todir="${buildfolder}certs/certs" />
  <copy file="../certs/keys/${certname}.pem" todir="${buildfolder}certs/keys" />
  <copy todir="${buildfolder}certs/certs/ca">
    <fileset dir="../certs/certs/ca" />
  </copy>

  <jar destfile="${buildfolder}jar/xmplary.jar" filesetmanifest="mergewithoutmain" duplicate="preserve">
    <manifest>
      <attribute name="Main-Class" value="${entry}" />
      <attribute name="Class-Path" value="/" />
    </manifest>
    <fileset dir="bin"/>
    <zipfileset excludes="META-INF/*.SF" src="../xmpcommon/lib/smack.jar"/>
    <zipfileset excludes="META-INF/*.SF" src="../xmpcommon/lib/smackx.jar"/>
    <zipfileset excludes="META-INF/*.SF" src="../xmpcommon/lib/ormlite-core-4.42.jar"/>
    <zipfileset excludes="META-INF/*.SF" src="../xmpcommon/lib/log4j-1.2.17.jar"/>
    <zipfileset excludes="META-INF/*.SF" src="../xmpcommon/lib/JSON-java.jar"/>
    <zipfileset excludes="META-INF/*.SF" src="../xmpcommon/lib/joda-time-2.1.jar"/>
    <zipfileset excludes="META-INF/*.SF" src="../xmpcommon/lib/ormlite-jdbc-4.42.jar"/>
    <zipfileset excludes="META-INF/*.SF" src="../xmpcommon/lib/hsqldb.jar"/>
    <zipfileset excludes="META-INF/*.SF" src="../xmpcommon/lib/bcmail-jdk15on-147.jar"/>
    <zipfileset excludes="META-INF/*.SF" src="../xmpcommon/lib/bcpkix-jdk15on-147.jar"/>
    <zipfileset excludes="META-INF/*.SF" src="../xmpcommon/lib/bcprov-debug-jdk15on-147.jar"/>
    <zipfileset excludes="META-INF/*.SF" src="../xmpcommon/lib/bcprov-ext-debug-jdk15on-147.jar"/>
    <zipfileset excludes="META-INF/*.SF" src="../xmpcommon/lib/jcommon-1.0.17.jar"/>
    <zipfileset excludes="META-INF/*.SF" src="../xmpcommon/lib/jfreechart-1.0.14.jar"/>
    <zipfileset excludes="META-INF/*.SF" src="../xmpcommon/lib/vt-crypt-2.1.4.jar"/>
    <zipfileset excludes="META-INF/*.SF" src="../xmpcommon/lib/commons-loggin-1.1.1.jar"/>
  </jar>
  <signjar jar="${buildfolder}jar/xmplary.jar" filesetmanifest="mergewithoutmain" storepass="starwars" keystore="/xmpcommon/files/keystore.jks" />
</target>
</project>