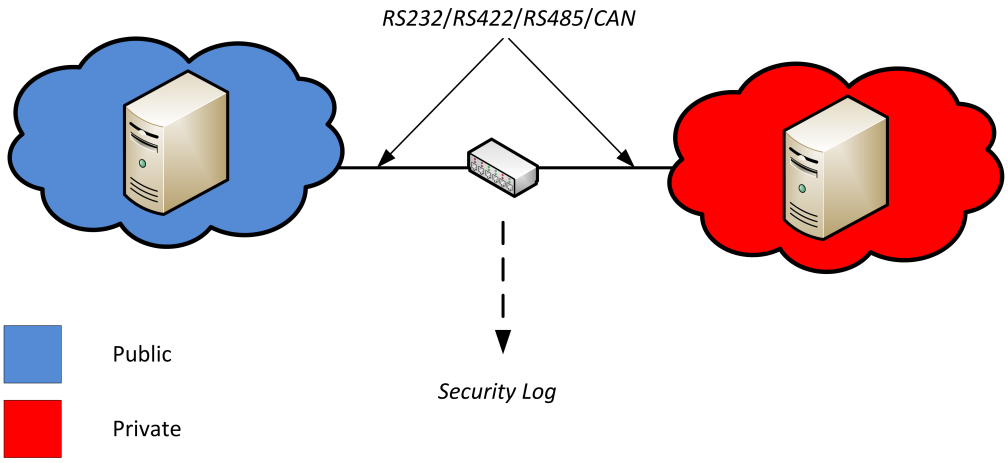
CHALMERS



Filtering Security Mechanism for Digital Communication

Master's Thesis in Computer Systems and Networks

ANDERS NORDIN,
HANNES SANDAHL

Department of Computer Science & Engineering
Chalmers University of Technology
Gothenburg, Sweden 2015

Filtering Security Mechanism for Digital Communication

ANDERS NORDIN
HANNES SANDAHL

Examiner: ANDREI SABELFELD
Supervisor: MUSARD BALLIU

Chalmers University of Technology
Department of Computer Science and Engineering
SE-412 96 Gothenburg
Sweden
Telephone: +46 (0)31-772 1000

Department of Computer Science and Engineering
Gothenburg, Sweden. June 2015.

Abstract

Many digital systems handle data that is either public or private depending on its sensitivity level. In these systems it is important that sensitive data is not lost or exposed to the public domain, even if parts of the system has been compromised with malicious code. In the military domain it is common for systems to be divided into different information zones based on the confidentiality of the data that the subsystem handles. In some cases these subsystems need to exchange data with each other. Therefore, it is important that sensitive data is not lost or exposed to lower classification levels, even if one unit has been compromised by malicious code. As of today and to our knowledge, no device that can control this information flow for serial communication in common protocols like RS232, RS422, RS485, and CAN exists. This thesis proposes a device that could be placed between two information zones in order to ensure that classified data is not exposed to lower classification levels. The software is developed in SPARK and acts as an *Intrusion Prevention System*. It is based on the latest research within the area of anomaly detection. This thesis includes how such a device should be designed, developed, certified, and integrated in order to meet requirements from the military. Finally, the device is tested and evaluated with respect to both performance and security.

Acknowledgements

We would like to start by thanking Saab Electronic Defence System in Gothenburg for giving us the opportunity to conduct this thesis. Special thanks to Fredrik Magnusson, our supervisor, for feedback and guidance throughout the project. We would also like to thank Anders Råberg for helping us with the Ada code and Torgny Hansson, our manager, for having us in his team. Finally, we would like to express our gratitude towards our supervisor at Chalmers University of Technology, Musard Balliu, for his advice and valuable feedback on the report.

Anders Nordin and Hannes Sandahl, Gothenburg 15/05/29

List of Terminology

Below follows a list of useful words that will be used in the report. The list is retrieved from RFC 2828, Internet Security Glossary, May 2000 [1].

Adversary (threat agent) An entity that attacks, or is a threat to, a system.

Attack An assault on system security that derives from an intelligent threat, i.e., an intelligent act that is a deliberate attempt (especially in the sense of a method or technique) to evade security services and violate the security policy of a system.

- **Active vs. passive** An "active attack" attempts to alter system resources or affect their operation. A "passive attack" attempts to learn or make use of information from the system but does not affect system resources. (E.g., see: wiretapping.)
- **Insider vs. outsider** An "inside attack" is an attack initiated by an entity inside the security perimeter (an "insider"), i.e., an entity that is authorized to access system resources but uses them in a way not approved by those who granted the authorization. An "outside attack" is initiated from outside the perimeter, by an unauthorized or illegitimate user of the system (an "outsider"). In the Internet, potential outside attackers range from amateur pranksters to organized criminals, international terrorists, and hostile governments.

Authentication The process of verifying an identity claimed by or for a system entity.

Authorization

1. An "authorization" is a right or a permission that is granted to a system entity to access a system resource.
2. An "authorization process" is a procedure for granting such rights.
3. To "authorize" means to grant such a right or permission.

Countermeasure An action, device, procedure, or technique that reduces a threat, a vulnerability, or an attack by eliminating or preventing it, by minimizing the harm it can cause, or by discovering and reporting it so that corrective action can be taken.

Covert Channel A intra-system channel that permits two cooperating entities, without exceeding their access authorizations, to transfer information in a way that violates the system's security policy.

Risk An expectation of loss expressed as the probability that a particular threat will exploit a particular vulnerability with a particular harmful result.

Security Policy A set of rules and practices that specify or regulate how a system or organization provides security services to protect sensitive and critical system resources.

Social Engineering A euphemism for non-technical or low-technology means – such as lies, impersonation, tricks, bribes, blackmail, and threats – used to attack information systems.

System Resource (Asset) Data contained in an information system; or a service provided by a system; or a system capability, such as processing power or communication bandwidth; or an item of system equipment (i.e., a system component—hardware, firmware, software, or documentation); or a facility that houses system operations and equipment.

Threat A potential for violation of security, which exists when there is a circumstance, capability, action, or event, that could breach security and cause harm. That is, a threat is a possible danger that might exploit a vulnerability.

Vulnerability A flaw or weakness in a system's design, implementation, or operation and management that could be exploited to violate the system's security policy.

Wiretapping An attack that intercepts and accesses data and other information contained in a flow in a communication system.

Zero-day Attack An attack targeting an unknown vulnerability.

Contents

1 Introduction **1**
- 1.1 Problem Description . . . 3
- 1.2 Aim . . . 4
- 1.3 Contributions . . . 5
- 1.4 Limitations . . . 5
- 1.5 Outline . . . 6

2 Computer Security Concepts **7**
- 2.1 Computer Security . . . 7
- 2.2 Security Models . . . 9
  - 2.2.1 Bell-LaPadula . . . 9
  - 2.2.2 Biba . . . 10
  - 2.2.3 Chinese Wall . . . 11
- 2.3 Intrusion Detection and Prevention Systems . . . 12
  - 2.3.1 Signature-based Detection . . . 12
  - 2.3.2 Anomaly-based Detection . . . 13
- 2.4 Anagram . . . 14
- 2.5 SPARK . . . 17
- 2.6 Serial Communication Architecture . . . 19

3 Threat Model **21**
- 3.1 Attacks and Vulnerabilities . . . 21
  - 3.1.1 System Design and Development . . . 22
  - 3.1.2 System Testing . . . 23
  - 3.1.3 Operation . . . 24
  - 3.1.4 Maintenance and Storage Media . . . 25
  - 3.1.5 Disposal . . . 26

4 Evaluation of the Anomaly-based Detection System **27**
4.1 NMEA 0183 27
4.1.1 Detection Rate 29
4.1.2 False Positive Rate and Training Data 29
4.2 The Simple Protocol 30
4.2.1 Detection Rate 31
4.2.2 False Positive Rate 32
4.3 Execution Time 32

5 Securing the Systems **34**

6 Conclusion and Future Work **41**
6.1 Conclusion 41
6.2 Future Work 42

Bibliography **45**

1

Introduction

As shown in a report [2] by PwC, *Computer Emergency Response Team* (CERT), and *United States Secret Service*, cybersecurity incidents multiply in frequency and with it comes a great increase in cost. The cybersecurity programs of US organizations are not nearly good enough to rival the technological ability, tactical skills, and persistence of the adversaries. This is according to us probably true for organizations all over the world. Insider and employee vulnerabilities are often overlooked or underestimated and the IT security of the often used third-party suppliers is not assessed. The report also shows how most of the US companies do not strategically invest in cybersecurity and ensure that it is aligned with their overall business strategy.

Another report [3] shows how employees are often involved in incidents related to cybersecurity. These insiders are often overlooked when looking at possible vulnerabilities. It is stated that employees in many cases may unwittingly compromise data through loss of mobile devices or fall victim of social engineering. It is pointed out that almost a third (32 %) of the respondents reported that insider crimes are more costly or damaging than incidents carried out by outsiders. Although many companies do not have an insider-threat program in place and are, therefore, not prepared to prevent, detect, and respond to internal threats. Also, companies tend to handle insider crimes internally. Law enforcement should be involved to make sure that future employers could make an actual threat assessment of that person and take appropriate actions to protect their assets.

The *Department of Defense* (DoD) writes that the US cannot be confident that their critical IT systems will work under an attack [4]. This is due to the fact that they are facing a *full spectrum adversary*, i.e., a well-founded adversary that uses military and intelligence capabilities to conduct a sophisticated cyberattack. We are convinced that this is true for other countries, organizations, and companies as well. The report mentions that the US has used a common approach to secure their critical systems, i.e., taken great care to secure the use and operation of their hardware, but not the

underlying software. This approach has not kept up with the capabilities and tactics of the threat agents. The DoD concludes that:

> The cyber threat is serious, with potential consequences similar in some ways to the nuclear threat of the Cold War.

The PwC report from year 2015 [3], shows how compromises performed by nation-states, organized crime, and competitors are among the fastest growing threats as of year 2014. An astonishing 86 % increase in respondents who say they have been compromised by nation states is reported. The volume of compromises is probably under-reported, given the ability of nation-state adversaries to perform attacks undetected. The aerospace and defense industry is the most targeted sector for certain attacks. This since it is an industry whose secrets can contain information concerning the national security.

Today we face threats from organized crime groups, activists, and hackers. In addition to this there is an emerging threat of cyberterrorism and cyberwarfare between nations, targeting military systems and infrastructure. Researchers claim that this is a real and imminent risk, and often mention *Stuxnet* as an example [5]. Stuxnet is a very sophisticated worm discovered in year 2010. It was targeting a very specific *programmable logic controller* (PLC) at an Iranian power plant and used attacks that targeted four unknown vulnerabilities, which is very rare. It was considered groundbreaking since it chose its target very precise, its level of sophistication, and implications for future malware. As of today, no one has taken responsibility for Stuxnet, but nation involvement is strongly suspected. If that is the case, it is hard proof of governments actively pursuing offensive capabilities, not only network defense which used to be the case. There is also a concern that the technology behind Stuxnet will be copied and used for other targets as PLCs are used widely in many automated industrial systems.

This introduction conclude that digital systems, in particular defense systems, are very exposed to threats and attacks from a vast number of threat agents. This is especially worrying since most digital systems handle sensitive data in some way. Some examples of sensitive data, could be the current position of a military convoy or personal information about the staff. Therefore, most data is usually separated into either a public or private domain, depending on its sensitivity. Military systems often uses a security model to enforce military security policies. The model classifies information into different security levels, e.g., `TOP SECRET`, `SECRET`, `CONFIDENTIAL`, `RESTRICTED`, and `UNCLASSIFIED` and should guarantee that information only flows between classes according to the security policies of the organization. However, there could be several reasons why systems, military systems in particular, would want to divide information into different classification zones.

- Parts of the system need to handle classified information while others not.
- It might be expensive to implement a high security level in the entire system due to the cost associated with stricter requirements.
- It might be impossible to implement a high security level in the entire system.

- Processes, like development, maintenance, and disposal, can be unnecessary complicated with maximum security through the entire system.
- Usage might become unnecessary complicated for the operators with a high level of security through the entire system.

The separation into different zones are sometimes only possible under the constraint that the zones still can exchange information to some extent. Exchanging data from a lower security level to a higher is usually not a problem. However, in some cases information from a higher classified system needs to send information to a lower classified, for example assume that there is a military defense system classified as `SECRET` which can calculate the impact of a missile, where it is going to hit, and when. The information about the missile can be very sensitive and must be protected within the system, but still, the people that are located in the area of impact must be warned. This could be done by sending a message to an alarm system, which could be labeled as `UNCLASSIFIED`. Therefore, one would need to ensure that sensitive information is not leaked in an unauthorized manner, e.g., the computer of the `SECRET` system could be compromised. An adversary should not be able to use this to getting hold of sensitive or private information by leaking it to a system labeled as `UNCLASSIFIED`, i.e., by violating the confidentiality rules. One must also guarantee integrity, e.g., that the information received is not modified by an unauthorized party. Some services are critical, like the alarm, and need to be available for authorized users when needed and cannot be blocked in any way.

1.1 Problem Description

To our knowledge, there exists no filtering security mechanism that checks data traffic between systems, using serial digital communication, such that classified data is not leaked to zones with lower classification level in digital communication. That is a mechanism that in a predetermined manner examines the content of a message and terminates malicious packets while letting the legit ones through. A mechanism of this kind needs to inspect the entire packet to prevent that any part of it is used for malicious intent. The mechanism should be placed between the two systems, as illustrated in Figure 1.1.

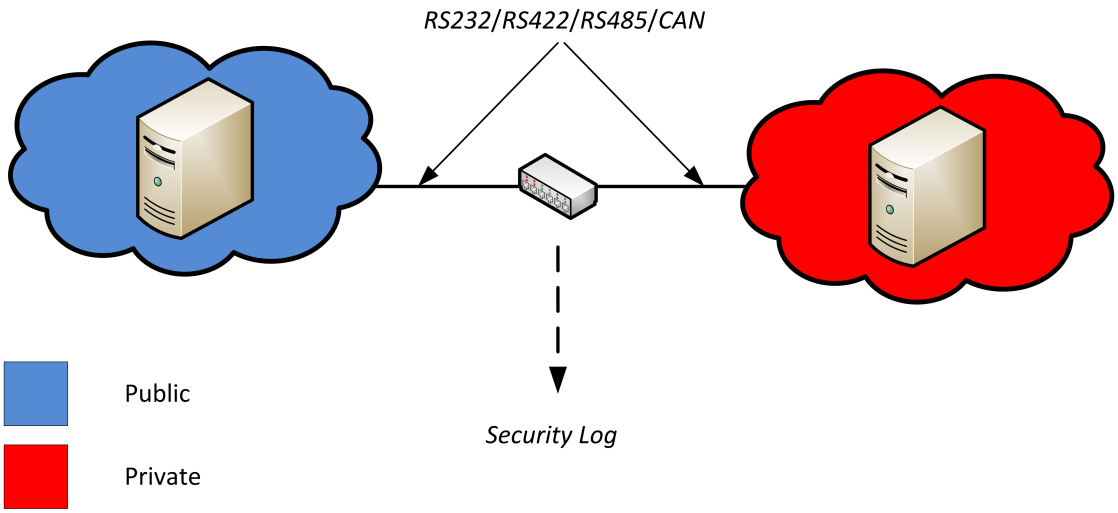


Figure 1.1: Placement of the device.

A device called GARM, developed for the Swedish Armed Forces, is used in a similar manner. However, GARM is only designed for Ethernet and not for other commonly used serial communication protocols, i.e., it is not generic and cannot be used in conjunction with other protocols. Also, only a first version of GARM exists and as the standards has been renewed there is currently no device that fulfills the requirements. Several flaws also plague the system; the hardware used for it is outdated, it does not support authentication, and the performance is quite lacking.

1.2 Aim

The purpose of this thesis is to conduct a study on how a filtering security device for digital communication should be designed, developed, certified, and integrated to meet the requirements. The requirements concern both performance and security. It is also important that the device can be trusted by customers and therefore the device requires auditability. The questions to answer are:

- Which kind of attacks are possible and plausible?
- What is the current threat assessment on the system?
- What kind of security is needed with respect to the attack vectors?
- How is it possible to prove that the device is trustworthy?
- Would it be acceptable that some parts of the device is not auditable?
- How much delay, caused by a filtering security mechanism, can be accepted?
- How is it possible to detect if there has been an attempt to tamper with the device?

- How to log information in a secure manner?
- How to ensure that only authorized personnel can configure the device?
- How to prevent traffic from circumventing the filter?

We will in this report answer all of these questions. The latter part of the thesis consists of development of the actual software for the filtering mechanism. The design and implementation will be based on the answers to the questions in the list above. The hardware assessment and construction is left for future work.

1.3 Contributions

We propose a concept for the software of a filtering security mechanism that can be used to help provide confidentiality and integrity in a system. The concept builds on the latest concepts of intrusion prevention that detect attacks and taking action against them. We demonstrate how the programming language SPARK can be used to enhance the software security of the mechanism, but also how the mechanism can be designed to provide a high detection rate, a low false positive rate, and a more than sufficient performance.

We cannot guarantee that all the security properties in a system is fulfilled by just adding the filtering mechanism. Why the current security solution is not enough from an end-to-end perspective is explained and a new framework for an end-to-end solution is presented. This solution should be adopted as soon as possible to prevent the otherwise inevitable compromises of current and future systems.

1.4 Limitations

A filtering security device is relevant for all IT systems that handle sensitive information. This master thesis focuses on a device that follows the standards and requirements for Swedish military systems.

Digital communication is possible through many types of channels with different protocols. Therefore, this master thesis focuses on a generic solution that can be used in conjunction with most of the common interfaces used in military systems, namely: RS232, RS422, RS485, and *Controller Area Network* (CAN). Furthermore, the research will focus on the security this device alone can add, security from an end-to-end perspective will be evaluated using a more general approach. Hence, protection against malicious code and malware that have infected the end systems will not be a part of such mechanism and should be handled by the systems. As should also any demands on redundancy. To clarify, the main focus lies on the research for how to create a filtering device for digital communication, i.e., how to design, develop, certify, and integrate it into an existing system.

1.5 Outline

The rest of this thesis is organized as follows:

Chapter 2 Explains necessary background about computer security, security models, intrusion detection, Anagram, and SPARK etc.

Chapter 3 Describes the threat model, e.g., which attacks that are possible and plausible.

Chapter 4 Describes and evaluates the test results for the filtering security mechanism.

Chapter 5 Discusses how the proposed solution could be used as a countermeasure in actual systems. The chapter also discuss the importance of end-to-end security and how it should be applied.

Chapter 6 Summarizes the report with our concluding remarks. This chapter also contains directions for future work.

2

Computer Security Concepts

This chapter describes the fundamental concepts and techniques used in this thesis. It will introduce the concept of security models, intrusion detection systems, intrusion prevention systems, Anagram, and SPARK as well as defining central terms. These are key concepts when it comes to the design process, development, certification, and integration of a device that needs to be able to offer high assurance, i.e., it can ensure that the assets are being protected in a satisfactory way.

2.1 Computer Security

The *National Institute of Standards and Technology* (NIST) defines *computer security* as follows in their handbook for computer security [6]:

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

This quote introduces three key terms that characterize the fundamental security objectives for both data and information services, i.e., *confidentiality*, *integrity*, and *availability*. Stallings and Brown [7] choose to define these terms as follows:

Confidentiality This term covers two related concepts:

Data Confidentiality Assures that private or confidential information is not made available or disclosed to unauthorized individuals.

Privacy Assures that individuals control or influence what information related to them may be collected and stored and by whom and to whom that information may be disclosed.

Integrity This term covers two related concepts:

Data Integrity Assures that information and programs are changed only in a specified and authorized manner.

System Integrity Assures that a system performs its intended function in a unimpaired manner, free from deliberate or inadvertent unauthorized manipulation of the system.

Availability Assures that a system works promptly and services is not denied to authorized users.

These three concepts form the *CIA triad* which can be seen in Figure 2.1.

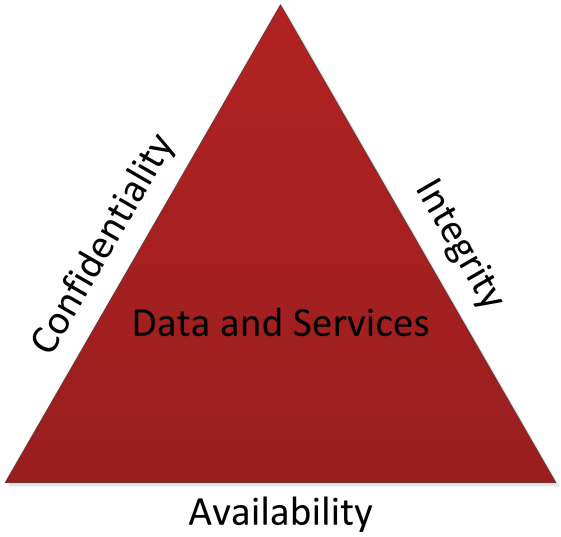


Figure 2.1: The security requirements triad.

Although the use of the CIA triad to define security objectives is well established; Stallings and Brown [7] point out that some see the need to include additional concepts to give the complete picture. Two of the most commonly mentioned are:

Authenticity The property of being genuine and being able to be verified and trusted; confidence in the validity of a transmission, a message, or message originator. This means verifying that users are who they say they are and that each input arriving at the system came from a trusted source.

Accountability The security goal that generates the requirement for actions of an entity to be traced uniquely to that entity. This supports non-repudiation, deterrence, fault isolation, intrusion detection and prevention, and after-action recovery and legal action. As truly secure systems are not an achievable goal, one must be able to trace a security breach to a responsible party. Systems must keep records

of its activities to permit later forensic analysis to trace security breaches or to aid in transaction disputes.

2.2 Security Models

Two historical facts highlight a fundamental problem that needs to be addressed in the area of computer security [7]. First, all complex software systems have eventually revealed flaws or bugs that subsequently needed to be fixed. Second, it is very difficult, in fact probably impossible, to build a hardware and software system that is not vulnerable to a variety of attacks.

Providing strong security involve both design and implementation. It is thus very difficult to ensure that the design of any hardware or software module in fact does provide the level of security that was intended. This may result in unanticipated security vulnerabilities. Even if the design is correct in some sense, it is probably hard to actually implement it without errors or bugs which also could lead to vulnerabilities.

These problems have led to the development of a method to prove, logically or mathematically that a particular design does satisfy a stated set of security requirements and that the implementation of that design faithfully complies with the design specification.

A security model is thus an abstraction that provides a conceptual language for administrators to define security policies. Typically, security models define hierarchies of access or modification rights that members of an organization can have to objects.

2.2.1 Bell-LaPadula

The *Bell-LaPadula* model [8, 9] is perhaps the most influential computer security model. It was developed in the 1970s as a formal model for access control. It consists of three basic elements:

Subject (S) The entity capable of accessing objects, e.g., a user (via a process).

Object (O) The resource to which access is controlled, e.g., records, files, and messages.

Access right The way in which a subject may access an object. Could concern read, write, and execute.

Each subject and object in the model is given a *security class*, denoted $f(S)$ and $f(O)$ for subject **S** and object **O** respectively. These classes form a strict hierarchy and are refereed to as *security levels*. One example, used in the military, is the classification scheme:

`Unclassified` $<$ `Restricted` $<$ `Confidential` $<$ `Secret` $<$ `Top Secret`

When multiple security levels are defined, the requirement is referred to as *multilevel security*. The general concept of a confidentiality-centered multilevel security is that a subject at a high security level may not pass information about an object to a subject at

a lower level. A multilevel security system must implement the following three properties to enforce confidentiality [7]:

ss-property (No read up) A subject can only read an object of less or equal its security level; $f(S) \geq f(O)$.

***-property (No write down)** A subject can only write into an object of greater or equal its security level; $f(S) \leq f(O)$.

ds-property A subject may grant other subjects access to an object based on the discretion of the owner, constrained by the *mandatory access control* (MAC). A subject can therefore only grant access to objects for which it has the necessary authorization and which satisfy the MAC rules.

These properties provide confidentiality and thus, no access is allowed in a system that does not satisfy these three properties. Note that the model does not say anything about the other properties in the CIA triad. Integrity cannot be assured since it is possible to write up, i.e., a subject can overwrite information in a greater security level in an unauthorized manner.

2.2.2 Biba

The Bell-LaPadula model, as previously mentioned, deals with confidentiality and thus is concerned with unauthorized disclosure of information. The *Biba security model* [10] is instead intended to deal with integrity, i.e., the unauthorized modification of data. This could be useful if certain data should be visible on multiple or all security levels, but only modifiable by certain subjects in a controlled way.

The basic elements of the model are the same as for Bell-LaPadula, i.e., there are subjects and objects. Each subject and object is assigned an integrity level, denoted as $I(S)$ and $I(O)$ for subject **S** and object **O** respectively.

A system must implement the following properties to enforce integrity:

Simple integrity (No read down) A subject can read an object only if the integrity level of the subject is less or equal to the integrity level of the object; $I(S) \leq I(O)$.

Integrity confinement (No write up) A subject can modify an object if the integrity level of the subject is greater or equal to the integrity level of the object; $I(S) \geq I(O)$.

Invocation property A subject can invoke another subject only if the integrity level of the first subject is greater or equal to the integrity level of the second subject; $I(S_1) \geq I(S_2)$.

A perhaps more elaborate and practical model is the *Clark-Wilson security model* [11]. It is aimed towards commercial applications rather than military. In short it is based on two concepts that are commonly used to enforce commercial security policies, namely:

Well-formed transactions A subject should not manipulate data arbitrarily, but in a constrained way that preserves or ensure the integrity of it.

Separation of duty among subjects Any subject permitted to create a well-formed transaction may not be permitted to execute it.

2.2.3 Chinese Wall

The *Chinese Wall security model* [12] concerns both integrity and confidentiality by using both optional and mandatory access concepts. It was developed for commercial applications in which conflicts of interest can arise. This model is of course also applicable to the military, but between different systems rather than organizations. The elements of the model are the following:

Subject (S) Active entity that may wish to access objects. User and process etc.

Information Corporate information organized into a hierarchy with three levels:

Object (O) Individual resource containing information, each concerning a single corporation. Records and files etc.

Data set (DS) All objects that concerns the same corporation.

Conflict of interest class (CI) All data sets whose corporation are in competition.

Access rules Rules for read and write access.

This model does not assign security levels to subjects and objects as Bell-LaPadula and Biba and in that sense is not a true multilevel security model. Instead, the access rights of a subject are determined by previous accesses. The basic is that subjects are only allowed access to information that is not held to conflict with any other information that they already possesses. Once a subject accesses a **DS**, a wall is set up to protect information in other **DSs** in the same **CI**. Two rules are needed to enforce the Chinese Wall model:

Simple security property A subject **S** can read an object **O** only if:

- **O** is in the same **DS** as an object already accessed by **S**, ***OR***
- **O** belongs to a **CI** from which **S** has not yet accessed any information.

***-property** A subject **S** can write to an object **O** only if:

- **S** can read **O** according to the **Simple security property**, ***AND***
- All objects that **S** can read are in the same **DS** as **O**.

2.3 Intrusion Detection and Prevention Systems

Intrusion detection is a service or process that monitors and analyses events in order to find attempts of gaining unauthorized access to system resources. If such an event is discovered, the typical action is to send a warning to the security officer and log that event [1]. This is referred to as a *passive system*. A more simplified explanation is to see an *Intrusion Detection System* (IDS) as a burglar alarm, the system cannot prevent anyone from breaking in, but it can warn and record the event.

A *reactive system* or an *Intrusion Prevention System* (IPS) contains all capabilities as the IDS, but with the possibility to also stop the intrusion, e.g., exit the user session or shut down the network [13].

An IDS/IPS can either be placed at the host or connected directly to the network. A host-based IDS/IPS is configured to only monitor an interface or an application of a certain host. While a network-based IDS/IPS monitors traffic over a defined network [7]. In general, there are four outcomes for an IDS/IPS. These are illustrated in Table 2.1.

	Intrusion	No Intrusion
Alarm	True Positive	False Positive
No Alarm	False Negative	True Negative

Table 2.1: Classification outcomes.

True Positive and *True Negatives* are desired results. The first one is equal to when an alarm is activated due to intrusion; the second is equal to when no alarm is activated as there is no intrusion. A false alarm, i.e., *False Positive* is a more interesting case. These are crucial to avoid since it can prevent legitimate data from being transmitted. In most cases a false positive alarm must be examined by a supervisor in turn to determine that the alarm can be disregarded. Therefore, many false alarms can be very time-consuming for the officer. The last, namely *False Negative* is when an intrusion is evading detection.

2.3.1 Signature-based Detection

There are two general approaches for how an IDS/IPS could operate in order to detect an intrusion. *Signature-based detection* is perhaps the most common approach. It monitors the traffic and compares the packets against a database with signatures of known attacks or suspicious activity, i.e., a type of *blacklisting*. A signature is based on an analysis of the behavioral pattern of the attack. A signature database can become very large after some time which can have impact on the speed of the communication. Another problem is previously unknown attacks, the so-called zero-day attacks. Since the system is only aware of already known attack signatures, zero-day attacks are unlikely to be detected [7]. Also, signature-based detection systems are considered very limited due to the fact

that it needs frequent updates in order to discover new attacks [14].

2.3.2 Anomaly-based Detection

In contrast to signature-based detection, where you store patterns for suspicious activity, *anomaly-based detection* learns what is normal for the system and reacts to abnormal patterns, i.e., a type of *whitelisting*.

One of the main issues with anomaly-based detection is the high rate of false positives because most protocols keep changing. A more unusual event triggered by the user can easily be recognized by the IDS/IPS as an intrusion attempt [7]. The main advantage is that it probably will detect zero-day attacks since an attack is likely to consist of patterns never seen before and thus will be classified as an anomaly.

There exists a number of anomaly based detection approaches, some utilize machine learning and statistical methods [15]. PAYL is a shortening for *PAYLoad anomaly detection* and was one promising approach for a self-learning detection system. In order for it to learn what is considered as normal traffic, PAYL has a training phase where it creates a profile. The profiles uses 1-gram (equal to a single byte) to build a byte-frequency distribution model. In this profile each 1-gram of a packet is counted such that the average frequency, as well as the variance and standard deviation is known. In operation, each 1-gram of a message is analyzed and if the value exceeds given thresholds, an alarm will be triggered. PAYL is designed for networks with high speed and is applicable to any network. Test results have shown that it has a low false positive rate and a high detection rate [16]. Bolzoni et al. [17] further develops PAYL into a system called *POSEIDON*, but uses the output of self-organized maps instead of the length of the payload. The authors claim that it results in less models and with higher detection rate. However, the problems with some systems, like PAYL, are that they often are unable to detect stealthy worms and targeted attacks. Kolesnikov and Lee [18] created such a worm which gathers data from the network. The data is then analyzed in order to find frequently occurring byte sequences. These sequences are added as padding, surrounding the malicious payload such that the worm can mutate itself. This causes the payload to be considered as normal since a vast amount of the data is commonly recurring, i.e., the amount of malicious data relative to the padding of normal data is low enough to pass the set threshold. Kolesnikov and Lee [18] showed that the polymorphic worm using this technique was able to evade anomaly-based detection systems such as PAYL. As mentioned earlier in this section, signature-based detectors are vulnerable to zero-day attacks. But, zero-day attacks may be the most important to prevent in military systems because if these systems are attacked, it is probably with new and custom made attacks. This is due to the fact that the systems often use highly customized protocols and that the threat agents could have access to great amount of resources.

2.4 Anagram

Our approach is based on *Anagram* [19] because of its impressive results regarding detection rate, low false positive rate, and performance. Anagram is a content anomaly detector that models a mixture of high-order *n-grams* designed to detect anomalous and suspicious packet payloads. A n-gram is a contiguous sequence of n items from a given sequence, e.g., a 6-gram is six contiguous bytes in a message.

Anagram uses automated statistical learning to train a content model with normal traffic of a system, i.e., learn the characteristics of normal and attack-free data. The model is then used to detect anomalies which the system needs to take action against. New attacks will contain data that has never been seen before, i.e., the packet will be considered anomalous. A binary-based model outperforms a frequency-based model due to the fact that the packets of real attacks or information leakages will contain more n-grams not observed in training than the normal packets used to train the model.

Anagram uses a *Bloom filter* [20] and a binary-based detection model. A Bloom filter is basically a bit array of m bits. Any individual bit i in this array is set if *the hash of an input value mod m is equal to i*. By using a Bloom filter, Anagram gets memory and computationally efficient. The filter is a set-based data structure used to represent the set of observed n-grams. A set entry is represented with just a few bits instead of n bytes for each n-gram. This is similar to a hash table in the sense that it is a one-way data structure that can contain many items, but generally is orders-of-magnitude smaller. Operations on a Bloom filter are $O(1)$, keeping it computationally fast. It cannot have false negatives, but may contain false positives due to possible collisions in the hash function. Although the false positive rate may be optimized by changing the size of the bit array as well as using several hash functions. It is better to use multiple hash functions due to the fact that it is highly unlikely that if one hash function collides between two items, several other functions would also collide. Thus, all of the hash functions must be set in a Bloom filter for one to be able to verify that a certain item is present.

A visual representation of Anagram can be seen in Figure 2.2. In the figure it is displayed how a NMEA 0183 message is inserted into a Bloom filter by using 6-grams and k hash functions. The NMEA protocol is commonly used for communication with marine electronics, like sonars and GPS. A n-gram n is represented by a 1 at position i in the Bloom filter if:

$$i = Hash_x(n) \bmod m$$

Where $Hash_x(n)$ is the hash value of n produced by the *x:th* hash function, $0 \leq x \leq k$, and m is the length of the Bloom filter. As we can see in the figure, the 6-gram `$GPRMC` is represented at position: 1, 2, 15, and 17 in the array, while `GPRMC1` is represented at position: 6, 8, 17, and 21. Note that we have a collision at bit 17 in the filter. This since two different hash values map to the same bit. As stated earlier this can occur since a Bloom filter is a probabilistic data structure. However, this behavior can in a sense be neglected since several hash functions are used.

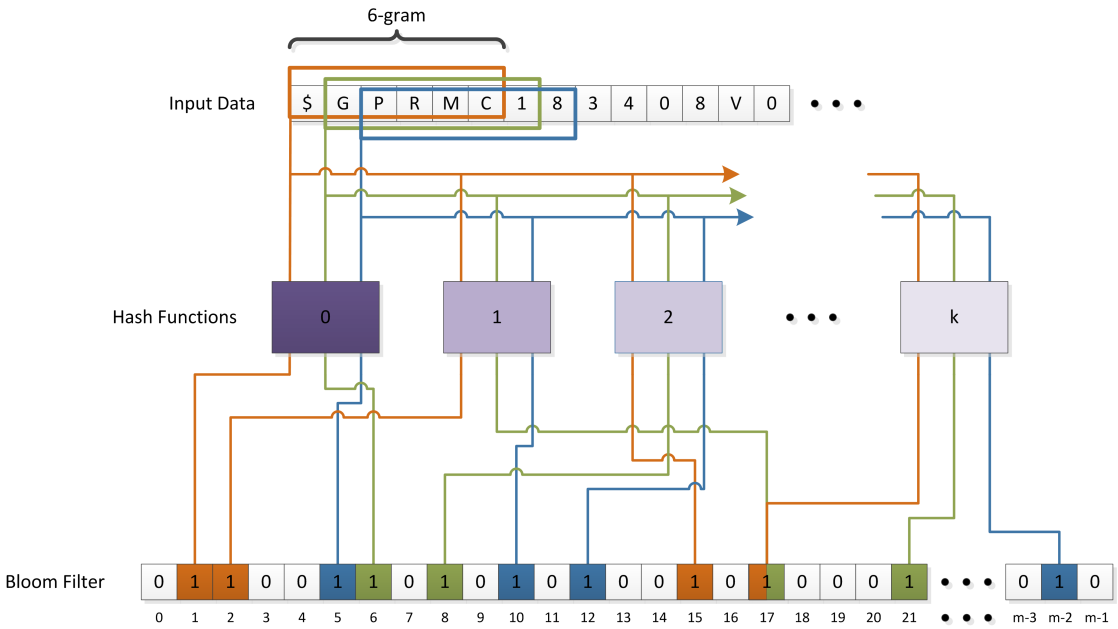


Figure 2.2: Inserting 6-grams into a Bloom filter.

During the training phase each unique n-gram is stored in the Bloom filter by using several hash functions and the modulo operation. The pseudocode for training phase is seen in Algorithm 1.

Algorithm 1 Training of the model

```
1:  procedure TRAIN(trainingdata)
2:      for each packet in trainingdata do
3:          for each n-gram in packet do
4:              for x = 0 to k do
5:                  hashes[x] ← Hash_x(n-gram)    ▷ hashes is an array used to store all
    the hash values for n-gram
6:              end for
7:              for each hash in hashes do
8:                  i ← hash mod m    ▷ i is the position at which n-gram is represented
9:                  bloomfilter[i] ← 1
10:             end for
11:         end for
12:     end for
13: end procedure
```

In the next part, the analyzing phase, the same steps are taken, but instead of inserting values into the filter, Anagram checks if the bits at the calculated positions are set or not. Should all of these bits be set, one can with great probability say that the

currently analyzed n-gram has been seen during training. If one or more of the bits are not set, one can with absolute certainty say that the n-gram has not been seen before and thus is classified as anomalous. Anagram then scores each packet based on the number of unobserved n-grams it contains. The score is compared against a threshold value which determines how many anomalous n-grams that can be tolerated before the packet is considered as erroneous, e.g., no more than 10 % of the total amount of n-grams in a message can be anomalous. This is done according to this formula:

$$Score = \frac{N_{new}}{T} \in [0, 1]$$

Where N_{new} is the number of n-grams not seen during the training phase and T is the total number of n-grams in the packet. The pseudocode for analyzing phase is showcased in Algorithm 2.

Algorithm 2 Analysis of message

```
1:  procedure ANALYZE(packet)
2:      nrOfAnomalous ← 0
3:      nrOfTotal ← number of n-grams in packet
4:      for each n-gram in packet do
5:          for x = 0 to k do
6:              hashes[x] ← Hash_x(n-gram)
7:          end for
8:          for each hash in hashes do
9:              i ← hash mod m          ▷ i is the position at which n-gram is represented
10:             if bloomfilter[i] = 0 then
11:                 nrOfAnomalous ← nrOfAnomalous + 1
12:                 break
13:             end if
14:         end for
15:     end for
16:     if nrOfAnomalous/nrOfTotal < Threshold then
17:         Forward packet
18:     else
19:         Discard packet
20:     end if
21: end procedure
```

Wang et al. [19] show how they achieve high detection rate while maintaining a low false positive rate. Their tests suggest that Anagram has less than a 0.01 % false positive rate along with a 100 % detection rate for a variety of worms in the used data set based on network traffic.

2.5 SPARK

Barnes, the author of [21], introduces SPARK by defining it as a high-level programming language designed for writing software for high-integrity applications. These applications include both safety and security requirements. Safety critical applications are said to be those where life, environment or even expensive property are at risk if the applications contain flaws. Security, on the other hand, concerns the confidentiality and integrity of information as well as the access to it, i.e., availability. These properties are beneficial to any type of program and the advantage of SPARK is that it enables errors to be prevented or detected in a more predictable manner.

Barnes writes that the general goal of SPARK is to provide a language which increases the likelihood of the program behaving as intended. There is a reduced risk of disaster arising, caused by any residual errors in the program [21].

SPARK encourages the development of programs in an orderly fashion with the aim that the program should be correct by virtue of the techniques used in its construction. The contracts are the core of SPARK, and are used to encode the intentions and requirements for certain components of a program, i.e., the specifications. The contracts allow the SPARK technology to perform information-flow analysis and formal verification of the implementation against the specifications [22]. Consider the following program specification:

```
procedure Add (X : in Integer);
```

This does not tell much. The only thing one could say is that there is a procedure called **Add** and that it takes one parameter of type **Integer** called **X**. This is enough to compile the code, but it says nothing at all about what **Add** actually does. In fact, it might do anything or even nothing at all. It definitely does not have to add anything or even use the variable **X**. It could, for example, multiply two unrelated global variables and store the result in a third. However, by using contracts the specification could be changed to:

```
procedure Add (X : in Integer) with
        Global => (In_Out => Result);
```

This states that the global variable **Result** is the only global variable that can be used. **In_Out** expresses that the initial value of **Result** must be used (**In**) and that the same variable will be assigned a new value (**Out**). As of now, it is known that **Add** will produce a new value of **Result** and that it will use the initial value of **Result** as well as the value of **X**. In SPARK an input parameter must be used to prevent ambiguous use of variables. It can also be concluded that the procedure may not affect anything else, e.g., it cannot store the result in an arbitrary unknown global variable nor have any malicious intent.

However, it is still not anything that requires **Add** to perform an addition to compute the new value of **Result**. SPARK can assure that **Add** uses addition by adding a new contract as seen below:

```
procedure Add (X : in Integer) with
        Global => (In_Out => Result),
        Post   => Result = Result'Old + X;
```

The annotation **Post** is called a postcondition and explicitly says that the final value of **Result** must be the result of adding its initial value (**'Old**) to the value of **X**. It is also possible to define preconditions. For example, it could be required that **X** is strictly positive; this could be specified by (**Pre**) as seen here:

```
procedure Add (X : in Integer) with
        Global => (In_Out => Result),
        Pre    => X > 0,
        Post   => Result = Result'Old + X;
```

To guarantee that the value of **Result** only depends on **Result** and **X**, the annotation commencing **Depends** is used. To conclude this example the complete specification is given:

```
procedure Add (X : in Integer) with
        Global  => (In_Out => Result),
        Depends => (Result => (Result, X)),
        Pre     => X > 0,
        Post    => Result = Result'Old + X;
```

These contracts are checked statically by SPARK tools before execution to prevent runtime errors. Apart from strengthening the interfaces via, "programming by contracts" as described earlier in this section, SPARK also include a number of other design factors used to reduce the number of residual errors in the code. Barnes [21] defines these as:

Logical soundness For the behavior of a program to be completely predictable, it is vital that the language, in which it is written, is precise. Consider, for example, the following statement:

```
Y := F(X) + G(X);
```

This is allowed in most languages, but the order of evaluation is not defined. If **F** and **G** have side-effects then it is possible that **Y** is assigned a different value depending which of the functions are called first. SPARK does not suffer from this ambiguity since the functions cannot have side-effects, this since they are true mathematical functions which may observe the state of some part of the system, but cannot change that state.

Simplicity of language definition Simplicity is generally considered good since it reduces the risk of a program meaning something different from what it appears to mean. Generally one can expect that simplicity of definition means simplicity of reasoning which implies simplicity of supporting tools and simplicity of testing. If tools are simpler they are more likely to be reliable such that risks are reduced.

Expressive power The language must not be too simple such that it cannot provide key benefits of a modern language and its concept of information hiding. One must also be able to make stronger assertions about the values of variables and their relationships than traditionally in imperative programming languages.

Security and integrity The language must be secure in the sense that it should not have rules that cannot be checked with reasonable effort. Also, the behavior of any program must be within certain well defined bounds. This is achieved by ensuring that the program does not stray outside a well-defined computational model. For example, a program should not be able to jump or write to arbitrary locations.

Verifiability Safety critical programs have to be shown to be correct. In order to do this it is necessary that the language constructions are such that a program can be subjected to rigorous mathematical analysis.

Bounded space and time requirements In order to prove that a program functions as expected, it is necessary to be able to predict the amount of storage space that it requires. Thus, it must be possible to predict the maximum amount of space required prior to execution. General dynamic storage allocation is thus prohibited which means that recursion, the declaration of arrays with dynamic bounds and pointer types are forbidden in SPARK. Bounding time is more difficult, but just as important. SPARK is designed such that, non-terminating situations, e.g., infinite loops, generates a warning.

2.6 Serial Communication Architecture

Serial communication is the process of sequentially sending one bit at a time over a digital channel or computer bus. It is often used for long distance communication and in computer networks due to signal integrity and high transmission speed. A generally known and used serial communication standard is Ethernet.

RS-232/422/485 [23] are all examples of standards for a serial communication transmission of data. RS is shorthand for just *Recommended Standard*. These have been used in personal computers, but has been replaced by *Universal Serial Bus* (USB). Although these protocols are still commonly used within networking equipment, industrial machines, and military systems since they are not limited to a cable length of five meters like USB. Furthermore, the RS standards do not require complex software support since they do not incorporate a protocol for transfer nor require an external device for decoding received data. Some key benefits of these standards are:

RS-232 Single ended and can drive high voltage devices.

RS-422 Differential operation and a cable length of up to 1 000 m.

RS-485 Similar to RS-422 but allows an increased number of drivers and receivers (up to 32).

The *Controller Area Network* (CAN) [24] is ideally suited to many industrial protocols. It is often used because of a low cost, good performance, and upgradeability. The standard does not only provide a physical media for communication, like RS-485, but also defines necessary mechanisms to avoid data collisions. It allows microcontrollers and other devices to communicate with each other, this without a host computer.

All of the mentioned standards have official documentation that provides requirements, specifications, guidelines, and characteristics. This documentation can be used to ensure that the processes are fit for their defined purpose [25].

3

Threat Model

In order to argue that a device is secure, one must be able to show a threat assessment with positive results. This chapter highlights and discusses possible attacks which we found during the project. The attacks will be evaluated and analyzed. Referring back to the research questions, this chapter cover which attacks that are possible and plausible as well as describes the current threat assessment on the system. As the device is designed to be used in military systems that can be placed in a hostile environment, the reader should have in mind that the threat agents that may want to damage or destroy this device may have a lot of time and resources.

3.1 Attacks and Vulnerabilities

Figure 3.1 shows a rough sketch of the overall lifecycle of the filtering security mechanism. It contains the main security problems during each phase of the lifecycle. As can be seen, several of the more general and recurring threats concern social engineering, inadequate knowledge, inadequate documentation, authentication, and authorization. Firstly, through social engineering an adversary could try to manipulate authorized personnel and bypass security mechanisms. In short, the concepts builds upon the thought that it is easier to ask for a password than to break into the system. The threat agent may want to gain information of a device or insert malicious code to alter its execution. Social engineering can be avoided, or at least hampered, by educating the users to be more aware of these attacks [7]. Secondly, improper knowledge can of course affect every part of the lifecycle, code could be developed incorrectly or results could be interpreted wrong. Proper knowledge of the filtering mechanism and the target system is a must, regardless of which phase one is working in. Thirdly, poor documentation can result into misunderstandings and human errors. Therefore, documentation should always be written with great care. Faulty documentation could thus, indirectly, cause the system to have an altered behavior. At last, authentication and authorization is crucial to provide

accountability. In short, if the personnel working on the device cannot be identified or have more access than they need, the filtering mechanism cannot be trusted. It should be pointed out that personnel that handle the control of permissions should also have restriction, e.g., it should not be possible for the permission security officer to add a higher level of clearance to himself.

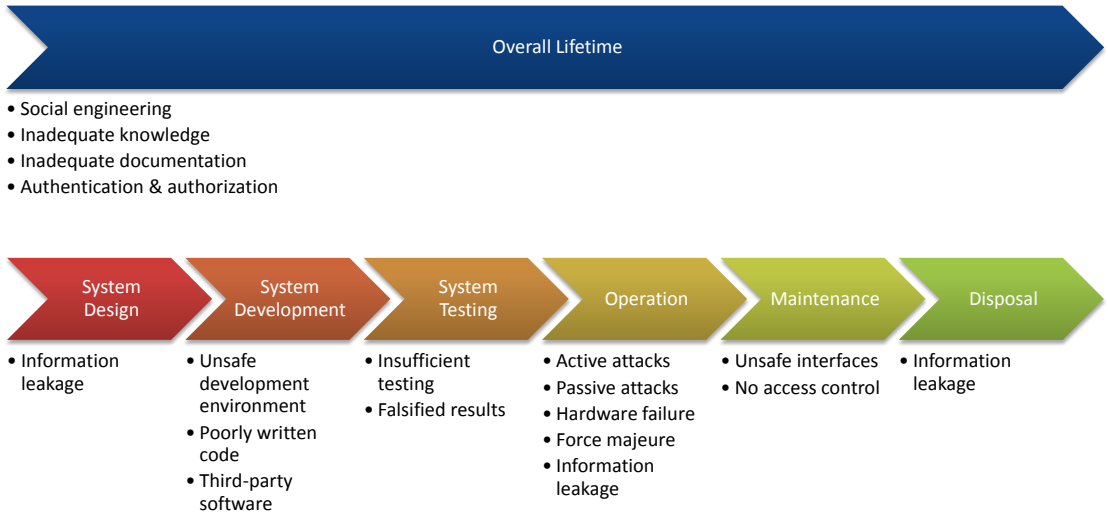


Figure 3.1: The overall lifecycle of a filtering security mechanism and threats in each phase.

The following subsections discuss more specific vulnerabilities during each phase and critical components, including those seen in the figure.

3.1.1 System Design and Development

The system design is the first phase in the lifecycle where the device is formed and planned. Information leaked from this phase can help a threat agent find vulnerabilities to construct an attack against the device. Therefore, details about the design process should be kept hidden from unauthorized personnel.

The development of the systems concerns both the environment in which the device is developed and the personnel that have access to it. Malicious code that is implemented at this stage may be very hard to discover when the system is deployed. Therefore, development of the system must be performed in a secure environment. The development environment should only include the necessary software. All unknown applications should be removed before the actual development starts in order to avoid malicious software. The development environment should also not have any connection to ”the outside world”. This means that no connection to the Internet for example. Every software and all documentation that is needed within the environment must be carefully reviewed and analyzed to guarantee that it does not contain any malware or have some malicious intent. Items that pass the review might enter the environment through a special ”gateway”, e.g., a data diode, which can ensure information is only sent in one

direction. This environment and the ”gateway” needs to be monitored carefully. Also, strong authentication and authorization is a must. Preferably it should have a multifactor authentication approach, e.g, one-time passwords combined with biometric tokens, like fingerprints. The solution should also be based on a strong encryption.

Source Code

In order for users to trust the device its source code and its compiler need to be auditable, which means that it should be possible to review all parts of the code and functions that are used. The code needs to have comments such that each function can be understood and controlled. In Section 2.5, the usage of contracts in SPARK are described. The contracts are used to verify, assure, and prove that the subprograms do what they are intended to do (i.e., recall from the example with the **Add** function). The software will most definitely have to be updated in the future, such process can be a serious challenge if tools used in the development is deprecated. Therefore, the software should be built with as few dependencies as possible. The code should be handled with a version control tool, which is usually the common way when developing software. It can also keep track of the author of certain code parts, which can come in handy when code is rewritten and updated in the future. The compiler needs to be completely auditable and well documented as well. Otherwise, one cannot be sure that the final program does what it actually should according to the implementation.

Tools for storage of the code should also be considered, where is it stored? Who can access? How can it be verified to be genuine? How do we keep track of what changes were made and by whom? Once again authentication, authorization, authenticity, and accountability needs to be carefully considered.

3.1.2 System Testing

Testing of the software is needed to ensure that the software meets the specified requirements. We argue that the testing should be performed in a debugging environment separated from the operational environment such that debug messages does not reveal sensitive information. The development environment and the testing environment should be separated such that a breach in one of them does not damage the other. This is based on the thought that disclosure of the source code could be more severe together with testing data and vice versa. Badly written documentation can pose a vulnerability as error codes are misunderstood and erroneous states are disregarded. Much of the responsibility is put on the developers who writes the test cases, a certain bug might never be found, both intentionally and unintentionally. Here, the choice of programming language have a big impact. A language like SPARK natively prevents the developer to introduce certain bugs possible in languages like C and Java. One should also ensure that specified events trigger the logging procedure.

3.1.3 Operation

When the device is active in a live environment it is also exposed to attacks targeting the physical interfaces. In order to secure the device against physical hardware tampering we recommend that the casing, which surrounds the hardware, is sealed with a tamper-evident security seal. There exists a lot of different types of security seals commercially off the shelf. The device should also be built to avoid covert channels, e.g., emitting radiations. Another type of attack is *denial-of-service* (DoS) attacks which is when the adversary attacks the availability property of the system. Such an attack can simply be performed by transmitting valid packets to a level where the device cannot process the packets and crashes. Anagram has natively no capabilities to prevent DoS attacks since it just analyses the content on a packet, not the frequency of it. We also argue that it is possible to perform a DoS attack by sending messages which triggers the logging function and eventually fills up the memory where the log files are stored. We suggest that by implementing certain thresholds it is possible to avoid such attacks. There are also circumstances which have natural causes, e.g., power outage. If such an event occurs and the device needs to be restarted we suggest that the software with its configuration is stored in a read-only memory, as long as the memory is not damaged it can be rebooted into its initial state. However, it is still important that no traffic can bypass the device when it is down. Fault tolerance in general is a concept that needs to be considered, maybe two filtering devices should be running in parallel, such that the backup can take over if the primary fails.

Recalling from Section 2.4 about Anagram, the system has a training phase where normal traffic is defined. This training phase should only be possible in a secure environment such that an adversary cannot train the device with malicious traffic in order to execute an attack undetected. Therefore, such an option should be unavailable in operational mode. Nevertheless, we cannot stress enough that the training phase is the most vulnerable state. If the data used for training or the training environment itself cannot be trusted, the mechanism cannot ensure that it adds any security.

As with all attacks where the adversary modifies data in order to inject malicious code, the data will not match against what is known as valid traffic any more, which basically is the whole point with Anagram, and it will be recognized as anomalous traffic. Another type of attack is the *replay attack* where the adversary captures packets and later retransmits them to get an unauthorized effect. As Anagram cannot protect against this kind of attack we claim that the responsibility lies on the end-systems to protect against such attacks, we discuss this further in Chapter 5.

Network

The device is supposed to be placed between two end-systems, monitoring traffic in both directions through an interface. It is not possible through this device alone to detect if someone is sniffing packets. This is simply because sniffing is a passive attack that does not emit anything that could be detected. We claim that in order to secure the whole connection, without modifying the end-systems, one needs to secure the network

connections with perimeter protection. Even though the traffic rate consists of valid packets the device must be able to process a high flow of valid traffic, to prevent the device from becoming a bottleneck.

Logging

The logging is an important function in the device as it gives the administrators knowledge of events. During runtime it is important that the log function cannot be turned off as an adversary could execute an attack that will go undetected. For logs to be helpful there needs to be enough details in order to understand the event, however, too much details can be abused by an adversary, e.g., the device can be probed with different packets to identify patterns in the log file. One needs to determine how much information is enough from case to case based on the sensitivity on the data. Accessing logs can reveal a lot of information. Therefore, access to the logs should be restricted to a certain group of operators or administrators. Also, the permission to remove and edit logs should be considered.

Malicious Code

Despite necessary precautions, the threat agent may find a way to transfer malicious code, either targeting the device directly or the end-system. In either way the result can have great impact on its target, in worse case, very sensitive information is leaked, modified, or made unavailable. Malicious code that targets the end system, only passing through the monitored connection must be detected and dropped. Chapter 4 describes how malicious code can be detected during analysis. To prevent malicious code that directly targets the filtering device one could utilize the advantages of read-only memory (ROM).

This kind of memory physically prevents anything, including malware, to be written to the memory after the initial write of the program. Although it is secure, it can be impractical for maintenance. However, there exists memories that takes advantage of the benefits of ROM, but instead of writing the data to it the data is programmed into it. This type of memory is called programmable read-only memory (PROM).

3.1.4 Maintenance and Storage Media

A USB-port can facilitate the maintenance work, e.g., easy to plug-in a keyboard. However, such IO-interfaces should be avoided to the extent possible. It can pose a vulnerability, e.g, a threat agent could insert a memory stick with malicious code. Another issue to be considered is hard drive crashes, a hard drive does not last forever and will eventually deteriorate. If such an event occurs traffic should not be able to circumvent the device. In terms of fault tolerance, the authors suggest a replica to ensure a more reliable connection. Such type of question lies outside the scope of this thesis and should be considered by the architect of the complete system.

Technical support, related to maintenance work, is another phase that could constitute a vulnerability. Performing maintenance work or technical support should only be carried out by authorized personnel that have undergone a security assessment.

3.1.5 Disposal

Eventually all products are phased out in favor of a newer version or a better technique. Since the device has been configured with traffic from used protocols, we suggest that the memory should be destroyed when the device is disposed to prevent an adversary to find out the scoring level which can be used to avoid detection. Another reason is to protect the product itself from getting in the wrong hands.

4

Evaluation of the Anomaly-based Detection System

In this section the prototype is tested with two types of protocol data, one more complex and one simpler. The NMEA 0183 is in comparison with the other protocol much more comprehensive and is used in a variety of marine applications such as GPS receivers and sonars. The simple protocol, which we choose to call it, is used in a military application. We chose these two protocols as they represent two completely different types of protocols. However, we are aware of that the device needs to be tested with real protocols in order to ensure it fulfills the requirements.

The tests concern performance of the systems, the time to analyze messages, in order to ensure that the rate is high enough to meet the protocol specifications. The performance thus vary depending on which machine the tests are executed on. The testing environment we have used consists of an Intel i7-3720QM 2.60 GHz processor with 8.00 GB RAM. The system is also tested for false positive rate and its detection rate. Finally, for NMEA 0183, false positive rate is tested in relation to how much training data that is used.

4.1 NMEA 0183

This first part uses the protocol NMEA 0183 where NMEA stands for *National Marine Electronics Association*. NMEA 0183 operates at 4.8 Kbps and its successor, NMEA 2000, operates at 250 Kbps [26]. Since this protocol is released to the public there exists several generators which produce random sentences (NMEA messages). For this test we have used a generator [27], which follows the NMEA 0183 version 2.0 specification. The generator uses the current time in order to generate the first sentence, next sentences are generated one per second thus simulating a GPS with 1 Hz. Every third of the sentences have a relation to each other, which means that the location is somewhat similar. For

the next three sentences, the coordinates are completely different, thus giving a very wide amount of location spread over the earth. Below follows an example of a NMEA 0183 sentence from the test data:

```
$GPRMC,083748,V,6624.462,N,17620.882,W,25.9,4.36,150515,,E*46
```

The sentence consists of a lot of integers. We discussed the impact of integers, and other data types with wide range, have on anomaly-detection systems in Section 2.3.2. A short description of the sentence structure [28] follows:

$
: NMEA 0183 sentences always start with a dollar sign.

GP
: **GP** stands for GPS unit and identifies the source. In our test data set, all sentences start with **GP**.

RMC
: *Recommended Minimum sentence C*, these three letters identify the type of message. It exists many types in order to support all devices which NMEA operates on. In our tests we have used seven different types.

083748
: The time at which the data is captured. In this sample the time is 08:37:48 UTC.

V
: Navigation receiver warning. **A** is active and **V** is void.

6624.462,N
: Latitude, 66°24'27.72"N.

17620.882,W
: Longitude, 176°20'52.92"W.

25.9
: The speed over ground in knots.

4.36
: The angle which the target is moving.

150515
: The date, which here is 2015-05-15.

Empty
: A field for magnetic variation.

E*46
: The checksum of the sentence.

4.1.1 Detection Rate

For the first test we have created twelve sentences where a keyword is inserted into the sentence at arbitrary positions. The sentences do not exist in the training set and the keywords are created with characters which can be found in the training set. As we do not know the type of end system which the device is applied at, we do not know how a certain attack could look like. Therefore, the purposes with the keywords are that they should illustrate an attack where the threat agent has compromised a system and tries to leak information through the network. The keywords are: **RM ***, **GARM**, **CD .**, **PRAG ***, **RMV ***, **GREP ***, **SWEDEN**, **MERGE ***, **PRAG 16**, **SCREAM**, **GREVE**, and **SAVE**. The training set consists of 2 200 000 sentences and we can conclude that it is possible to have a threshold of 35 % or below in order to detect all keywords, i.e., have a detection rate of 100 %. A threshold of 35 % specifies that no more than 35 % of the total amount of n-grams can be anomalous. This threshold is only possible when the system is configured to use 12-grams. We want to threshold to be as high as possible such that the system can tolerate valid variations for coordinates and time, but at the same time be able to detect all sentences that include these keywords. However, one could argue whether these keywords are realistic, a threat agent could use a covert channels which may only use one bit in order to communicate information. To protect against covert channels the training set must be greatly increased and the threshold, the tolerance of anomalous n-grams in the messages, must be set to zero. However, there is a trade-off; by putting the threshold to zero the false positive rate increases a lot.

4.1.2 False Positive Rate and Training Data

This section presents what kind of false positive rate that can be expected. We show that the false positive rate is closely related to the amount of training data that is used. The system is configured to use 12-grams and have a threshold of 35 %.

At first the system is trained with 800 000 messages and is fed with the same 800 000 messages. In order to prove that the system is functioning, this run showed, as expected, a 0 % false positive rate. For the next test we use a set of 2 200 000 messages training data. For each run the data to be analyzed is increased, but the training data remains the same. It should also be pointed out that the messages in the analyzed data do not exist in the training data.

In Figure 4.1 the blue-solid-squared line represent the increase of false positives as the analyzed data is increased. The graph shows that it is possible to achieve a 0 % false positive rate for 100 000, 200 000, and 300 000 messages. However, in the next run with 400 000 messages the false positive rate increases to 2.28 % and continues to increase.

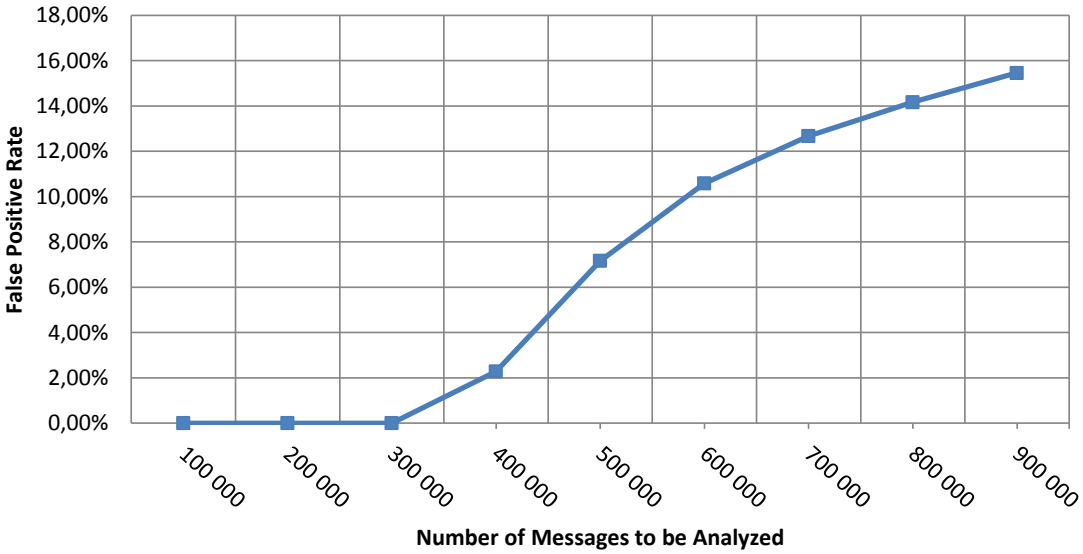


Figure 4.1: Number of false positives related to a constant amount of training data.

We cannot stress enough the importance of a sufficient amount of training data. This is especially true about NMEA 0183 that is a much more comprehensive than the simple protocol. As NMEA 0183 includes integers like coordinates, course, and speed, etc., which are rarely the same, the system must tolerate some variation in order to not report all messages as anomalous. More training data results in less false positives since more of the possible values of the protocol have been seen. This test showed that it is possible to achieve a 0 % false positive rate, even with a much more comprehensive protocol. However, we are convinced that the amount of training data can be increased in order to lower the false positive rate for greater data sets. It is hard to give any general directions how much training data that is sufficient, this since the size of the training set must be adapter to the protocols and the type of system it is applied to.

A Final Word Regarding the Training Data Set

In some military system it can be particularly important that a certain type of message is never blocked, for example it could be an alarm signal which must be let through. Anagram can guarantee that the message will not be blocked as long as the message is added in the training set. To guarantee that one or several critical packets are let through one must make sure that they exist in the training data set.

4.2 The Simple Protocol

The simple protocol is bit-oriented and has a message size of three to five bytes. The protocol requires a data rate of 9.6 Kbps. Due to company restrictions it is not possible

to publish any further details about the protocol or test the prototype with real data. However, we have instead developed our own test data generator in Java according to the protocol specification. The generator uses the strong random number generator in Java, *java.util.Random*, in order to generate the next bit in the message. The generator has the ability to determine whether the packet is valid or invalid. Depending on what packet is requested, the packet is either saved or disregarded. We define an invalid message by a message that breaks the specification, e.g., invalid header or invalid checksum. The message is normally very static and only allows a couple of bits to change, but the generator alters all bits for an invalid message. However, even though the complete message is invalid, the generator cannot ensure that there exist unique byte sequences which the system has not seen during the training phase. For this protocol we should underline that even if the protocol is bit-oriented, the system will still interpret each one and zero as a byte. This is to test if the solution can work with protocols that have a very low range of values.

4.2.1 Detection Rate

In this phase the system is trained with 300 000 valid messages. The system is then executed with 25 invalid packets that have been manually reviewed to be erroneous. Figure 4.2 plots four different thresholds with the same packet setup.

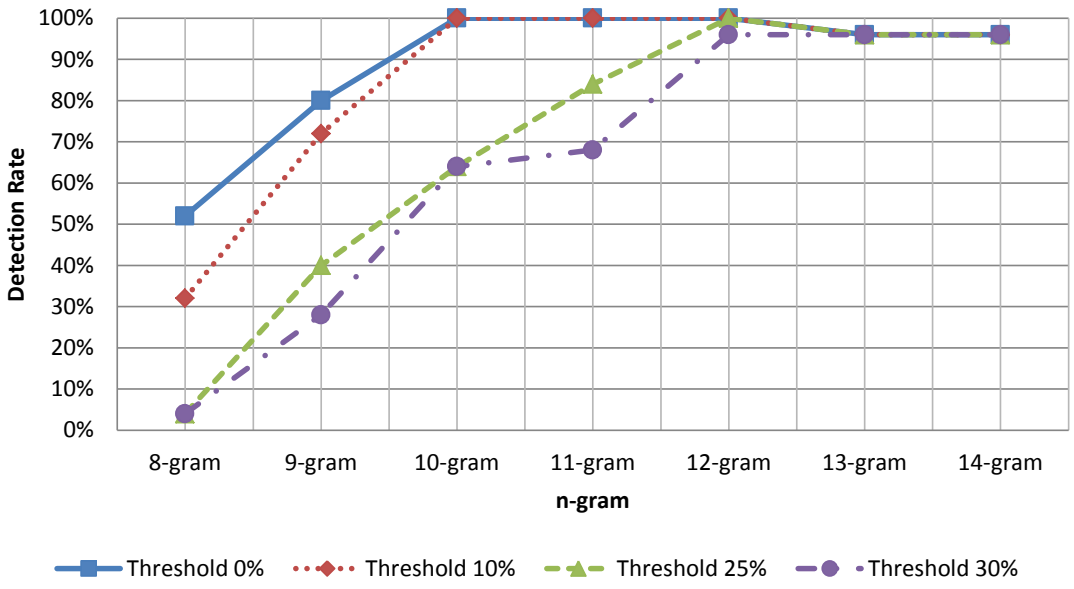


Figure 4.2: Detection rate for the simple protocol.

The plot shows a blue-solid-squared line which has a threshold of 0 %, i.e., no anomalous n-grams are tolerated in a packet, the line reaches 100 % detection for 11-gram, 12-gram, and 13-gram. The red-dotted-diamond line represent a threshold of 10 % and

has a similar appearance with 100 % detection for the same n-grams. The green-dashed-triangle line has a threshold of 25 % and has a 100 % detection only on the 12-gram setup. Finally, the purple-dashed-dotted-circle line has a threshold of 30 % and can achieve 96 % detection rates as maximum.

This test states that it is possible to achieve 100 % detection with enough training data and the size of the n-gram set to 12. This in conjunction with the threshold set to 25 %, i.e., 25 % or more of the n-grams in the message must be anomalous to consider the entire message as anomalous. The result is mostly due to that the message size is very small with only a few bits that could be altered each time. With 300 000 messages in the training set it is highly probable that all valid messages is included, i.e., more or less like white-listing.

4.2.2 False Positive Rate

This section presents the result where the system is tested for false positives using the simple protocol. The system is tested with two setups, firstly, the system is trained with 300 000 valid packets and then the system analyzes a file with another 300 000 valid packets, two separate files produced by the generator written in Java. For this setup we find no false positives. For the second setup the system is trained with the same packets that are analyzed, this test also shows a false positive rate of 0 % as expected.

4.3 Execution Time

The execution time is measured in seconds from that the operation starts until it is finished. For this test we have used the simple protocol. We have not performed this test for NMEA 0183 because the data rate should not depend on the protocol, but only on the size of the data set.

In the first test run the system is trained with 200 000 valid packets and thereafter executed with 200 000 invalid packets. The threshold is set to 20 % which means that, if more than 20 % of the n-grams of the packets has not been seen during the training phase the packet is classified as an anomaly. Figure 4.3 plots the execution time for both the training and analyze phase.

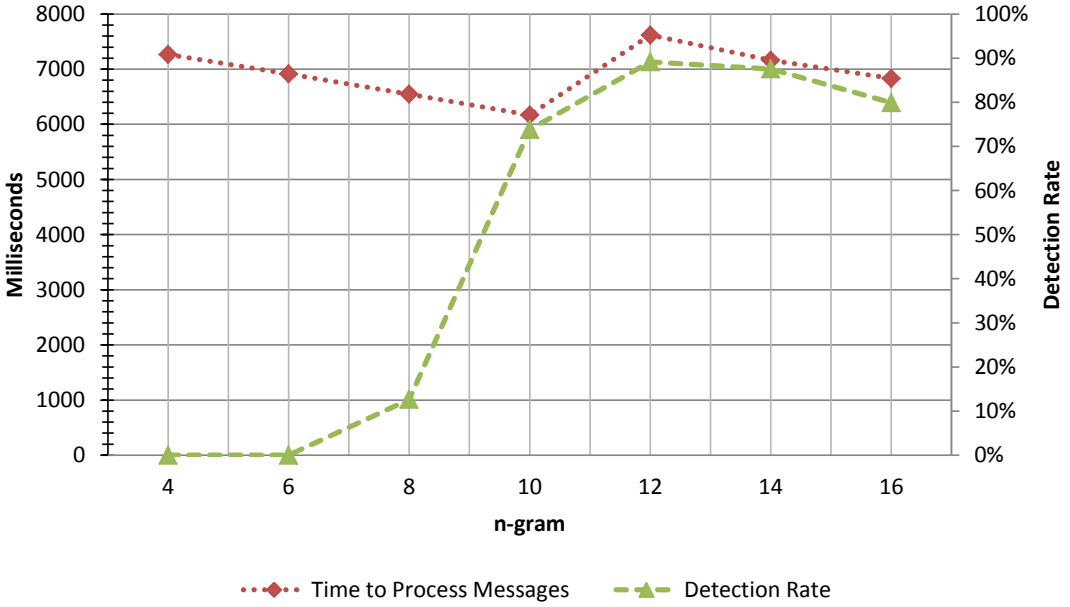


Figure 4.3: Shows how a higher detection rate relates to the time it takes to process the messages.

The chart shows two lines, the red-dotted-squared one shows the time it takes to process all packets for the given n-gram. The green-dashed-triangle line shows the detection rate. We estimate that the impact of the logging function adds an average of 27 μs per event. The calculated average transmission rate is 10.75 Mbps when there are no anomalous packets and 3.93 Mbps in a worst-case scenario when all of the analyzed packets are anomalous.

In the prototype, the system reads from a text file in order to simulate packets transferred through an interface. We are aware that this is not directly transmittable to the actual data rate of the system which should also consider the impact of which hardware the tests are executed on. However, the simple protocol performs at its lowest rate 3.93 Mbps when all packets are considered anomalous, and the protocol requires 9.6 Kbps. This means that the system has the ability to process about 400 times faster than required. NMEA 0183 requires 4.8 Kbps or 250 Kbps (for the newer version). For NMEA 0183 the system can process packets about 800 times faster and for NMEA 2000 about 15 times faster.

The test gives a clear indication that the implementation has very large margins compared to what the protocols need in order to function which conclude that the prototype will not become a bottleneck in the network.

5

Securing the Systems

Working together, both within and between nations, organizations, and companies, is vital to advance IT security according to a recent report [2]. 82 % of the companies with high-performing security, in the survey, practices collaboration with others. This deepens the knowledge of both security and current threats. *National Institute of Standards and Technology* (NIST) strongly encourages information-sharing in the NIST Cybersecurity Framework [29]. The framework is a risk-based compilation of guidelines that aims to help organizations identify, implement, and improve their cybersecurity stance. It is organized into five continuous functions:

Identify Develop the fundamental understanding to manage cybersecurity risk to organizational systems, assets, data, and capabilities.

Protect Develop and implement the appropriate safeguards, prioritized through the risk management process of the organization, to ensure delivery of critical infrastructure services.

Detect Develop and implement the appropriate activities to identify the occurrence of a cybersecurity event.

Respond Develop and implement the appropriate activities, prioritized through the risk management process of the organization (including effective planning), to take action regarding a detected cybersecurity event.

Recover Develop and implement the appropriate activities, prioritized through the risk management process of the organization, to restore the capabilities or critical infrastructure services that were impaired through a cybersecurity event.

Our concept of a filtering security mechanism could be categorized into the *Detect* and *Respond* functions. Thus, this solution alone is not enough to secure a system in

the field. One must look at the whole chain that a system consists of. McGraw supports this theory in his article [30] and writes:

> A central and critical aspect of the computer security problem is a software problem.

Thus he puts emphasis on the impact which software has on the security of a system. Software defects with security consequences, like implementation bugs such as buffer overflows and design flaws, will follow us for years. These kinds of bugs will probably increase as software continues to grow in complexity and extensibility. This is true even for military systems that is expected to have a long lifetime and could have remote solutions. By using SPARK one ensures and proves that the program is free from several of the most common vulnerabilities which include: improper input validation, integer overflow or wraparound, buffer overflow errors, improper initialization, improper validation of array index, and information leakage. If other languages are considered one should make sure to use a static analysis tool to detect implementation flaws.

IT security must be considered early in the lifecycle as the software engineering process might have the most powerful impact on the overall security of the system. Knowing and understanding threats, designing for security, and subjecting all assets to a thorough objective risk analysis and testing is all important tasks that needs to be performed. It goes without saying that it is easier to protect software that is free from flaws than something full of vulnerabilities.

As previously stated, IT security is about designing and building software to be secure, building secure software, and educating developers, architects, and users in how to build and use secure systems. One must also consider the protection needed after the product is developed. This includes countermeasures against malicious code, obfuscating code, locking down executables, monitoring, enforcing the security policies using technology, and dealing with complex systems.

Hardware used must also be thoroughly assessed in order to prove that it works as specified and must go through the same process as software to some extent. It is also important to consider covert channels that could pose a serious threat. A covert channel could be hardware that emits radiation, the radiation could then be translated into binary code and represent actual information.

In order to securely develop software, we recommend the design process by McGraw [30] which is illustrated in Figure 5.1.

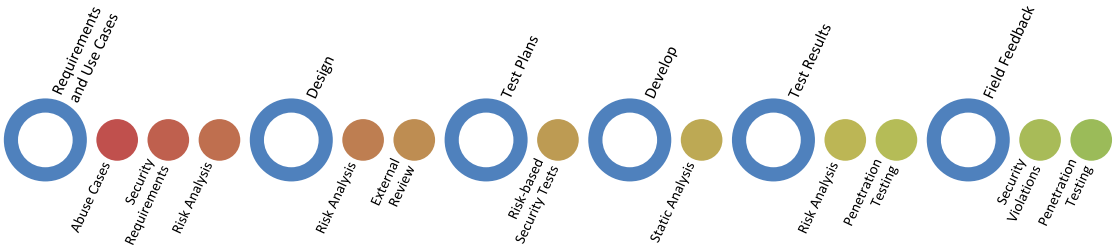


Figure 5.1: Timeline on process steps throughout development of secure IT systems.

Security should be clearly defined at the requirement level. The security requirements should cover functional security (like cryptography and *Message Authentication Code* (MAC)) as well as emergent characteristics. Abuse cases, like use cases, describe the behavior of the system, but under an attack. These should contain the assets to be protected, the current threat agents, and for how long the protection should be operational.

At the design phase the system should be coherent and employ security principles, such as the principle of least privilege. Every member of the personnel involved in this phase should clearly document assumptions made and identified potential attacks. Risks should be acknowledged and ranked such that mitigation of the vulnerabilities can be performed. External review is often necessary and beneficial since undiscovered flaws may cause costly problems later on.

Focusing on the code, one should focus on the implementation flaws. The SPARK toolset includes static analysis, while third-party tools must be used with other languages. Code review is necessary as bugs could be catastrophic from a security standpoint. Taxonomies of common coding errors could be of great help since it is better to avoid the known flaws rather than correcting made mistakes [31].

When it comes to testing one must ensure to perform both testing of security functionality with standard functional testing and risk-based security testing based on current attacks and threats. Penetration testing has the advantage of giving a good understanding for the functionality of a system in a live environment. This kind of testing should be performed with a *white-box* approach as it takes the system architecture into account. Using a *white-box* approach, the tester has access to all information about the system, like code and documentations.

Deployed systems should be carefully monitored when in use such that security violations can be detected. This since, no matter how strong we believe the protection is, attacks may occur. Logging is a way to gather knowledge about attacks and vulnerabilities such that it can be used in development to improve and correct both current and future systems.

The protocols used in military systems, like NMEA 0183 and many other used protocols, cannot by themselves guarantee the security properties. This since for each property it has the following vulnerabilities (among others):

Confidentiality Full disclosures since packets are sent in clear text.

Integrity No way to detect if a message was changed or manipulated.

Availability DoS attacks can easily flood devices to cause unintentional behavior.

Authenticity Does not contain a field to authenticate its sender, such that messages can be sent by a mistrusted source.

Accountability No way to prove that a certain message was received or sent.

A filtering security mechanism can help prevent disclosure (confidentiality) and manipulation (integrity) since unrecognized traffic will be detected and discarded. However, this is only true for traffic that passes through the device. To protect the whole chain one must and should secure the protocols already at low-level.

It is not trivial to recommend how the security properties should be fulfilled since it depends on the assets the system possesses and the risks associated with it. One must also consider the cost that comes with security. The monetary cost, performance, user experience, and security all depend on each other and therefore there is a trade-off between them. Thus, implementing and maintaining countermeasures often comes with a price. It can influence the user experience, have a negative impact on performance, and add direct monetary cost caused by the implementation and maintenance. The protection cost must thus be balanced against the cost of security failure and recovery as well as the potentially intellectual cost. As seen in Figure 5.2 one should always implement countermeasures if the risk level is high and cost of treatment low, while it is not true for the opposite. Of course we have a large gray area where it is not trivial to say if protection should be implemented or not. Here, good judgment and knowledge is needed to make the right decisions.

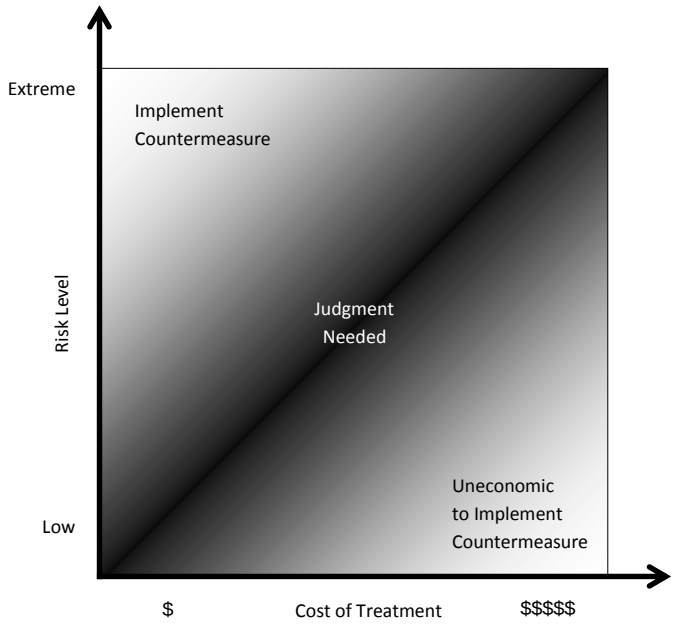


Figure 5.2: Risk level vs. cost of treatment.

The automotive industry has to some extent the same challenges as the military industry [32, 33]. That is:

Hardware Existing and new systems might have strong limitations when it comes to hardware. Both processing memory and processing power might be lacking. This is probably not true for a threat agent, which should be assumed to have the best possible equipment and resources.

Real-time Closely connected to the hardware. Limited hardware performance makes complex instructions take more time. However, some systems must be able to run in real-time to ensure safety for both personnel and civilians.

Autonomy The protection mechanism should not need attention from the operator of the system. It should therefore be as autonomous as possible and only require the attention of the operator in critical situations.

Physical constraints The system must endure harsh conditions, it could be affected by: dust, sand, high or low temperature, moisture, and shocks.

Lifecycle The lifetime of a military system is much longer than for an ordinary computer system. The security mechanisms and the rest of the system need to function all the time without degradation.

Compatibility Security architecture might have the need to be compatible with several different systems without too much modification.

With this in mind we suggest that communication between systems of different classification must consist of the following items:

Cryptography Should be used to encrypt and authenticate packets sent over the bus. Encryption prevents unauthorized disclosure of data. Although encryption might require computation power it could be solved with a separate hardware module that only handles encryption and decryption. Cryptographic techniques deal with the confidentiality, integrity, authenticity, and accountability properties.

Anomaly detection Monitors the traffic to detect anomalous data and take action against it.

Software integrity Used to ensure that the implementation is free from vulnerabilities and is reliable.

Fault tolerance A security service that requires zero downtime must add redundancy. Several hardware components could be added such that one can take over if another goes down. This deal with the availability property.

Education and training Education is according to us the single most important protection against social engineering, both from inside and outside threats. It helps the personnel being aware of the threats and act accordingly. It should lead to a more secure system architecture, development phase, and implementation in the long run.

Authentication and authorization Security of an entire product is often not stronger than its weakest link. A product built by untrustworthy personnel cannot be trusted; it is as simple as that. The principle of least privilege is an important property that states that one (a user, system, or process, etc.) should have no more access than absolutely needed for legitimate use.

We cannot give a universal solution on what to use since each case is unique in regards of assets and risk. However, it can be concluded that the security must be improved. Many of the systems used and developed for the military must be safe, lives can be at stake. A filtering security mechanism is merely a component used to add security, but the protocols need to be designed with security in mind. The general threat level increases and some of the threat agents are nations with tremendous resources.

We are highly convinced that anomaly detection is needed. Possible attacks from governments will probably, as Stuxnet [5], be highly customized and sophisticated using unknown vulnerabilities. Traditional signature-based detection will thus not be enough to hinder these attacks. An anomaly-based *Intrusion Prevention System* (IPS) could be used to enforce the security models used within military systems. For example, an anomalous traffic pattern in a computer network could mean that a compromised computer is sending out sensitive data to an unauthorized destination or that the otherwise normal data has been modified during its transit.

Our proposed solution has the advantage of not needing any information about the used protocol. That means; no details about the specification of allowed messages, cycle times, header sizes, etc., are required. Instead of requiring all this knowledge as well as individual configurations of thresholds for all these specific characteristics, the presented solution has the ability to self-learn the normal behavior during the training phase. Moreover, this approach has very low memory and hardware requirements and can support two network interfaces at the same time (two-way communication). Besides, it allows for easy extension and adaption to new protocols, just by replacing the model with an updated version of the normal behavior based on a new record of system traffic.

6

Conclusion and Future Work

During the thesis we have built an intrusion prevention system based on the latest research and showed that it is applicable in military systems. We have trained the prototype with data based on protocols used within the military for serial communication and evaluated it accordingly. For this section we summarize our findings and propose what should be done for future development.

6.1 Conclusion

We cannot stress enough about the importance of robust end-to-end security from protocol level and up. Our proposed system is not enough to secure a system by itself. This since it cannot alone ensure all the security properties and can only filter data that passes through it. Nevertheless, we believe that the developed concept in this report shows impressing results with regards of performance, detection rate, and low false positive rate. It should be further developed such that it can be included in different systems as a strong countermeasure that can deal with information leakage and data manipulation. The solution is generic such that it can be used in any system using any protocol.

As shown, the solution cannot alone secure a system, in fact not any device can do that. Therefore, we have looked at the whole lifecycle for such a system and proposed how a company should assess threats and protect themselves. This includes proper risk assessment and evaluation of assets as well as building security in from the start, it is nothing you can add later. Every systems need is different; therefore every need for IT security is different and should be custom made accordingly.

We see a rise in threats, both from insiders like employees and from outsiders like nations. This is serious and action must be taken now as it will take years before proper security is in place. Security must be considered throughout the development of a product as security is only as strong as its weakest link. If we cannot trust the people building it, we cannot trust the system.

6.2 Future Work

We see a clear need to thoroughly evaluate any hardware that could potentially be used as well as how to secure it. This, because a security solution used in the field needs to be secure in every aspect, secure software is not enough.

Hardware could be seen as the access point to the software and will be a natural attack vector. High demands should be placed on the hardware, and included firmware, to ensure that it only performs the required operations and nothing more. We are convinced that the hardware should be kept as simple as possible such that it can be tested and verified according to its specifications without an unreasonable long process. The hardware itself needs to be secured from tampering and unauthorized access.

A natural next step would thus be to evaluate existing hardware on the market to see if it could fit the requirements of the system in which the device would be placed in. Also, different countermeasures used to secure the hardware, like security seals, should be considered and systematically evaluated.

Social engineering is according to us the greatest threat against a solution as described in this thesis. One would have to look at how to counteract this because it is often easier to manipulate an involved party, like an employee holding the authentication parameters, than to directly target a system designed to be secure. It has shown to be hard to protect a system against social engineering and we cannot overstate the importance of education and information about the existence of the problem.

Bibliography

[1] R. Shirey, Internet Security Glossary, RFC 2828 (May 2000).
URL `https://www.ietf.org/rfc/rfc2828.txt`

[2] PwC, United States Secret Service, CERT® Division of the Software Engineering Institute at Carnegie Mellon University, CSO magazine, US cybercrime: Rising risks, reduced readiness key findings from the 2014 US state of cybercrime survey.

[3] PwC, Managing cyber risks in an interconnected world key findings from the global state of information security survey 2015.

[4] J. R. Gosler, L. Von Thaer, Task force report: Resilient military systems and the advanced cyber threat, Washington, DC: Department of Defense, Defense Science Board.

[5] T. M. Chen, Stuxnet, the real start of cyber warfare?, Network, IEEE 24 (6) (2010) 2–3.

[6] B. Guttman, E. Roback, An Introduction to Computer Security: The NIST Handbook, DIANE Publishing, 1995.

[7] W. Stallings, L. Brown, Computer Security: Principles and Practice, 2nd Edition, Pearson, 2012.

[8] D. E. Bell, L. J. LaPadula, Secure computer systems: Mathematical foundations, Tech. Rep. ESD-TR-73-276, Deputy for Command and Management Systems (November 1973).

[9] D. E. Bell, L. J. La Padula, Secure computer system: Unified exposition and multics interpretation, Tech. Rep. ESD–TR–75–306), DTIC Document (1976).

[10] K. J. Biba, Integrity considerations for secure computer systems, Tech. rep., DTIC Document (1977).

[11] D. D. Clark, D. R. Wilson, A comparison of commercial and military computer security policies, in: 2012 IEEE Symposium on Security and Privacy, IEEE Computer Society, 1987, pp. 184–184.

[12] D. F. Brewer, M. J. Nash, The chinese wall security policy, in: Security and Privacy, 1989. Proceedings., 1989 IEEE Symposium on, IEEE, 1989, pp. 206–214.

[13] K. A. Scarfone, P. M. Mell, Sp 800-94. guide to intrusion detection and prevention systems (idps), Tech. rep., Gaithersburg, MD, United States (2007).

[14] S. Zanero, S. M. Savaresi, Unsupervised learning techniques for an intrusion detection system, in: Proceedings of the 2004 ACM symposium on Applied computing, ACM, 2004, pp. 412–419.

[15] M.-L. Shyu, S.-C. Chen, K. Sarinnapakorn, L. Chang, A novel anomaly detection scheme based on principal component classifier, Tech. rep., DTIC Document (2003).

[16] K. Wang, S. J. Stolfo, Anomalous payload-based network intrusion detection, in: Recent Advances in Intrusion Detection, Springer, 2004, pp. 203–222.

[17] D. Bolzoni, E. Zambon, S. Etalle, P. Hartel, Poseidon: A 2-tier anomaly-based intrusion detection system, arXiv preprint cs/0511043.

[18] O. Kolesnikov, W. Lee, Advanced polymorphic worms: Evading ids by blending in with normal traffic.

[19] K. Wang, J. J. Parekh, S. J. Stolfo, Anagram: A content anomaly detector resistant to mimicry attack, in: Recent Advances in Intrusion Detection, Springer, 2006, pp. 226–248.

[20] B. H. Bloom, Space/time trade-offs in hash coding with allowable errors, Communications of the ACM 13 (7) (1970) 422–426.

[21] J. Barnes, SPARK: The Proven Approach to High Integrity Software, Altran Praxis, 2012.

[22] AdaCore and Altran UK Ltd, SPARK 2014 Reference Manual, 3rd Edition.

[23] Texas Instruments, Standard interface guide, Online Document.
URL `http://www.ti.com/lit/sg/slyt605/slyt605.pdf`

[24] Texas Instruments, Introduction to the controller area network (can), Online Document.
URL `http://www.ti.com/lit/an/sloa101a/sloa101a.pdf`

[25] International Organization for Standardization, Standards.
URL `http://www.iso.org/iso/home/standards.htm`

[26] L. A. Luft, L. Anderson, F. Cassidy, Nmea 2000 a digital interface for the 21st century, in: Institute of Navigation Technical Meeting, 2002.

[27] R. Kulenov, Random nmea sentences generator @ONLINE (2015).
URL `http://freenmea.net/emitter`

[28] D. Depriest, Nmea data@ONLINE (1994).
URL `http://www.gpsinformation.org/dale/nmea.htm`

[29] National Institute of Standards and Technology (NIST), United States of America, Framework for improving critical infrastructure cybersecurity.

[30] G. McGraw, Software security, Security & Privacy, IEEE 2 (2) (2004) 80–83.

[31] K. Tsipenyuk, B. Chess, G. McGraw, Seven pernicious kingdoms: A taxonomy of software security errors, Security & Privacy, IEEE 3 (6) (2005) 81–84.

[32] I. Studnia, V. Nicomette, E. Alata, Y. Deswarte, M. Kaâniche, Y. Laarouchi, Survey of security threats and protection mechanisms in embedded automotive networks, in: Dependable Systems and Networks Workshop (DSN-W), 2013 43rd Annual IEEE/IFIP Conference on, IEEE, 2013, pp. 1–12.

[33] S. Checkoway, D. McCoy, B. Kantor, D. Anderson, H. Shacham, S. Savage, K. Koscher, A. Czeskis, F. Roesner, T. Kohno, et al., Comprehensive experimental analyses of automotive attack surfaces., in: USENIX Security Symposium, San Francisco, 2011.