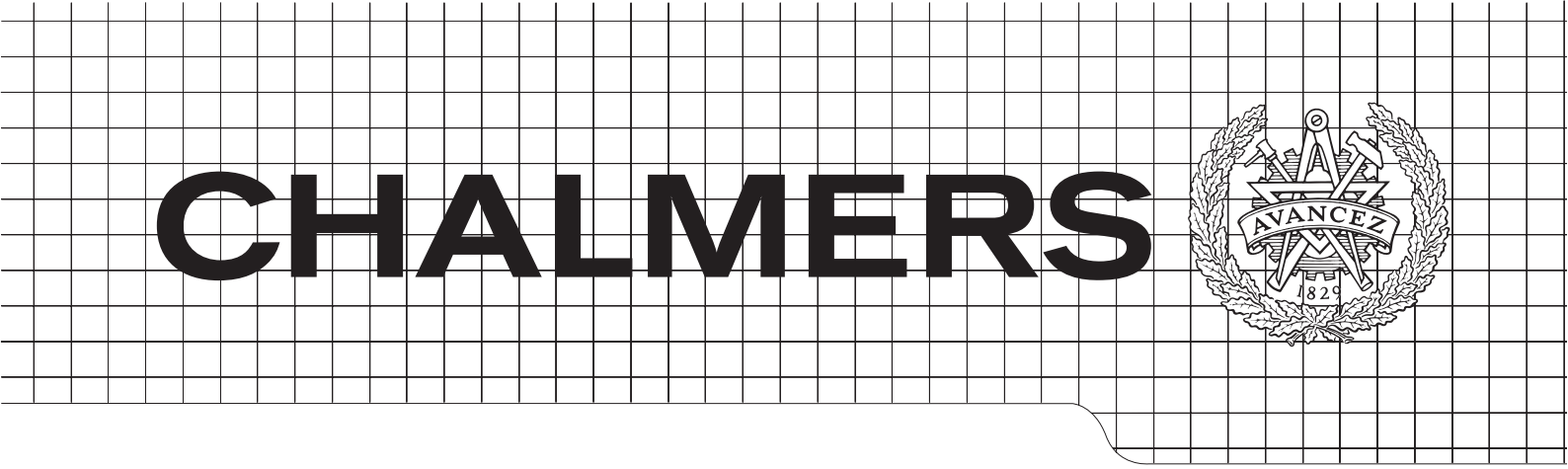

Vehicle Control Unit Security using Open Source AUTOSAR

Master's Thesis in Software Engineering

Anton Bretting & Mei Ha

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

Vehicle Control Unit Security using Open Source AUTOSAR

ANTON BRETTING
MEI HA

Examiner: Matthias Tichy
Supervisor: Riccardo Scandariato

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

Department of Computer Science and Engineering
Göteborg, Sweden June 2015

Abstract

Security threats against software in cars could affect the safety of the vehicle as the numbers of computers and advanced functionality in cars increase. This thesis report presents a case study where two variants of Microsoft's threat modeling technique STRIDE are applied to a limited part of the AUTOSAR platform. The two variants are performed separately and the outcomes are compared to evaluate which STRIDE variant performs better in an automotive environment by analyzing the false positives and true positives found. It was found that STRIDE-per-element was a better STRIDE variant for the purpose of the automotive domain. Moreover, the case study found that the SecOC module mitigates most of the spoofing and tampering threats, and that the biggest group of remaining threats is denial of service or flooding of the CAN network.

Keywords. AUTOSAR, Threat modeling, STRIDE, Security

Acknowledgements

We would like to thank our university supervisor Riccardo Scandariato for the valuable feedback. Thanks Arccore for giving us the opportunity of doing this thesis and special thanks to our supervisor Karl Erlandsson at Arccore, and other people in Arccore for making the thesis possible.

Anton Bretting, Mei Ha, Gothenburg 4/6/15

Contents

List of Figures

List of Tables

Abbreviations

AUTOSAR Automotive Open System Architecture

BSW Basic Software

CAL Crypto Abstraction Layer

CAN Controller Area Network

CANIF CAN Interface

COM Communication

CSM Crypto Service Manager

DFD Data Flow Diagram

E2E End-to-end

ECU Electronic Control Unit

HMAC Hash-based message authentication code

MAC Message Authentication Code

MCU MicroController Unit

NvM Non-volatile Memory

PDU Protocol Data Unit

PduR PDU Router

RTE Runtime Environment

SDL Security Development Lifecycle

SecOC Secure Onboard Communication

STRIDE Spoofing, Tampering, Repudiation, Information disclosure, Denial of service, Elevation of privilege

UML Unified Modeling Language

1

Introduction

Safety has traditionally been regarded as one of the most important attributes in the automotive industry. However, as the number of computers and advanced functionality in cars increase, new challenges arise. Security threats, e.g. hacker attacks against the software in cars could potentially affect the safety of the vehicle [17][6]. According to Xiaoling [39], 60% of the cars worldwide will be connected to the internet by 2017, and more car manufacturers allow third party developers to develop applications that connect to the car systems. This opens up the cars to new threats, and the requirements for new security features increase. The security becomes even more critical in the coming years when self-driving cars will be introduced to public roads [1].

New devices such as the CANtact make it easier and cheaper to connect to the CAN network in the car via the OBD-II port, and from there the functionality of the car can be controlled [8].

The new threats to security and safety have led to an increase in research on software security in cars. Projects such as EVITA and HEAVENS aim to identify vulnerabilities in automotive software and to find ways to minimize or eliminate the vulnerabilities [12][34].

To counter the growing threats against the automotive industry, a well-established security technique from the IT industry will be evaluated. The technique that will be used in this study is STRIDE, which is a threat modeling technique developed by Microsoft. STRIDE stands for Spoofing, Tampering, Repudiation, Information Disclosure, Denial of Service, and Elevation of Privilege and was developed to help identify the different types of attacks against a system [32].

The largest software platform for the automotive industry is Automotive Open System Architecture (AUTOSAR) [2] and the focus of this thesis will therefore be to evaluate the security of AUTOSAR using STRIDE.

AUTOSAR clearly defines layers to be able to fit into most modern Microcontroller Units (MCUs) that are controlling the logic within the vehicle Electronic Control Units

(ECUs). The AUTOSAR stack handles important aspects such as communication, diagnostics and most of the peripherals. The AUTOSAR Run Time Environment (RTE) also allows for application software portability and re-use [2]. In the 4.2.1 release, AUTOSAR introduced a module called Secure Onboard Communication (SecOC), which is supposed to add authentication mechanisms to AUTOSAR to counter the new threats.

The objective of the thesis is to compare two variants of STRIDE, namely STRIDE-per-element and STRIDE-per-interaction to see which one yields the best result. The comparison will be done by applying the STRIDE variants to AUTOSAR.

1.1 Delimitations

This thesis is delimited to only modelling a part of the platform of the AUTOSAR with a reduced level of detail due to limited time. The limited part that will be focused on in this thesis is shown in Figure 2.3. The AUTOSAR platform will be analyzed with two variants of STRIDE; other methods of threat analysis will not be explored.

Implementation is a small part of the thesis, therefore only a subset of the CAL and SecOC modules will be implemented. The focus will be to create a working example to demonstrate.

1.2 Outline of the thesis

This thesis report is structured as follows: First, the background to the thesis will be introduced, whereafter Chapter 3 discusses the related work. After that Chapter 4 will introduce the research approach used in the thesis. Chapter 5 will describe the software implementation. Then Chapter 6 will describe the process of creating the threat model. The case study will be presented in Chapter 7. Chapter 8 will present the result of the study. Lastly, Chapter 9 consists of discussion concerning the result and future work.

2

Background

This chapter gives a background to the thesis. First, AUTOSAR is introduced and an overall description about the AUTOSAR architecture. Second, threat modeling is described followed by STRIDE and last, a detailed description about two variants of STRIDE: STRIDE-per-element and STRIDE-per-interaction.

2.1 AUTOSAR

AUTOSAR is an open software architecture for the automotive industry. It was created 2003 by a consortium of automotive manufacturers and suppliers. The purpose of AUTOSAR is to move the focus from a component-driven development process to a more function-driven development process. The architecture is designed to allow manufacturers to seamlessly move functionality between different ECUs and to reuse functionality, in order to reduce the cost of development and to make it easier to perform software updates over the vehicle lifetime [2]. Without AUTOSAR the software of every ECU has to be tailored for the function that it is supposed to run, which makes it very time consuming to move functions from one ECU to another. This time is significantly reduced with the introduction of AUTOSAR. The applications in AUTOSAR are completely hardware independent [19]. Almost 80% of the car production in the world comes from AUTOSAR partners [2].

2.1.1 Architecture

As shown in Figure 2.1 the AUTOSAR Platform uses a three layered software architecture.

The top layer is the application layer. This layer contains the actual applications that will run on the ECU, for example engine control or cruise control software. The application layer is the only layer where the functionality is not specified by the AUTOSAR standard. The second layer is the Run time Environment (RTE). This is an abstraction

layer between the AUTOSAR operating system in the Basic Software Layer and the applications in the application layer. The last layer is the Basic Software (BSW), which contains the AUTOSAR operating system. This includes diagnostic, memory, communication and system services. The three software layers run on a microcontroller as seen in Figure 2.1. The BSW layer is further divided into four layers [4]. The layering of AUTOSAR and the sub layers of the BSW can be seen in Figure 2.2.

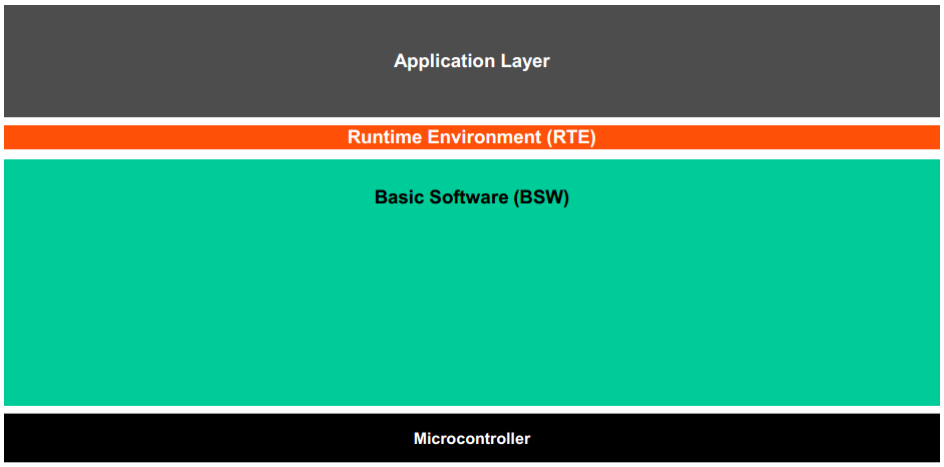

Figure 2.1: Overview of AUTOSAR Software Layers [4].

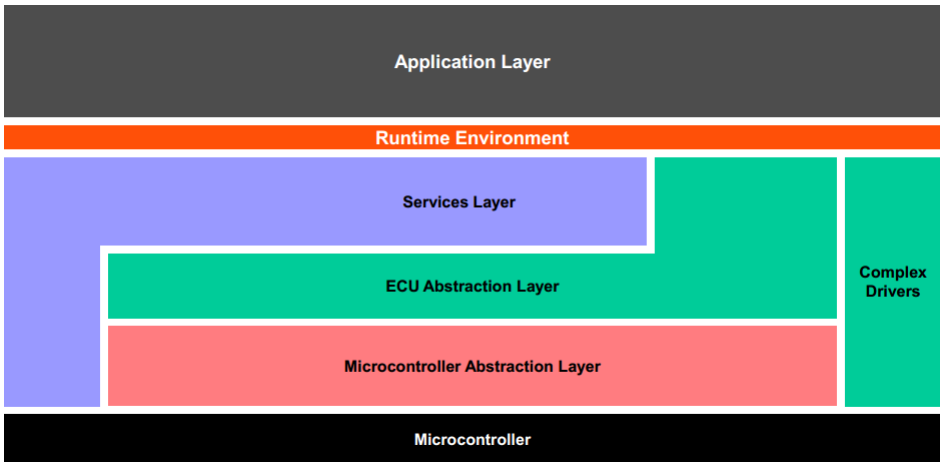

Figure 2.2: Overview of BSW sub layers [4].

Application Layer

The application layer consists of AUTOSAR software components which are running on the ECUs, e.g. the ABS functionality. The AUTOSAR applications are hardware

independent and can be moved between different ECUs without making changes to the software[2].

Runtime Environment (RTE)

The RTE provides the AUTOSAR software components in the application layer to interact with the basic software modules. The RTE is ECU-specific since it depends on the specifications of the ECU to access the correct communication channels[2].

Basic Software (BSW)

The BSW layer provides system services to the application layer so that the applications can perform their functionality. The BSW does not perform any functional jobs itself.

As seen in Figure 2.2, the four sub layers of the basic software are the Microcontroller Abstraction Layer, the ECU Abstraction Layer, the Service Layer, and the Complex drivers.

The Microcontroller Abstraction Layer (MCAL) is a microcontroller specific layer that provides the rest of the BSW with a standardized interface to access the microcontroller [4].

The ECU Abstraction layer provides the BSW with a microcontroller and hardware independent interface to peripherals and devices.

The Service layer is the highest layer in the BSW and provides basic services for applications, RTE and BSW modules. The Service layer also offers [2]:

- Operating system functionality
- Diagnostic protocols and NVRAM management
- Cryptographic services
- Vehicle network communication (e.g. CAN, LIN, Flexray..)

The services provided by the service layer are mostly microcontroller and hardware independent [4].

The last layer is the Complex Driver layer, which spans from the hardware to the RTE. It provides the possibility to integrate drivers for special purpose devices, such as devices with very strict timing constraints or devices and other drivers that are not specified within AUTOSAR [2].

2.1.2 Secure Onboard Communication

In release 4.2, a module called Secure Onboard Communication (SecOC) was specified by AUTOSAR [5]. This module was added to increase the security by adding authentication mechanisms for critical data. The module was designed to be resource efficient and to seamlessly integrate with the current communication systems in AUTOSAR.

As shown in Figure 2.3, SecOC uses either the Crypto Service Manager(CSM) or the Crypto Abstraction Layer(CAL) to provide cryptographic functions. The SecOC module works by using either Message Authentication Codes (MAC) or digital signatures of the messages to ensure that the received data is sent by the right ECU and contains the correct data [5].

Figure 2.3 also shows how the SecOC module is connected to the AUTOSAR communication systems. The SecOC module is located in the services layer. It connects directly to the Protocol Data Unit (PDU) Router (PduR) to handle the security information of the PDUs. When the PduR receives a message that is configured for SecOC, the message is routed to the SecOC module. SecOC then processes the message and hands it back to the PduR for further routing to the final destination [5].

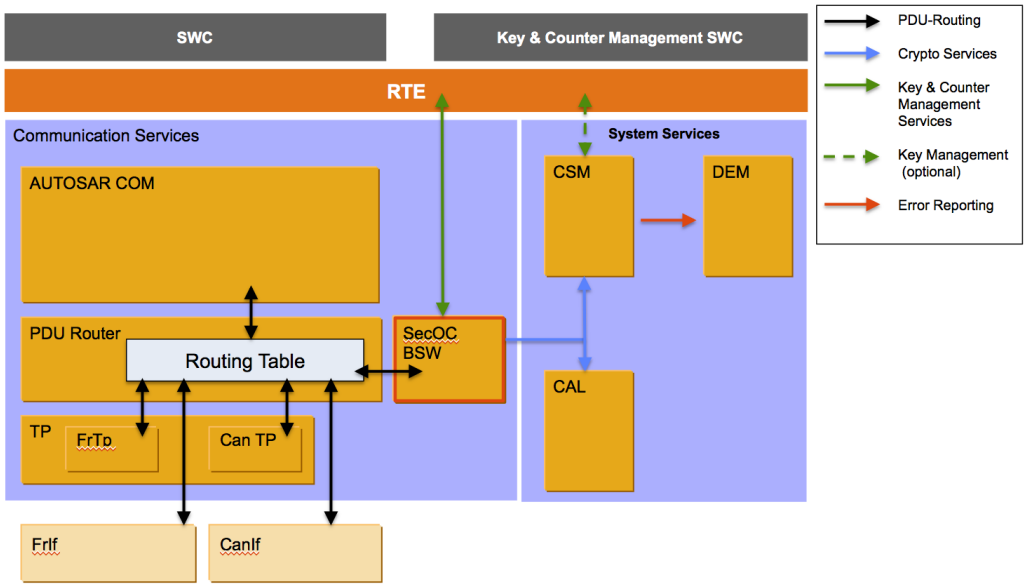

Figure 2.3: Integration of the SecOC module with the rest of the AUTOSAR communication stack [5].

2.2 Threat modeling

As defined in ISO/IEC 27000:2014 [15] and shown in Figure 2.4, a threat is a potential cause of an unwanted incident, which may result in harm to a system or organization.

One method to help with finding and evaluating threats to a system is threat modeling. Threat modeling has many application areas. It can be used as a requirement elicitation technique, to derive security test cases or as a design analysis technique [33][25]. This report will focus on the use of threat modeling as a design analysis technique.

Threat modeling can have different focus and different starting points. The three main approaches according to Shostack [32] are attacker-centric, software-centric and asset-centric.

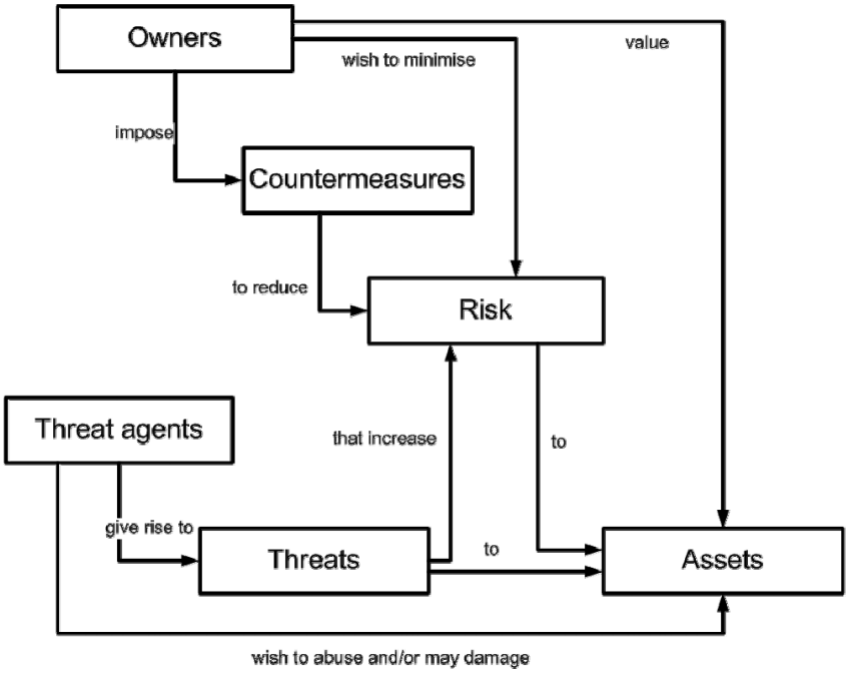

Figure 2.4: Security concepts and relationships [9].

In the Attacker-centric approach the starting point is the attacker; the evaluation starts with the goals, the skills and the motivation of the attacker, and tries to find ways the attacker can succeed with the goals [32]. The Attacker-centric approach often uses techniques such as attack trees or attack nets.

The software-centric approach starts from the design of the system, then steps through the system model and looks for threats against each element of the model. This approach often uses data-flow diagrams and use case diagrams. One example of this approach is STRIDE [32].

The Asset-centric approach starts from the assets of the system, e.g. confidential business information or credit card numbers, and examines how an attacker can threaten them [32]. This is often realized by creating attack trees or attack graphs. The Trike framework has support for the Asset-centric approach in the form of attack trees [30].

A following step to the threat modeling is to evaluate the risk that the threat can cause to the system, the probability of the occurrence of the threat have to be assessed, as well as the severity of the harm the threat can cause to the system [14]. The intentions, means and skill level of the attacker can be used to help with the assessment of the probability of a threat occurring. The attacker, or threat agent, can be defined as any type of individual, group or entity aiming to mount an attack against the system or organization [9].

2.3 Threat modeling with STRIDE

STRIDE is a threat modeling technique developed by Microsoft. It was first proposed in "The Threats to Our Products (1999)" by Kohnfelder and Garg [16], and was later released as a part of the Microsoft Security Development Lifecycle (SDL). STRIDE is named after the six categories that the threats are divided into, namely **S**poofing, **T**ampering, **R**epudiation, **I**nformation Disclosure, **D**enial of Service, and **E**levation of Privilege [13][32]. Table 2.1 contains the definitions of the STRIDE threat categories and what property of the system that the threats violate.

The STRIDE Threat-Modeling Process consists of several high-level steps for creating a threat model. These steps do not require much security expertise. By following the threat-modelling process steps, a systematic review of the system is performed and appropriate mitigations can be determined. The steps are as follows: [13]

1. **Define use scenarios.**

   The team determines the key scenarios which are within the scope of the specific project.

2. **Gather a list of external dependencies.**

   The application might use a database or server, which makes it important to list all other code the application depends on.

3. **Define security assumptions.**

   Making inaccurate security assumptions would incorrectly define the application as insecure, which makes it even more critical to define true security assumptions.

4. **Create external security notes.**

   External security notes would help users or other application designers to understand the security boundaries and how the security of the application could be maintained by using the application.

5. **Create one or more Data Flow Diagrams (DFD) of the application being modeled.**

   DFDs are created for the application, and it is important to create a correct DFD already in the beginning, otherwise the rest of the threat modeling process will be wrong.

6. **Determine threat types.**

   Microsoft uses STRIDE to determine threat types. The different types are described in Table 2.1.

7. **Identify the threats to the system.**

   The DFD elements are then listed, and depending on the STRIDE variant the different STRIDE threat types are applied to the elements.

8. **Determine risk.**

   The risk for each threat found in the previous step is determined by using numeric calculations and then prioritized from low risk to high risk.

9. **Plan mitigations.**

   There are some mitigation strategies that Howard and Lepner [13] provide. The first one is do nothing, which could be applied for low-risk threats. The second one is remove the feature, which would reduce the risk to zero. The third one is about turning off the feature, which should only be used to reduce the risk further. Warn the user is the fourth strategy, since some non-technical users make poor trust decisions. Lastly, countering threats with techniques is the most common strategy. Techniques such as authentication could be used to solve specific issues.

Table 2.1: The STRIDE Threat types [32].

Threat type	**Property violated**	**Threat definition**
Spoofing	Authentication	Pretending to be something or someone other than yourself.
Tampering	Integrity	Modifying something on disk, on a network, or in memory.
Repudiation	Non-Repudiation	Claiming that you didn't do something, or were not responsible. Repudiation can be honest or false, and the key question for system designers is, what evidence do you have?
Information disclosure	Confidentiality	Providing information to someone not authorized to see it.
Denial of Service	Availability	Absorbing resources needed to provide service.
Elevation of Privilege	Authorization	Allowing someone to do something they're not authorized to do.

When modelling the system, modeling smaller functional entities is often more efficient than modeling the entire software. A person of the design group, such as the architect, program manager or analyst usually owns the threat-model process. So, the person with the most security knowledge is the most appropriate for having the role of

building the threat model, while other engineers provide important design information [13].

STRIDE uses data flow diagrams (DFD) to model the system. A DFD contains four types of elements. The elements are External entities, Data flows, Data stores, and Processes. When the DFD is used for threat modeling, there is one more element to keep track of, which is the trust boundary. The trust boundary represents data moving from a high trust to low trust, or vice versa. The different elements are susceptible to different threats, so not all categories of STRIDE can be applied to them [13]. Figure 2.5 shows a simple example of a DFD model of a web server that receives commands from a user, gets data from a database and sends a response to the user. The dotted red line in the figure is the trust boundary, and denotes when the data leaves the high trust of the server to the low trust of the user. The DFD diagrams are then used to either by hand or with the help of a tool evaluate the threats to the system. This part is described further in Chapter 2.3.1 and Chapter 2.3.2.

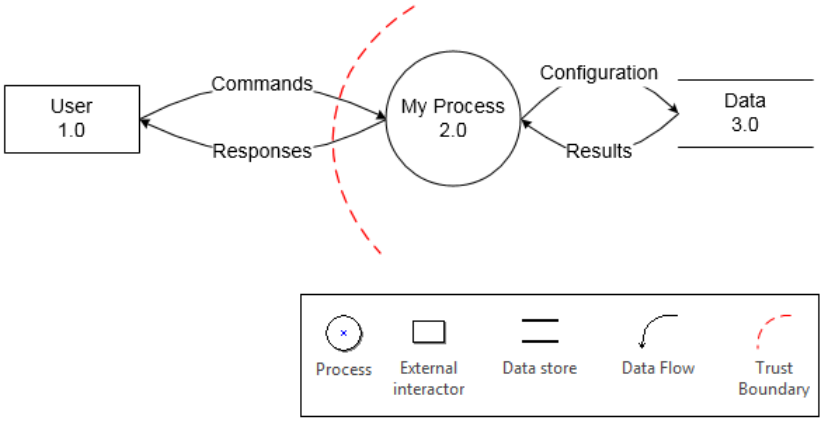

Figure 2.5: Example of an simple DFD model.

Howard and Lepner [13] describe that it is hard to determine whether the threat model is good, because it is rather subjective what is a good model. To counter this, Microsoft started using the metric Threat-Model Quality Guidelines to be able to separate the good from the not-so-good threat models in a more objective way [13]. As seen in Table 2.2, the quality guidelines give a clear way to determine the quality of a threat model. At a minimum all threat models should be rated ”OK” [13].

Shostack [32] describes two ways to judge if the threat modeling process is done. The first and easiest way is to see if a threat of each STRIDE category has been found. The second way is to see if a threat for each element in the DFD has been found. However, having met these criteria are not a guarantee that all threats to the model have been found. So, if these criteria are not met, the threat modeling process is not done [32].

The outcome of the threat modeling process is a set of documents that consists of background information on the application, data flow diagrams, which are used to define

Table 2.2: Threat-Model Quality Guidelines [13].

Rating	Comments
No threat model	No threat model is in place. This is not acceptable because it indicates that no threats are being considered.
Not acceptable	Threat model is out of date if: • Current design is significantly different from model. • Data in model is more than 12months old.
OK	• A data flow diagram or a list of the following exists: – Assets (processes, data stores, data flows, external entities) – Users – Trust boundaries • At least one threat is detailed for each software asset. • Model is current.
Good	• Threat model meets all definitions of ”OK” threat models. • Anonymous, authenticated, local, and remote users are all shown on the DFD. • All S, T, I, and E threats have been identified and classified as either mitigated or accepted.
Excellent	• Threat model meets all definitions of ”Good” threat models. • All STRIDE threats have been identified and have mitigations, external security notes, or dependencies acknowledged. • Mitigations have been identified for each threat. • External security notes include a plan to create customer-facing documents that explain how to use the technology safely and what the trade-offs are.

the high-level application model, but also a list of assets and threats that are ranked by risk. In many cases there is also a list of mitigations to the found threats [13].

2.3.1 STRIDE-per-Element

The first version of STRIDE that Microsoft released was the STRIDE-per-Element variant. In this variant, every element in the DFD is evaluated for threats.

Table 2.3 shows a mapping between the different elements of the DFD and the different categories in STRIDE. It can be seen that not all types of elements are susceptible to every type of threats.

Table 2.3: Mapping of STRIDE to DFD Element types [13].

DFD Element Type	S	T	R	I	D	E
External Entity	X		X			
Data Flow		X		X	X	
Data Store		X	X[1]	X	X	
Process	X	X	X	X	X	X

[1] If the Data Store contains logging or audit data, repudiation is a potential threat, because if the data is manipulated, the attacker could cover his or her tracks [13].

After the DFD model of the system has been created, a list of all the elements in the diagram has to be created. Table 2.4 show the list of elements created from the DFD in Figure 2.5.

Once the list of DFD elements is done, STRIDE will be applied to each element in the list. However, not all types of threats have to be applied to all types of elements. To help with this, Table 2.3 can be used. In Table 2.5, the result of the STRIDE-per-element analysis can be seen. The threats have been grouped after the STRIDE categories. After STRIDE has been applied to the list of elements, it is time to calculate the risk attached to each threat.

The advantage of STRIDE-per-element is that it is prescriptive; it helps to identify what to look for without having a checklist. When STRIDE-per-element is used by an experienced user, it can be useful for finding new types of weaknesses in components, but can also find many common issues even though novices use it [32].

One weakness of STRIDE-per-element is that the same issue shows up in in several places in a model, for example if several elements are a part of the same attack. Another weakness is that STRIDE-to-DFD mapping might be too general and not represent the issues in the specific project [32].

Table 2.4: Elements from DFD in Figure 2.5.

DFD Element Type	DFD Item Numbers
External Entity	User (1.0)
Data Flow	User command & response (1.0 ↔ 2.0)[1]
	Web configuration & results (2.0 ↔ 3.0)[1]
Data Store	Data (3.0)
Process	My Process (2.0)

[1] To reduce the number of entities in the list, the request and response have been combined. This can be done since the Data Flows is between the same elements and cross the same trust boundaries.

Table 2.5: Threats to the model in Figure 2.5.

Threat Type(STRIDE)	DFD Item Numbers
Spoofing	External entities: (1.0)
	Processes: (2.0)
Tampering	Processes: (2.0)
	Data Stores: (3.0)
	Data Flows: (1.0 ↔ 2.0), (2.0 ↔ 3.0)
Repudiation	External entities: (1.0)
	Processes: (2.0)
	Data Stores: (3.0)
Information disclosure	Processes: (2.0)
	Data Stores: (3.0)
	Data Flows: (1.0 ↔ 2.0), (2.0 ↔ 3.0)
DoS	Processes: (2.0)
	Data Stores: (3.0)
	Data Flows: (1.0 ↔ 2.0), (2.0 ↔ 3.0)
EoP	Processes: (2.0)

2.3.2 STRIDE-per-Interaction

The STRIDE-per-interaction approach was developed by Larry Osterman and Douglas MacIver. The meaning of this approach is that threat enumeration consider tuples such as origin, destination, interaction and the threats are enumerated against them. This approach had another goal during its development, that is to reduce the number of things that a modeler would have to consider [32]. However, according to Shostack [32], STRIDE-per-element and STRIDE-per-interaction are expected to lead to the same number of threats, but according to Shostack [32], the threats may be easier to understand with the STRIDE-per-interaction approach.

The STRIDE threats that are applicable to the interaction are also shown in the Table 2.6. In Table 2.7, an example of the threats described in plain text for each threat category is shown.

The difference between the two STRIDE variants are that the STRIDE-per-interaction approach is too complex without a reference chart handy, especially compared to STRIDE-per-element where the chart is easy enough to memorize and the approach, easy for beginners to understand [32].

Table 2.6: STRIDE-per-Interaction table: Threat Applicability [32].

#	ELEMENT	INTERACTION	S	T	R	I	D	E
1	Process	Process has outbound data flow to data store	X			X		
2	Data Flow (commands/responses)	Crosses machine boundary		X			X	X
3	Data Store (database)	Process has outbound data flow to data store		X	X	X	X	
4	External Interactor (browser)	External interactor passes input to process.	X		X	X		

Table 2.7: STRIDE-per-Interaction (Example) [32].

#	ELEMENT	INTERACTION	S	T	R	I	D	E
1	Process	Process has outbound data flow to data store	”Database” is spoofed, and Contoso writes to the wrong place.			P2:Contoso writes information in ”database which should not be in ”database”		
2	Data Flow (commands/ responses)	Crosses machine boundary		Data flow is modified by an attacker.			The contents of the data flow are sniffed on the wire	The data flow is interrupted by an external entity (e.g. messing with TCP sequence numbers.)
3	Data Store (database)	Process has outbound data flow to data store		Database is corrupted.	Contoso claims not to have read from ”database”	Database reveals information.	Database cannot be written to.	
4	External Interactor (browser)	External interactor passes input to process.	Contoso is confused about the identity of the browser		Contoso claims not to have received the data	~~P2: process not authorized to receive the data~~ (We can't stop it)		

3

Related Work

This chapter presents literature related to the thesis. First, other threat modeling techniques are presented. Second, risk assessment techniques are described. Last, security and safety in automotive are presented.

3.1 Other threat modeling techniques

This section describes the different threat modeling techniques such as abuse cases, misuse cases, attack trees and goal-oriented threat modeling. Descriptions of how to use the different threat modeling techniques are also included in this section.

3.1.1 Abuse cases

Abuse cases is a form of use cases that are specialized in capturing and analyzing security requirements for a system [23]. It was designed by John McDermott and Chris Fox in 1999. The technique is focused on how the system can be abused by malicious users. As stated by McDermott and Fox [23], abuse cases can be useful in requirements elicitation, design, and testing of systems. It can increase the awareness and understanding of the different security threats to a system since they can be made simple and abstract enough for a wide range of users and customers.

An abuse case is a specification of the interactions between a system and one or more actor where the end result is harmful to the system or one of the actors. They can be described by the same methods as use cases, use case diagrams and use case descriptions. Abuse cases use the same symbols as use cases, therefore, normal Unified Modeling Language (UML) tools can be used to create the diagrams [23].

3.1.2 Misuse cases

Misuse cases is a description of the malicious behavior of an unwanted user by using use cases which are helpful to describe the functional requirements of a system [10]. According to Opdahl and Sindre [26], there are a few comparative evaluations of misuse cases. One of the evaluations is about comparing misuse cases with two other threat modeling techniques for the same realistic example. Opdahl and Sindre [26] also state that the misuse cases technique was easy to learn and use, but the produced output could be hard to analyze afterwards.

3.1.3 Attack trees

Attack trees represent the security of the system being modeled, by using a tree structure where the goal is the root node where the leaf nodes represent different ways of achieving that root node goal. According to Schneier [31], the possible attack goals should be identified in order to create the attack tree, where each goal forms a separate tree even though they might share nodes and subtrees [31].

In sum, an attack tree is useful for finding threats if the attack tree is relevant to the system. Once the system is modeled with a DFD or other diagram, an attack tree is used to analyze it. Eliciting the attacks will require iteration over each node in the attack tree and consider whether the issue impacts the system [32].

3.1.4 Goal-oriented threat modeling

Goal-oriented threat modeling is a framework where the treat models are obstacle models, the reason being that the threats are obstacles to security goals. The attackers have their own anti-goals that need to be satisfied by the intentional threat obstacles [36]. According to Van Lamsweerde et al. [36], an anti-goal is what an attacker may want to achieve. The goal-oriented threat analysis works as follows: firstly, the initial goals, in this case anti-goals and classes of attackers should be identified. For each initial anti-goal and attacker class, an anti-goal refinement/abstraction graph is built to show how the anti-goals can be satisfied in view of the attacker's knowledge and capabilities. When the leaf conditions that meet the attacker's capabilities are reached, the refinement terminates. Secondly, new security goals are derived as countermeasures to counter the leaf anti-goals from the threat graphs [36].

3.2 Risk assessment

Risk assessment is used when threats are identified and need to be prioritized. One way to prioritize threats is to use two factors: damage and likelihood. The overall risk factor for each threat needs to be calculated, then the threat list will be sorted by decreasing order of risk. The prioritized threats can then be addressed starting at the top of the list [25].

3.2.1 Trike

Saitta et al. [30] describe that Trike is a framework for security auditing through generations of threat models, which is also associated with a tool that is still under development. Threat modeling approached from the Trike perspective is also different from other threat modeling techniques since it focuses on a defensive perspective instead of an attackers perspective [30].

When generating a Trike threat model, four things need to be considered:

1. Ensure that the system entails to each asset is acceptable to all stakeholders, with help from the system stakeholders.
2. Be able to tell whether this has been done
3. Communicate what has been done and its effect to the stakeholders.
4. Reduce the risks to all the stakeholders implied by their actions within their domains by making the stakeholders understand those risks.

According to Saitta et al. [30], much of the work in threat modeling should be automated and the trike methodology is designed to support automation, which allows quicker results from less initial information and more complete results with the same amount of effort, compared to other methodologies.

Trike is a particularly good communication device since all the threats to the system and the associated risks are put up in a clear and easy way for the stakeholders to understand without a security background.

3.2.2 CORAS

According to Lund et al. [21], CORAS is a security risk analysis method that consists of three artifacts which are a language, a tool and a method. Compared to other threat modeling methods that use different tree based notation for the analysis, CORAS is according to Lund et al. [21], more general than the tree based notation. The difference is that tree-based notations focus on more specific and limited parts of the analysis while the CORAS language is designed to support all phases of analysis process and is integrated in the risk analysis method [21]. All kind of the tree-based notifications can be simulated with the CORAS language since CORAS is flexible with respect to the level of abstraction in the analysis and is used for risk modelling, where simple graphic symbols and relations between these are used. Since CORAS is a diagrammatic language, the graphic relations make it easier to read the diagrams. The CORAS tool is a graphical editor that supports the CORAS language [21]. It can be used for making any kind of CORAS diagrams.

3.3 Security & safety in automotive

According to Lemke et al. [20], a requirement such as IT security is necessary for future automotive applications but also for the cars that exist today which use IT technology. The IT technology is not the only thing that requires security, the electronic devices have also been feasible targets for attacks or manipulation. Lemke et al. [20] also described that it is hard to implement security in the vehicular area because of insufficient cryptographic knowledge, a multitude of involved parties but also that communication between controllers in the automotive is unencrypted , something that increases the risk of serious attacks.

According to Koscher et al. [17], the automotive industry has always considered safety as a critical engineering concern and showed that with the access to the OBD-II port and the CAN network, all the ECUs of the car could be accessed and the behaviour could be manipulated, e.g. the engine could be stopped and the doors locked. As more sophisticated services and communications features are integrated into the vehicle, the attack surface of the automobiles will increase [17].

One of the challenges concerning security in the automotive is that the standard access controls are weak, Koscher et al. [17] discovered that the controls that existed were not used frequently. For example, a firmware could be loaded onto some key ECUs without any authentication, which makes it easier for attackers to attack the automobile.

4

Research Approach

The purpose of the research, the research questions, the methodology, and finally the validity threats are described in this chapter.

4.1 Research Purpose

The purpose of this study is to compare two variants of STRIDE by applying them to the AUTOSAR platform. As a prerequisite to this, modeling of the AUTOSAR platform is done and also implementation of the SecOC module. The implementation is done to add support for basic security in AUTOSAR.

The new AUTOSAR security modules are evaluated via threat analysis, which is performed with two variants. The threat analysis is done to examine whether the security of the Arccore AUTOSAR platform increases by implementation of the modules. The results of the two variants of STRIDE are evaluated to see whether the SecOC module follows its specification claim.

The new security modules are largely untested and little to no research has been done in the area. That is the reason why this area needs to be evaluated in order to see if threats can be reduced enough or if there is need for more security features.

4.2 Research Questions

To reach the goals of the thesis, the following questions are to be answered:

- **RQ1: Which variant of STRIDE (STRIDE-per-Element or STRIDE-per-Interaction) yields better results with regard to threat modeling?** STRIDE has a known problem called threat explosion, meaning that it can find too many threats to a system and it can be hard to go through the results and find the real threats among all the threats discovered. This study will explore the

differences between the two variants and analyze which one yields the best results in the automotive domain. In this case, best results such as fewer false positives, less time and easier to apply the STRIDE variant. True and false positives refer to the number of identified threats that were correct and incorrect, respectively [27].

- **RQ2: What security threats does SecOC mitigate?** As the amount of advanced functionality in cars increases, the security of the cars needs to be increased as well. This study examines what security threats the AUTOSAR module SecOC mitigates. Does SecOC mitigate the threats it was designed to mitigate, or are more security measures needed?

4.3 Research Methodology

This thesis reports on a 6 month (January 2015 - June 2015) case study. The thesis was conducted as a single case study at a company called Arccore. The key characteristics of a case study is that its conclusions are based on evidence and adds existing knowledge based on previously established theory or by building theory [29]. The study focuses on finding out which variant of STRIDE performs better in an automotive environment, as well as finding what threats the addition of SecOC mitigates. Each person will perform the same task, but with a different variant of STRIDE.

According to Runeson and Höst [29], different research methodologies serve different purposes. Robson [28] describes four different types of purposes for research, Exploratory, Descriptive, Explanatory and Improving. This study will be descriptive and explanatory, meaning that it will describe the situation, as well as seek an explanation of the situation [29].

To compare the two variants of STRIDE, several measurements will be used: the total number of threats found, the number of false positives found, the time spent performing the STRIDE analysis, and the similarities and differences in the true positives found. These measurements are used to answer RQ1.

To answer RQ2, the lists of the threats found will be used. The lists will be used both to answer what threats remain, as well as to see what types of threats that are mitigated by the addition of SecOC.

4.4 Validity Threats

This study has adopted four aspects of validity, which are external validity, internal validity, construct validity and reliability in order to denote the trustworthiness of the results [29]. This section discusses these four aspects and to what extent the results are true.

4.4.1 External validity

The external validity aspect is, according to Runeson and Höst [29], about to what extent it is possible to generalize the findings and how the findings are of interest to

other people outside the investigated case. The outcome of the two treatments are not generalizable since it is specific to the AUTOSAR platform, which is a threat to external validity.

4.4.2 Internal validity

Internal validity refers to the concern when casual relations are examined. For instance, when the researcher is investigating whether one factor affects an investigated factor then there is a risk that the investigated factor is also affected by factor B. There is a threat to the internal validity if the researcher does not know to what extent the investigated factor affects [29]. There is an internal validity threat for the productivity due to the use of tool in one of the two treatments, which was STRIDE-per-interaction. Since both of the treatments were done in the same area and the experience of applying the treatments were shared with the same supervisior. So, the practitioners could have influenced each other, which is a threat to the internal validity.

4.4.3 Construct validity

Construct validity reflect to what extend the operational measures that are studied represent what the researcher has in mind and what is investigated according to the research questions [29]. The tool that was used for STRIDE-per-element was not working as it should so the threat analysis was executed in an another way of than the researcher was prepared for in the beginning, which is a threat to the construct validity.

4.4.4 Reliability

The reliability concerns how the data and the analysis depends on the researchers that performed the experiment [29]. Another researcher should be able to perform the same experiment and get the same result.

The single researcher bias threat has been reduced since this case study has been done by two researchers. In addition, all research findings and each step has been peer reviewed by an external researcher (domain expert in Arccore).

The modeling of AUTOSAR was done by the researcher that did not have any expertise within the domain area, but the model has been reviewed by both a domain expert as well as a university supervisor to reduce the threat.

The STRIDE-per-element was done mostly with brainstorming, this is a concern when it comes to repeatability since the result of the STRIDE analysis is dependent on the person who performs it. To reduce this threat discussions with a domain expert was done to increase the understanding of the likely threats to the platform.

STRIDE-per-interaction was compared to STRIDE-per-element done in a tool, which reduced the threat for reliability since the tool always gave the same outcome if the same model from AUTOSAR was used.

5

Implementation

This chapter concerns the implementation of Crypto Abstraction Layer (CAL) and Secure Onboard Communication (SecOC) modules. There is a section on the testing that was done to ensure the quality of the implementation.

5.1 Overview

The focus of the implementation was to get runnable and testable code early. With that in mind, the first thing that was implemented was the CAL and the Cryptographic Primitive Library (CPL). This module is testable on its own and does not have connections to any other module. When the implementation of the CAL module was done, the focus switched to the SecOC module. This module is much larger and is integrated with several other modules, including CAL. An overview of the system, and how SecOC is connected to the other AUTOSAR modules can be seen in Figure 2.3.

5.2 Crypto Abstraction Layer (CAL)

After discussion with the supervisor at Arccore, it was decided to implement the CAL module instead of the CSM module and to limit the implementation to only a subset of the supported encryption algorithms. The algorithm selected for implementation was the Message Authentication Code (MAC). This algorithm was selected because it was the algorithm primarily used by the SecOC module [5]. The MAC is explained in Section 5.2.1.

CAL have interfaces for cryptographic library modules, called Cryptographic Primitive Library (CPL), which contain the cryptographic functionality [3]. CAL only provides a standardized interface to CPL encryption functions for other AUTOSAR software to use. This layered structure allows for changing of the underlying implementation of CPL

without having to change the overlying code. This is useful, since many manufacturers have their own cryptographic libraries, and this design allows them to easily switch libraries.

5.2.1 Message Authentication Code (MAC)

A Message Authentication Code (MAC) is a piece of information that is used to ensure the integrity and authenticity of a message. To perform this the MAC often uses a cryptographic hash function or a block cipher. It is also possible to combine two or more cryptographic methods when creating the MAC, this allows the MAC to be secure even if one of the cryptographic methods is found to be vulnerable. The version of MAC that was selected for this project is called Hash-based message authentication code (HMAC). This is described in more detail in the following section.

The way MAC works is by creating a MAC tag for the message, which is sent together with the message to the receiver. The receiver then creates its own MAC tag for the message. The two tags are then compared to verify that the message has not been tampered with. Figure 5.1 shows how the MAC can be used to verify the authenticity and integrity of a message.

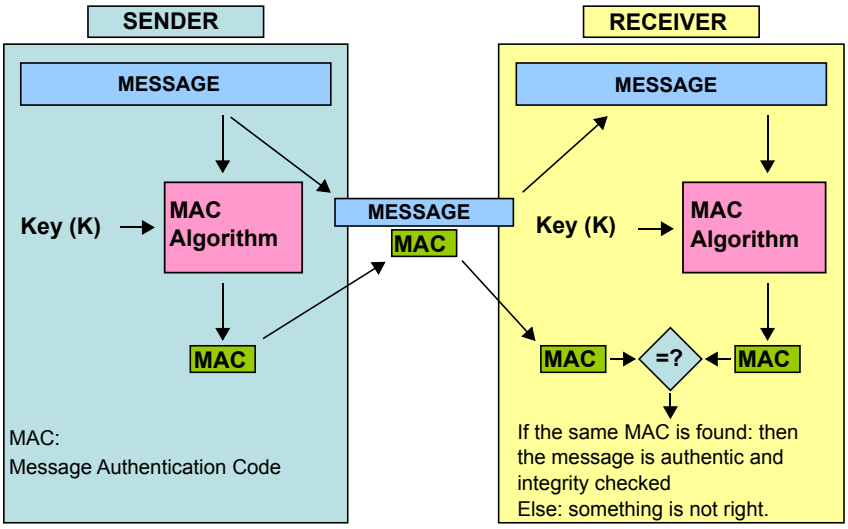

Figure 5.1: Example of how the MAC can be used.

Hash-based message authentication code (HMAC)

The HMAC is a version of MAC that uses a cryptographic hash function together with a cryptographic key to ensure the integrity and authenticity of a message. Any cryptographic hash function can be used, for example MD5 or SHA-1 [18]. The design goals of HMAC has to be able to use existing hash functions without any modifications, to preserve the performance of the function and to allow easy replacement of the function.

The quality of the HMAC output depends on the selected hash function. For this project the selected function is SHA2. It was selected because of relatively low memory usage and because it is a fast algorithm [7]. SHA-1 and MD5 was rejected because of known security flaws [37][35][38]. The version of SHA2 that was selected was SHA256, because it is secure enough while still keeping the MAC value short and the calculation of SHA256 is faster than the SHA512 version.

5.3 Secure Onboard Communication (SecOC)

The SecOC module as described in Section 2.1.2 provides necessary functionality for secure communication between ECUs within the vehicle architecture [5]. One of the objectives of the SecOC module is practicable authentication mechanisms for critical data on the level of the PDUs. According to the SecOC specification [5], symmetric and asymmetric methods for authenticity and integrity protection are supported.

SecOC uses either CSM (Crypto Service Manager) or CAL (Crypto Abstraction Layer) to provide cryptographic functions. The SecOC module works by using either Message Authentication Codes (MAC) or digital signatures of the messages to ensure that the received data contains the correct data [5]. To verify that the message is sent by the right ECU SecOC uses a freshness value. This can be either a counter or a timestamp. When a ECU sends or receives a packet the freshness value is updated.

SecOC defines two types of packets that are used throughout the module. They are called Authentic PDU and Secured PDU. The first one, the authentic PDU, is a message that requires protection from SecOC. The second one, the secured PDU, is a message that has a freshness value and a MAC value attached. This structure can be seen in Figure 5.2.

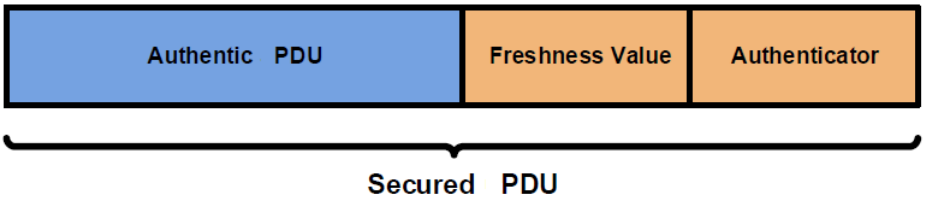

Figure 5.2: The structure of a secured PDU [5].

When SecOC receives an authentic PDU, it attaches the freshness value and then sends the PDU to the CAL module to calculate the MAC value. When that is done, the MAC value is attached to the message. It is now called a secured PDU, that is ready to be sent over the CAN network. When the receiving ECU gets the secured PDU, it is stripped into the original PDU, the freshness value and the MAC value. A new MAC value is then calculated and compared with the attached MAC value. If the two MAC values match and the freshness value is correct, the packet is verified, and the PDU is sent to the application. This flow can be seen in Figure 5.3.

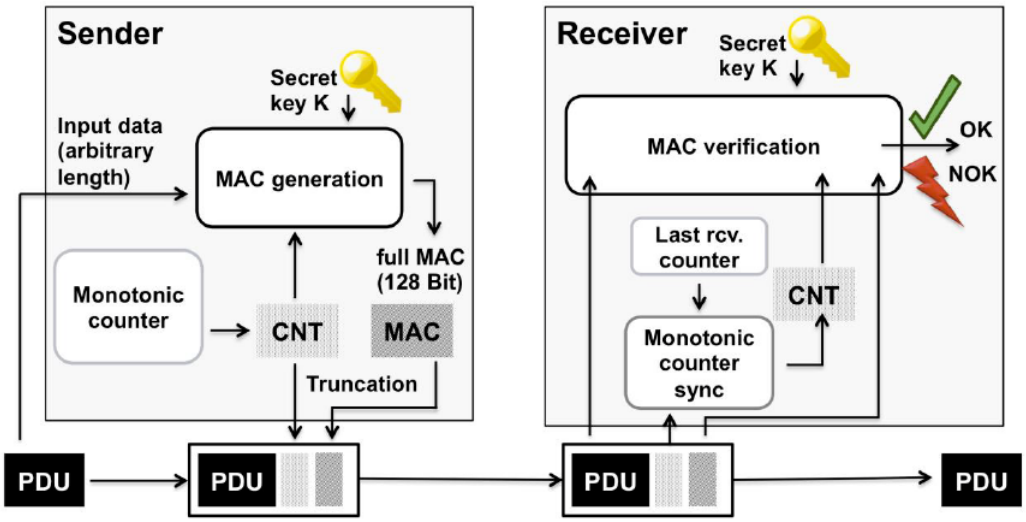

Figure 5.3: Message Authentication and freshness verification [5].

The implementation of SecOC was limited to a minimum working set, with only the basic functionality implemented, as a proof of concept. The implemented functionality is to send and receive messages but without support for transport protocol support and freshness value timestamps. This reduces the protection of the SecOC module to only verifying that the message is correct but not that it arrives in the correct order. This reduction in scope was done to be able to have a functional prototype working at the end of the thesis.

5.4 Testing

Testing is a large part of the implementation since it is necessary to show that the code actually works, for example to get the correct output. It is also a way of showing the quality of the code. The testing tool that is used for writing test cases is EmbUnit which is described in this section. In addition to the unit testing static code analysis was used to increase the quality of the code.

The Cal and SecOC test cases are implemented according to the requirements specified in AUTOSAR [2] and [5], where all of the possible outputs were tested to check if the functions passed or failed in a correct way.

5.4.1 EmbUnit Tests

EmbUnit Testing is an unit testing tool for embedded software for testers and developers that develop the software in C or C++ [11], which is used to write test cases for the implementation of CAL and SecOC modules.

The difference between EmbUnit and other unit testing tools is that the unit tests

in EmbUnit use a simple set of language constructs; it is also possible to generate code automatically from the test cases. In this case, all the test cases are written by hand depending on the features that need to be tested in the code. Standard libraries are not used in EmbUnit, which makes it optimal for small embedded systems since the small embedded systems usually have tight constrains on code and memory [11].

5.4.2 Static code analysis

To further increase the quality of the code, static code analysis in the form of PC LINT was used to analyse the code to verify if it followed the guidelines set by MISRA (Motor Industry Software Reliability Association) [24]. The aim is to increase the portability, safety and reliability of code in the embedded environment. The analysis focused on the code coverage on the unit tests, the complexity of the functions, as well as how the code was written. This was in order to remove bad coding practices and insecure code.

6

Creation of the DFD model

This chapter describes the process that is used to create the DFD. This part is done together and used as a starting point for both variants of STRIDE to avoid bias.

6.1 Threat modeling process

The threat modeling process that is used in this thesis is described in Section 2.3, which is a step by step guide to the STRIDE process. In this chapter step one to five will be covered.

Firstly, the key threat scenarios were defined. One example of a scenario is when the user tries to send unsafe packages through the CAN module by faking the authorization.

A list of external dependencies was then listed. The externals for this model is the applications that run on top of the communication stack and other devices connected to the CAN network that communicates with the ECU.

In this case, an initial model of a limited part of the AUTOSAR was created, this model is described later in section 6.6. Afterwards, the DFDs for the scenarios were created that are used for the STRIDE-variants and the threats are identified based on the variants.

To help with the creation of the model and to keep the model on the relevant parts of the platform, discussions with a domain expert were held. After the model was done, it was validated with the domain expert as well, to make sure that it was correct.

6.2 Define use Scenarios

The first step of the threat modeling process is to define which key scenarios are within the scope. There are two main scenarios, firstly, a packet arriving at the ECU and is successfully verified and sent to the application. Secondly, a packet arriving and followed by a failed verification and dropping of the package.

6.2.1 Verification successful

The first scenario is when a packet arrives from another ECU and is sent to SecOC for authentication and passed on to the Application. This can be seen in Figure 6.1. The packet arrives at the CAN driver and is stored in a buffer. The Can interface module then reads the packet and converts it from a CAN packet to a PDU packet, which can be used by AUTOSAR. CanIF sends the packet through the PDU router, which transfers the packet to SecOC for verification. The verification in turn is performed by the SecOC module, which returns the result of the verification to SecOC, which then sends the PDU back to the PDU router for delivery to the COM module. In the COM module the PDU packet is converted into signals which are then transmitted through the RTE and to the destination application.

6.2.2 Verification failed

In the scenario where the verification of the incoming packet fails, the route of the packet will start out the same as in the successful scenario. The difference occurs when the packet reaches the CAL module. CAL will see that the packet is invalid and sends a negative response to SecOC. SecOC will then proceed to drop the packet. This can be seen in Figure 6.2.

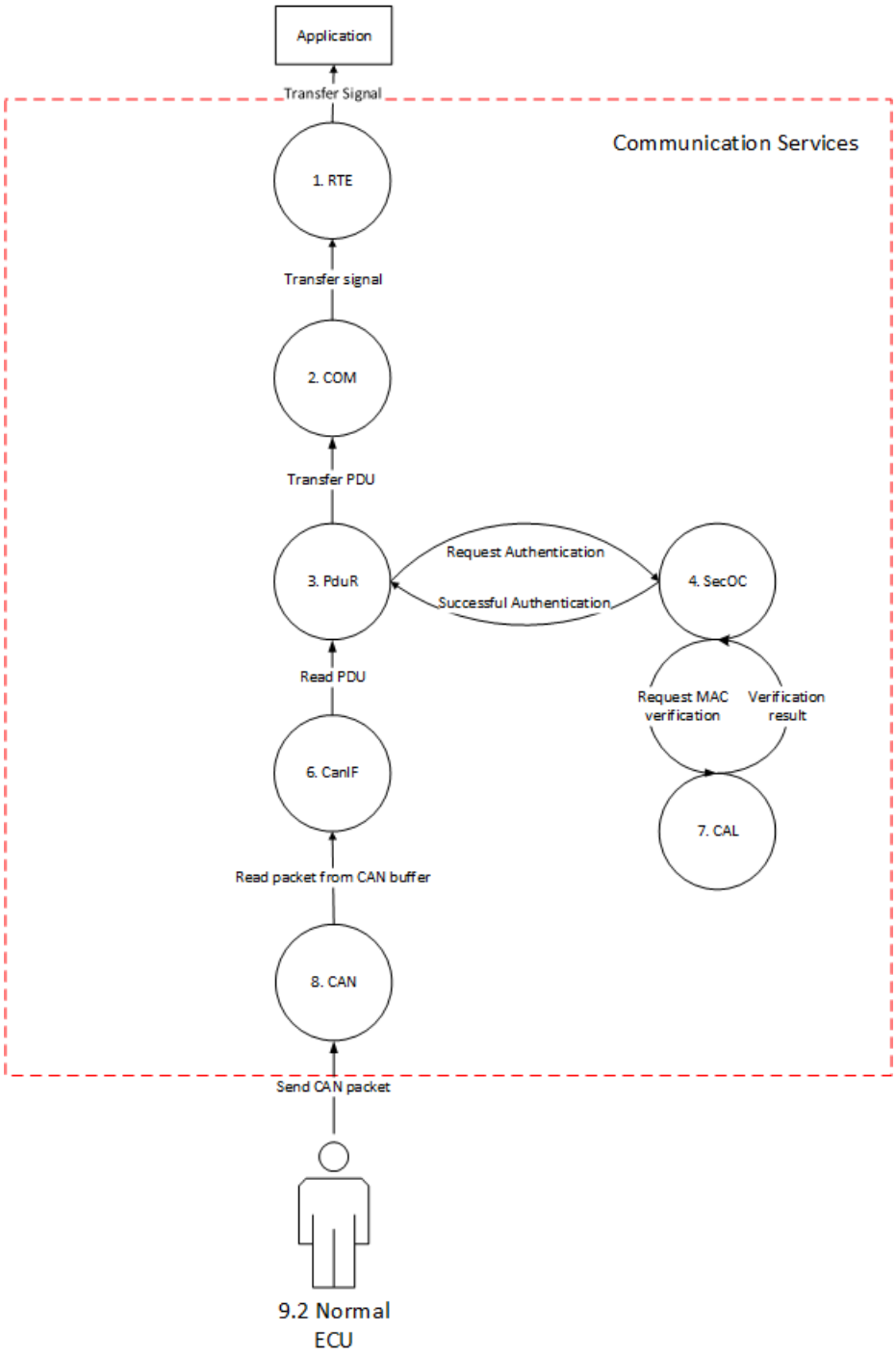

Figure 6.1: Scenario when a packet is authenticated.

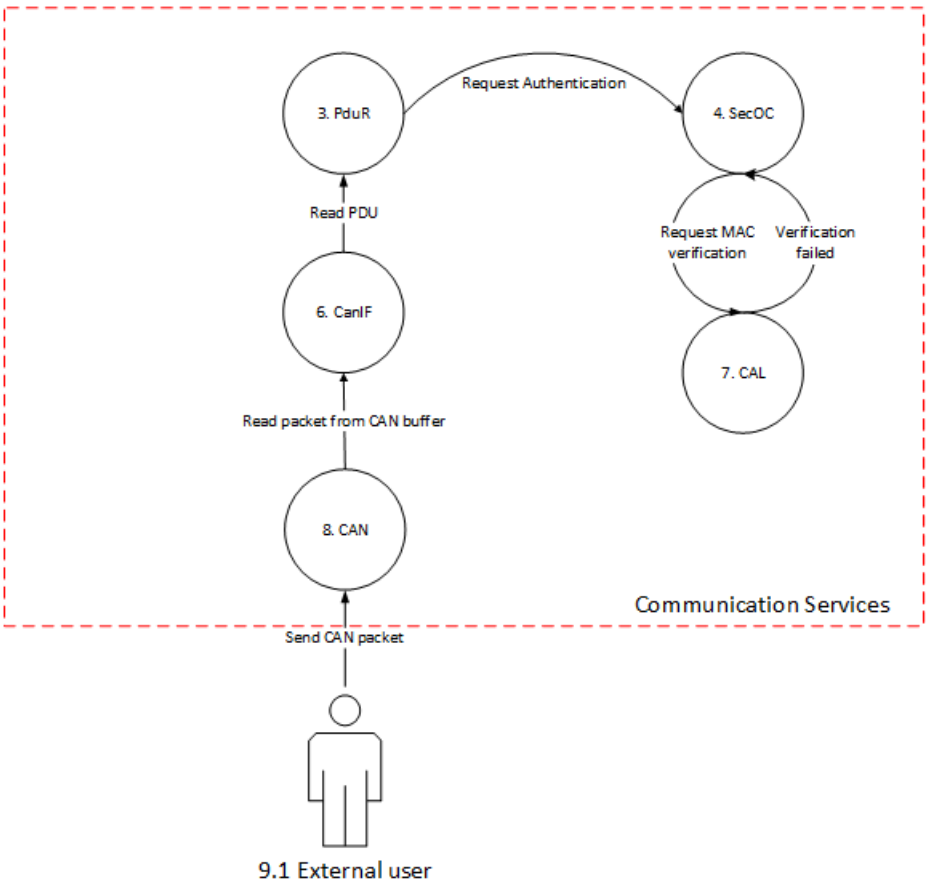

Figure 6.2: Scenario when a packet fails to be authenticated.

6.3 Gather a list of external dependencies

After defining the key scenarios, a list of external dependencies was gathered. As described in section 1.1, only a limited part of the AUTOSAR platform was used for threat modeling and the external dependencies for those are:

- Data stores, in form of buffers for each element.
- An external, user which could affect the AUTOSAR platform through different actions.
- An external ECU, which sends packages to the system.
- Application that communicates with the AUTOSAR platform by End-to-End (E2E) modules.

6.4 Define security assumptions

Defining security assumptions is critically important because inaccurate security assumptions will make the result of the STRIDE analysis meaningless [13]. The first security assumption is that the module SecOC that is implemented, ensures that the communication between the ECUs is secure by providing an authentication mechanism. The second assumption is that the implementation of the authentication mechanism is correct, and that there are no flaws in the authentication mechanism. The third assumption is that all ECUs have SecOC enabled and are using it for all communication.

6.5 Create external security notes

External ECUs that communicate with the AUTOSAR platform are secured by an authentication mechanism provided by SecOC since all ECUs will have SecOC implemented.

6.6 Create one or more DFD of the platform being modeled

This section describes the creation of the initial DFD model that is used for both STRIDE-per-element and STRIDE-per-interaction.

As a starting point for the modeling work, the AUTOSAR architecture was used, and especially the figure that shows the integration of SecOC into the other parts of AUTOSAR. The integration can be seen in Figure 2.3. This was decided after discussion with the supervisor at Arccore. To get the details of each module, the specifications of the modules were studied and experts of each module were asked.

6.6.1 Overview

Figure 6.3 shows the initial threat model for the communication subsystem that was chosen for this study. The focus of the model is the modules located in the Basic Software (BSW). The external entities in the model are: the Application, that connects to the RTE module, the Non-volatile Memory (NvM) that connects to the SecOC module and lastly another ECU that is connected via the CAN network to the CAN module.

The subcomponents of the model are described in more detail in the coming sections.

6.6.2 Application

The Application can be divided into two different classes, one that uses the AUTOSAR E2E protection, and one that does not. The E2E protection adds a Cyclic Redundancy Check (CRC) value and a counter to the message sent by the application, to increase the fault tolerance in the transmission. Even though the E2E protection is designed to increase the fault tolerance, it also adds a layer of security, since it makes modification

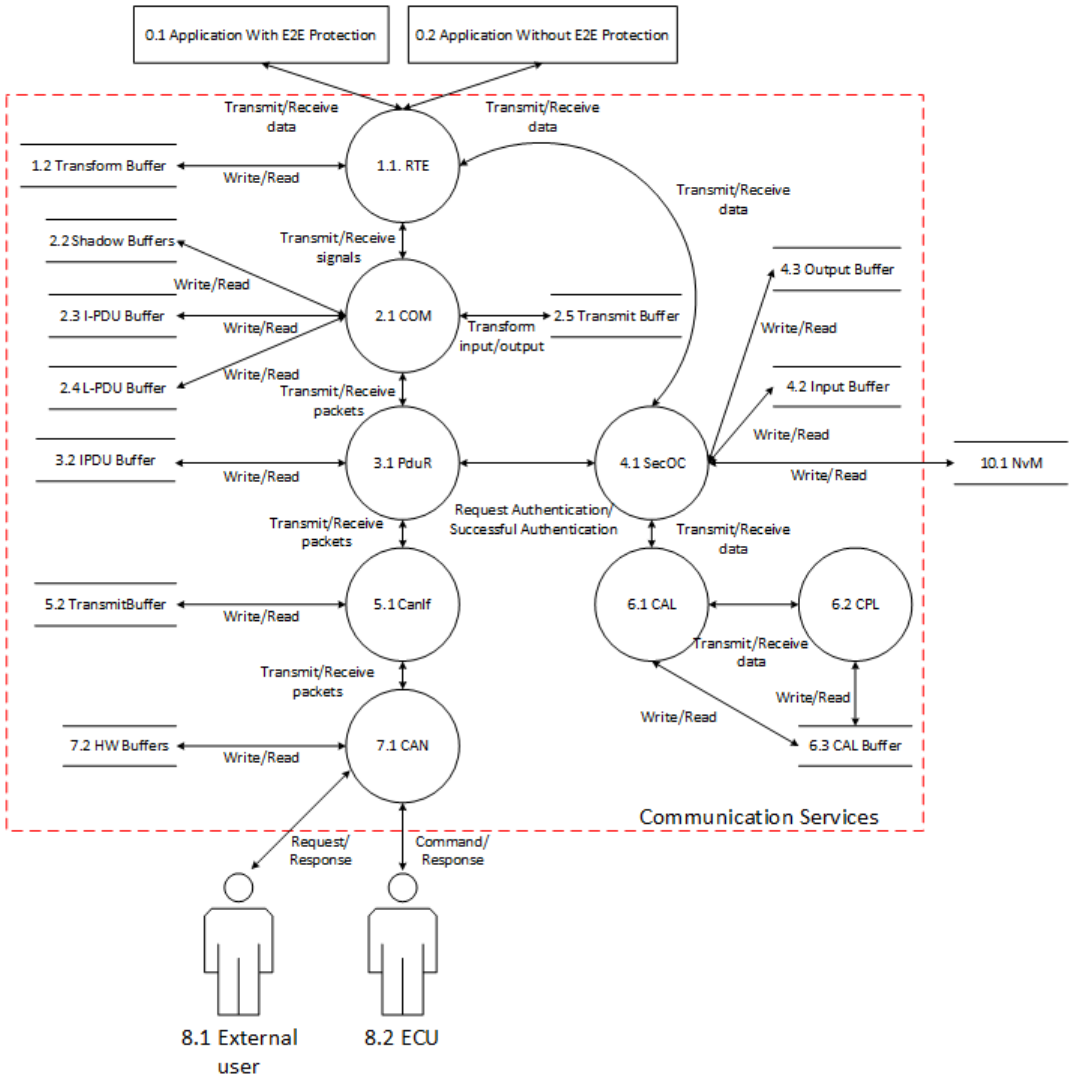

Figure 6.3: Overview of the system.

of the information sent harder. Any modification of the data must generate the same CRC value as the original message.

6.6.3 External user

There are two different categories of users that can communicate with the system over the CAN network. The first category is a normal ECU that sends packets to the system. An example of this could be the engine control unit or the brake system. The other category is an external user. This is something that is connected to the CAN network by someone else than the manufacturer. One example could be that the owner of the car wishes to increase the power of the engine and installs a new ECU to manipulate the engine control unit in order to increase the power of the engine.

6.6.4 RTE

Figure 6.4 shows the RTE part of the model. The RTE does not have any buffers except for the case when a transform function should be applied to the data sent from the RTE. For all other cases, the RTE relies on the modules of the layers above or below, in this case the Application and the COM module, to provide buffers for the data transfer.

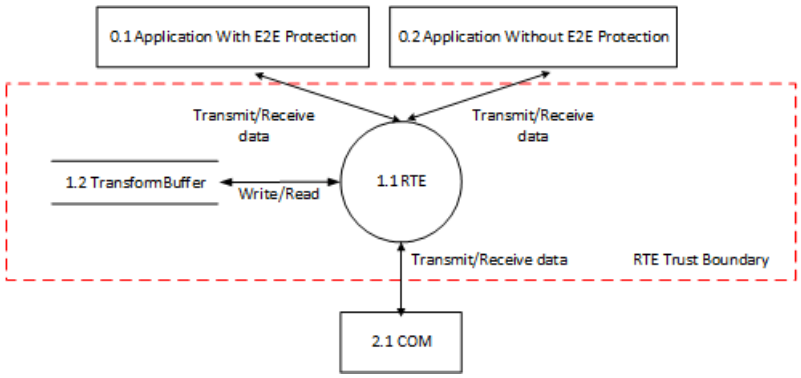

Figure 6.4: Detailed model of RTE module.

6.6.5 COM

The Communication (COM) module is layered between the RTE and PduR modules as shown in Figure 6.5. COM provides signal oriented data for the RTE, but also packs the AUTOSAR signals into PDU packets for transmission to lower layer modules through the PduR. The COM module contains four types of buffers. The transmit buffer is used to store PDU packets that are to be transmitted to the lower layers. The shadow buffers are used to group signals. The I-PDU and L-PDU buffers store the corresponding PDU type for further handling.

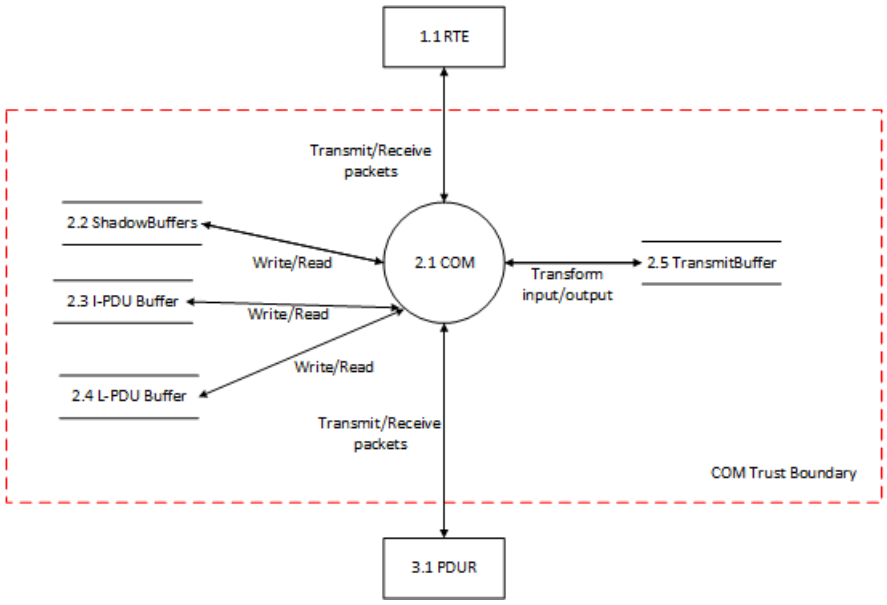

Figure 6.5: Detailed model of COM module.

6.6.6 PduR

The PduR module is responsible for routing of incoming and outgoing PDU packages which are shown in Figure 6.6. In this model it is connected to the CanIF, SecOC and the COM module. The PduR has one buffer, the I-PDU buffer, that is used when the PduR is used as a gateway between different networks, for example if the PduR is used to gateway messages between two different CAN networks.

6.6.7 SecOC

Figure 6.7 shows that the SecOC module is connected to the RTE, CAL, NvM, and PduR modules. The connection to the RTE is used for key and counter management. SecOC uses the CAL module to perform the MAC calculations. It is also connected to the NvM for storage of values when the ECU is powered down. Lastly, the PduR is used to route the packets to and from SecOC to the correct modules. SecOC has two buffers, one for packets that are waiting to be verified and sent to the application, and one buffer for packets that are waiting to be authenticated and sent to another ECU.

6.6.8 CanIf

The CAN Interface module provides the services of the CAN driver to the upper layer communications, shown in Figure 6.8. It converts CAN frames into PDU packets which the rest of AUTOSAR can use. It has one transmit buffer used to store packets that is to be sent, in the event that the CAN module is busy and is unable to handle the packet immediately.

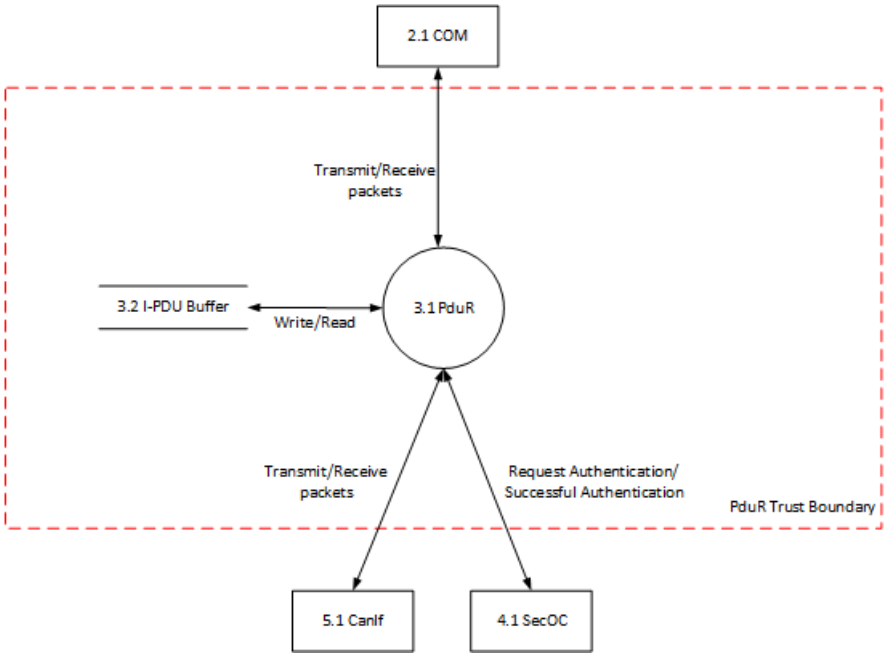

Figure 6.6: Detailed model of PduR module.

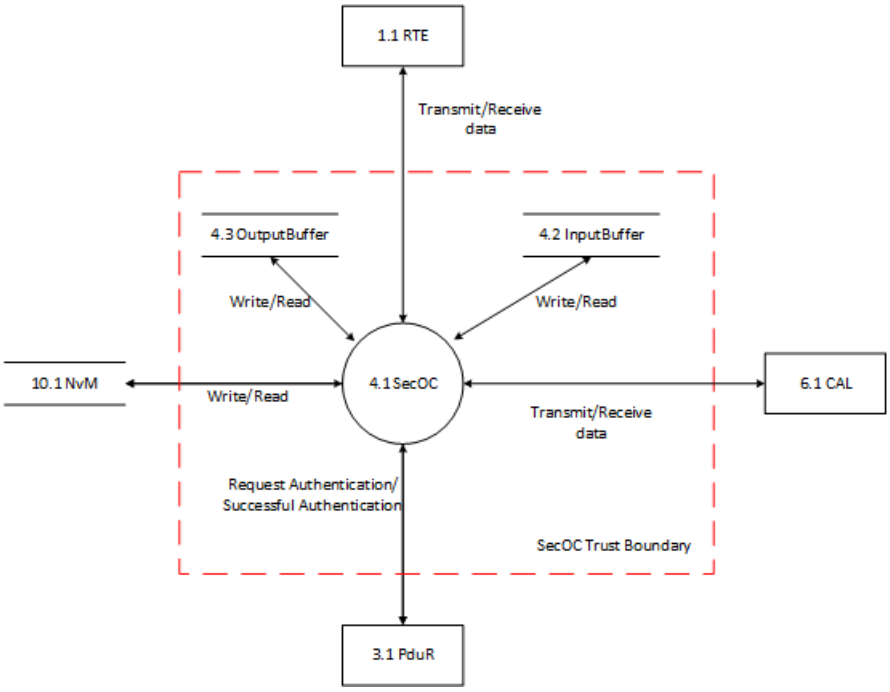

Figure 6.7: Detailed model of SecOC module.

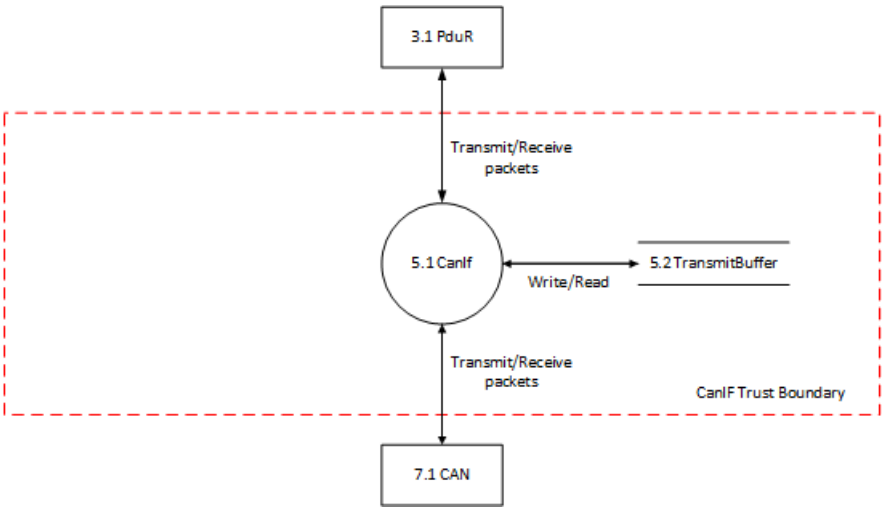

Figure 6.8: Detailed model of CanIf module.

6.6.9 CAL

The CAL module is divided into two parts, firstly the CAL module, and then the CPL module. They are located in the same diagram since they are specified in the same specification and are dependent on each other. Along the two modules, there is a buffer that is used by both modules to store temporary data used during the calculations of the cryptographic functions. This can be seen in Figure 6.9. The only external module that CAL communicates with is the SecOC module, which uses CAL to generate and verify MAC values.

6.6.10 CAN

The Controller Area Network (CAN) driver is part of the lowest layer, which performs the hardware access to the upper layer, shown in the Figure 6.10. The CAN module is independent from the hardware and provides services for callback functions and transmission initializations of the CanIF module for notifying events.

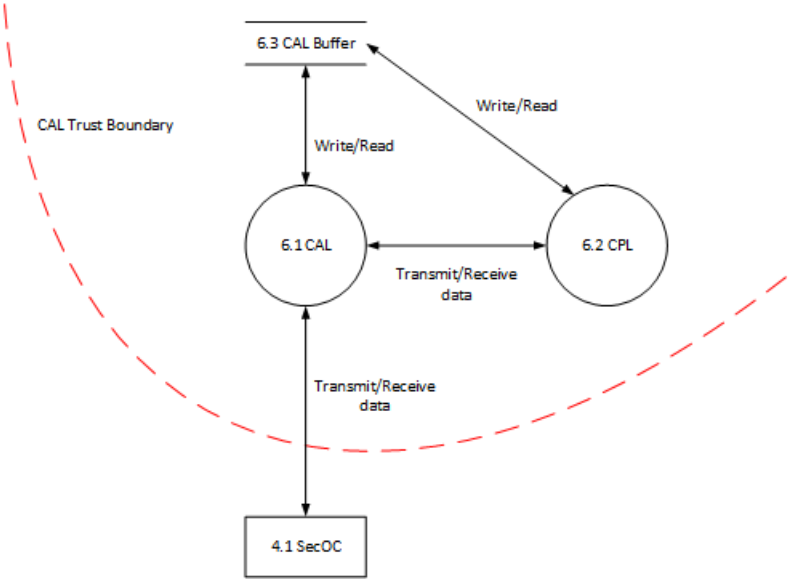

Figure 6.9: Detailed model of CAL module.

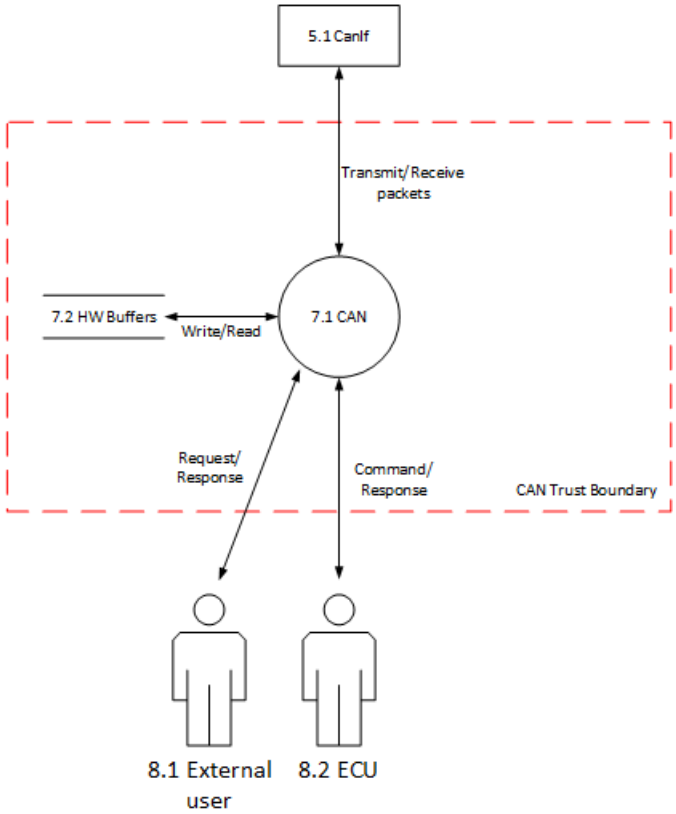

Figure 6.10: Detailed model of CAN module.

6.7 Delimitations

The model used is limited to a small part of the AUTOSAR model. The decision was made to focus on the communication stack and the communication over the CAN network. The focus is on the CAN network since CAN has become the dominant communcation network for in-car communications since 2008 [17]. It is also in the communication stack that the SecOC module is located and that made it a natural point of interest for the study. The model is also focused only on the BSW and will not include the application more than as an external interactor.

7

Case Study

This chapter provides information about the case study of this thesis work.

The study is about two individuals each applying one of the two variants of STRIDE to the initial model of AUTOSAR described in Chapter 6 and then comparing the variants to see what the differences are. The application of the two STRIDE variants will perform step six and seven from the STRIDE process described in Section 2.3. Alongside the comparison of the two variants of STRIDE the security of AUTOSAR will be evaluated as well, in order to analyze which security problems exist in AUTOSAR.

This study will not include step eight, determine the risk, and step nine, plan mitigations, from the aforementioned STRIDE process since the focus of this study is to investigate the differences between the STRIDE variants and evaluate the SecOC module.

7.1 STRIDE-per-Element

The base for the STRIDE-per-Element analysis was the initial model described in Chapter 6. The threat modeling process described in Howard and Lepner [13] was then applied to the model. The first plan was to use the SDL threat modeling tool to create the model as well as to perform the STRIDE analysis, but the tool proved to be very unstable so the analysis was performed by hand without tool support.

7.2 STRIDE-per-Interaction

The initial model was imported into the Microsoft Threat Modeling Tool 2014. The suggested threats relative to the model were shown by switching to the analysis view in the tool.

The tool provides a description of each threat and, eventually, how it could be mitigated. The threats that were mitigated by SecOC were removed from the table.

The result of the STRIDE-per-interaction analysis was then filled in the Table B.1 to show what threats apply to each interaction. These threats were derived from most of the suggested threats from the tool. The description of the threat per interaction is shown in Table B.2. The tables were then used for the comparison of STRIDE variants.

7.3 Comparison of the STRIDE variants

This section describes the factors that will be analysed for the comparison of the STRIDE variants.

7.3.1 Quantitative comparison

The comparison was divided into several different categories. The number of relevant threats found was compared, as well as the number of irrelevant threats found, and the precision. The distribution of the threats into the STRIDE categories was also covered.

In addition, the time spent performing the two variants of STRIDE was compared as well.

Precision is a measure of how good the STRIDE evaluation corresponds with the reality, i.e., what fractions of the threats found are relevant [22]. The precision is calculated with Equation 7.1 where TP stands for the true positives and FP, false positives.

$$P = \frac{TP}{TP + FP} \tag{7.1}$$

7.3.2 Patterns

The threats found by the two variants of STRIDE was compared to find similarities and differences in the threats found. Patterns in the threats found was investigated. Both the true positives and the false positives was be compared.

Similarities & differences in True Positives and False Positives

This comparison is done by assembling a list of the advantages and disadvantages of the variants and comparing them with each other. The list contains the experiences of the two practitioners of STRIDE.

8

Results

This chapter presents the result of the case study of the STRIDE variants analysis. The structure of this chapter is divided with the aim of answering the research questions based on the results. First, the STRIDE comparison is described, where calculations and findings are shown in tables and described. Second, the first research question defined in section 4 is answered based on the STRIDE comparison results. The last section answers the second and last research question.

8.1 STRIDE comparison

In this part the result of the comparison between STRIDE-per-element and STRIDE-per-interaction is presented.

8.1.1 Quantitative comparison

As shown in Table 8.1, a total of 99 threats were found with STRIDE-per-element. 45 of the threats were false positives and 54 threats were true positives. Overall the precision was 54,55%. The main part of the threats was found in the Tampering and Denial of service categories, while no threats were found in the Elevation of privilege category.

The result of STRIDE-per-interaction is shown in Table 8.2, 114 threats were found in total. 83 threats were false positive while 31 threats were true positive. Total precision was 27,19% and threats were found for each category but no true positive threats were found in Elevation of privilege.

Based on the above results, STRIDE-per-element and STRIDE-per-interaction find about the same amount of threats, but STRIDE-per-element has significantly higher precision.

In Figure 8.1 the precision of STRIDE-per-element and STRIDE-per-interaction is compared. It can be seen that in STRIDE-per-element, most threats were found in the

Table 8.1: Descriptive statistics for STRIDE-per-element.

	S	**T**	**R**	**I**	**D**	**E**	**Total**
Total threats	3	31	4	16	45	0	99
False Positive	0	30	2	13	0	0	45
True Positive	3	1	2	3	45	0	54
Precision	100,00%	3,23%	50,0%	18,75%	100,0%	100,00%	54,55%

Table 8.2: Descriptive statistics for STRIDE-per-interaction.

	S	**T**	**R**	**I**	**D**	**E**	**Total**
Total threats	20	6	8	26	21	33	114
False Positive	18	5	2	22	4	33	84
True Positive	2	1	6	4	17	0	30
Precision	10,00%	16,67%	75,00%	15,38%	80,95%	0,00%	26,32%

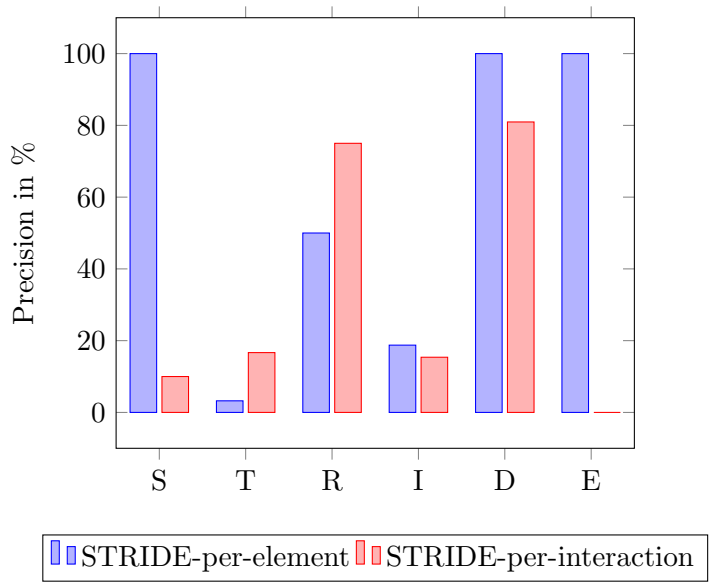

Figure 8.1: Comparison of the distribution of precision across the STRIDE categories.

denial of service category. In STRIDE-per-interaction, the threats were more evenly spread, but still with a large amounts of threats in the denial of service category.

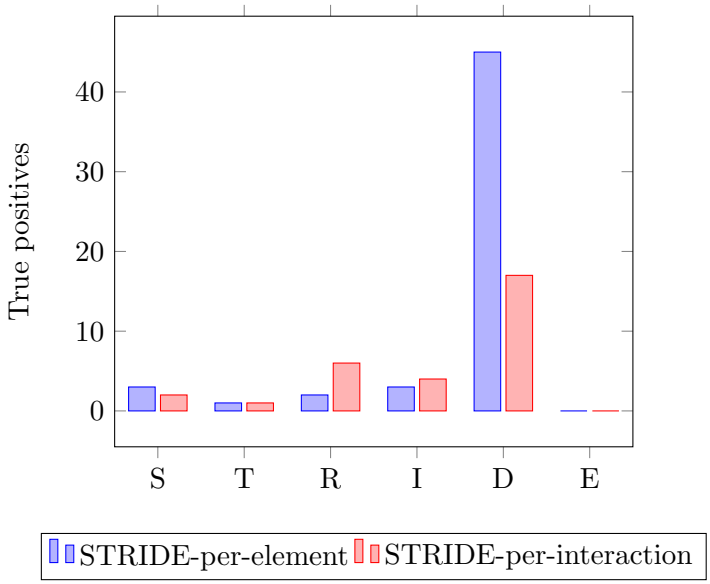

Figure 8.2: Comparison of True positives between STRIDE-per-element and STRIDE-per-interaction.

As shown in Figure 8.2, the overall precision is low for most categories. The most outstanding categories are spoofing(in the case of STRIDE-per-element), repudiation and denial of service where the precision is high. The category with the biggest differences between the two STRIDE variants is spoofing and elevation of privilege, where STRIDE-per-element have 100% while STRIDE-per-element have 10% and 0%.

Figure 8.3 compares the distribution of false positives between the two STRIDE variants. This is the category where the biggest difference between the two variants can be found. STRIDE-per-interaction found much more false positives in all categories except tampering. The false positives found by STRIDE-per-element was almost exclusively found in the tampering and information disclosure categories, while STRIDE-per-interaction had its false positives mostly in the spoofing, information disclosure and elevation of privilege categories.

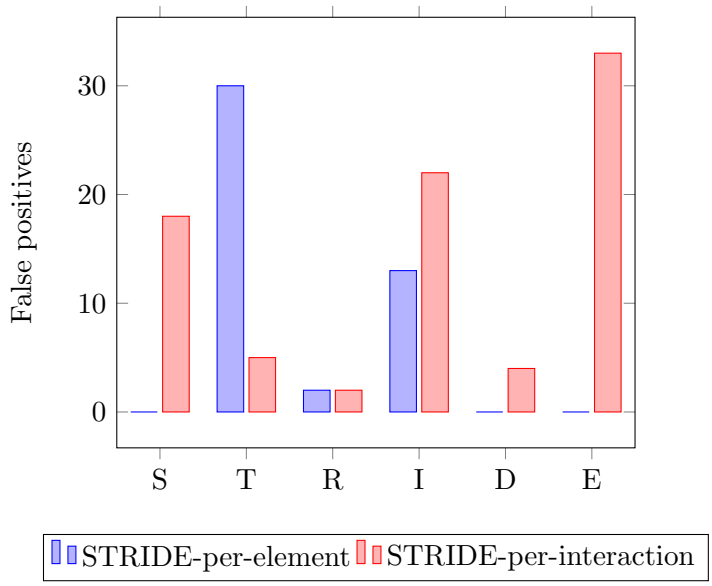

Figure 8.3: Comparison of false positives between STRIDE-per-element and STRIDE-per-interaction.

Time consumption

Based on the result of the time spent on applying the STRIDE variants shown in Table 8.3, applying STRIDE-per-element takes less time than STRIDE-per-element.

Table 8.3: Time spent applying the STRIDE variants.

STRIDE-per-element	STRIDE-per-interaction
26 h	32,5 h

8.1.2 Patterns

The patterns when comparing the STRIDE variants are described in this section. The similarities and differences are shown.

8.1.3 STRIDE evaluation

Based on the results of applying the STRIDE variants shown in Table 8.4, STRIDE-per-interaction is better if the outcome needs to be understood by non security experts but there is limited information about this variant and no other examples to follow than the book written by Shostack [32]. It is also time consuming and complex to apply STRIDE-per-interaction since each interaction to be filled into a table.

Table 8.4: Advantages and disadvantages of the STRIDE variants

STRIDE-per-element		**STRIDE-per-interaction**	
Advantages	Disadvantages	Advantages	Disadvantages
• Much training information available	• Unusable tool	• Easy to understand the threats	• Time-consuming
• Relatively easy to perform	• Rely on the experience of the user	• Easy-to-use tool	• Limited documentation
			• Complex to apply to bigger system

Similarities & differences in True Positives and False Positives

The threat descriptions are similar in both STRIDE variants even though STRIDE-per-element is based on brainstorming and STRIDE-per-interaction is based on the Microsoft Threat Modeling Tool 2014.

True positive

The true positives were similar between the two STRIDE variants. Most of the true positives threats were focused on the access to the memory, to read or change the keys used for the SecOC authentication mechanism, or to overburden the ECU to make it crashes.

False positive

The biggest difference between the two STRIDE variants is where the false positives were found. For STRIDE-per-element, the bulk of the false positives were found in the tampering and information disclosure categories, while STRIDE-per-interaction had most of the false positives in the spoofing, information disclosure and elevation of privilege categories.

The types of threats found differed as well. In the information disclosure category, the threats found by STRIDE-per-element were focused on disclosure of the key used for the SecOC authentication mechanism, while STRIDE-per-interaction had a more broad view and the threats were concerning all types of information handled by the ECU.

8.2 Which variant of STRIDE (STRIDE-per-Element vs STRIDE-per-Interaction) yields better results with regard to threat modeling?

This section answers the first research question that is defined in section 4.

Based on the statistics and the advantages & disadvantages, STRIDE-per-element was found to be better suited for use in the automotive domain and AUTOSAR. The precision for STRIDE-per-element is 54,55% compared to 26,32% for STRIDE-per-interaction. Table 8.5 shows which STRIDE variant that had the best values for each category.

Table 8.5: Comparison between per-element and per-interaction

	STRIDE-per-element	**STRIDE-per-interaction**
True positives	X	
False positives	X	
Precision	X	
Time spent	X	

8.3 What are the major security threats that SecOC mitigates?

The addition of SecOC to AUTOSAR mitigates most Tampering and Spoofing threats to the system.

The threat from tampering is reduced with the help of SecOC. The major tampering threat comes from an attacker changing the messages that is sent over the CAN network as well as tampering with the memory of the ECU. The tampering on the CAN network is mitigated by the MAC added by SecOC. Tampering with the memory via e.g. Universal Measurement and Calibration Protocol (XCP) could allow an attacker to read or change the keys used by the MAC. This would however require that the attacker knows the memory address where the key is stored.

The spoofing threats to the AUTOSAR communication stack come from two directions. First from the application side, this threat is however deemed to be unlikely to happen since the applications are statically linked to the system and are developed and/or integrated by the manufacturer of the vehicle. The other threat comes from the CAN network, which is the threat that SecOC mitigates. The addition of authentication removes the threat from other ECUs spoofing their identity.

8.3.1 Threats remaining

The remaining threats to AUTOSAR after the addition of SecOC is mainly coming from two categories. The first is different types of attacks against the memory, to get a hold of the encryption keys used by SecOC and therefor circumvent the security mechanisms added by SecOC. The other is Denial of service attacks and flooding of the CAN network.

9

Discussion

This chapter discusses the results presented in Chapter 8 and connects them to the theory presented in Chapter 2 and challenges in finding those results.

9.1 Implementation

The implementation of the SecOC and CAL modules were implemented by following the specifications provided by AUTOSAR [2]. As described earlier in Section 2.1.2, SecOC was released with the release of AUTOSAR 4.2.1 and this meant that the specification was untested and few had implemented it. This led to the specification being very fuzzy on some details and maybe not tested completely. This in turn led to the time spent on the implementation was longer than planned and the scope of the implementation had to be reduced.

9.1.1 Problems with SecOC specification

The SecOC specification contains a list of configurations parameters used in the implementation, but the problem was that there were no descriptions about the use of these configurations. In conclusion, the lack of information of the SecOC specification made it hard to finalize the implementation of the whole module and as described previously, the scope of the implementation was reduced more than initially planned.

9.2 DFD Creation

9.2.1 Modeling

This section discusses how to create a good model of the AUTOSAR platform and a reduction of the model created in this thesis could be used instead.

Creating a good model of the AUTOSAR platform

AUTOSAR is a platform software developers can build their system upon, and without having the full system implementation, making a model of the system can be a problem. The AUTOSAR platform is highly configurable and the threats against the system depends on how it is configured. This makes the modelling a challenge since the configuration and use of the platform can change a lot between different projects.

Reduction of the model

The threats that were found by the STRIDE variants were evaluated, it was mainly in the interfaces to the applications and the CAN network that the threats were found. With this in mind, the model might have been overly detailed. The model could has been reduced by joining the communication stack to one element and still get the same results. The reduced model can be seen in Figure 9.1. The only part where the full model generated more useful threats is in the denial-of-service case, but that is mostly because it is unclear which part of the system is the bottle neck and would stop working first. The evaluation of that could be performed as a separate investigation and might not be the focus for the STRIDE process.

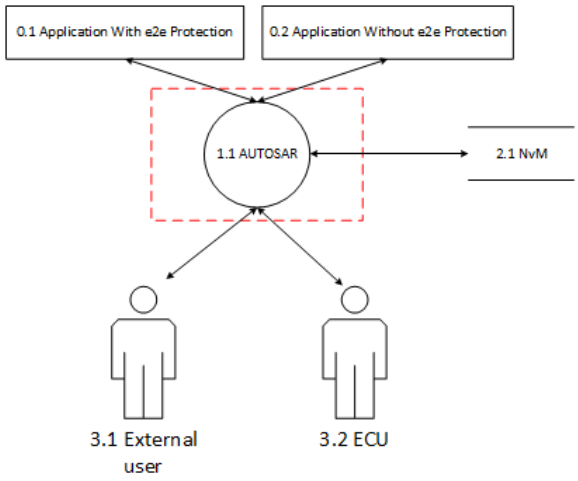

Figure 9.1: Reduced threat model.

9.3 Threats elicitation

9.3.1 STRIDE variants challenges

One of the purposes of the thesis is to evaluate the security by using two variants of STRIDE. Therefore, this section discusses the challenges of applying the variants of STRIDE to the AUTOSAR platform.

The low precision and high amount of false positives found by the two STRIDE variants could be the result of that STRIDE is a more focused application development outside the embedded environment, such as web development, and therefore is focused on the wrong type of threats for the automotive and AUTOSAR environment.

STRIDE-per-Element

The amount of threats found and the quality of the threats found relies largely on the participants of the STRIDE process, therefore it can be challenging to perform STRIDE-per-element the first times, especially without the help of the SDL threat modeling tool, which unfortunately was too unstable to be relied on. The tool was prone to crashing, and the model created was corrupted making the tool unable to load the model.

Most of the examples and instructions that exist for STRIDE-per-element are focused on web applications, and they do not seem to be of much use for adaptation into the embedded environment.

STRIDE-per-Interaction

Even though the STRIDE-per-interaction tool could generate a full report of each threat for each interaction, it was quite hard to just have a quick look of the report to get an idea of what the major threats of the system were. The STRIDE-per-interaction was done manually by filling out the threat applicability table based on the outcome from the tool. It was both time consuming and complicated but it was easier to understand the threats of the whole system by looking at the table after filling out the threats manually.

There is limited information about the STRIDE-per-interaction, since this variant was released one year ago. It was quite challenging to figure out if STRIDE-per-interaction variant is done in a correct way since there are no other examples or information about it except in the book written by Shostack [32].

Time consumption

The reason that STRIDE-per-interaction took more time to perform was that it is more complex to fill the tables manually even though the tool worked perfectly. In comparison, the STRIDE-per-element is easier to use and takes less time than STRIDE-per-interaction.

9.3.2 Threat modeling tools

The tools provided by Microsoft to assist with the modeling and the STRIDE analysis are very immature, especially the SDL threat modeling tool that was to be used for the STRIDE-per-element modeling. The tool was abandoned and the analysis was performed without tool support. This made the STRIDE analysis harder and more time consuming that it would have been with the help of the tools.

The tool used for STRIDE-per-interaction (Microsoft Threat Modeling Tool 2014) was compared to the tool used for STRIDE-per-element quite straight forward to use.

The outcome of the initial model was as expected easy to understand. As Shostack [32] describes, the threats are easier to understand with this variant because the tool gave both explanation of each threat and suggestions of mitigations for each threat, which was very useful.

9.4 Future work

This section discusses possible future work that extends the work done in this thesis.

Further study on how to create a good model of a system could be conducted. This field is hard, since there is no way to say if the model of the system is good. This is especially hard in the case of the AUTOSAR platform since there is only a platform and not a complete system. The high configurability of AUTOSAR also increases the challenge of creating a model of the platform.

This case study focused on a small part of the AUTOSAR platform, since as described in section 1.1, only a limited part of the AUTOSAR platform was focused. Further study with STRIDE on an extended model of the AUTOSAR platform would be interesting to see what types of threats will be found when larger parts of AUTOSAR is included in the model.

Finally, a larger study with several participants performing the same tasks as this study to see whether the differences observed will be confirmed would be interesting.

Bibliography

[1] Automotive News. (2015) Volvo to unleash self-driving cars on swedish roads. [Online]. Available: http://www.autonews.com/article/20150301/OEM06/303029948/volvo-to-unleash-self-driving-cars-on-swedish-roads

[2] AUTOSAR. (2015) AUTOSAR home. [Online]. Available: http://www.autosar.org/

[3] ——, *Specification of Crypto Abstraction Library.*

[4] ——, *Layered Software Architecture.*

[5] ——, *Specification of Module Secure Onboard Communication.*

[6] N. Beringer, "The connected car security boundaries," *ATZ worldwide*, vol. 115, no. 10, pp. 22–27, 2013. [Online]. Available: http://dx.doi.org/10.1007/s38311-013-0111-x

[7] J. Bryson and P. Gallagher, "Secure hash standard (shs)," *Federal Information Processing Standards, FIPS PUB (180-4)*, 2012.

[8] CANtact. (2015) CANtact. [Online]. Available: http://cantact.io

[9] Common Criteria, "Common criteria for information technology security evaluation, part 1: Introduction and general model," 2012.

[10] M. H. Diallo, J. Romero-mariona, S. E. Sim, T. A. Alspaugh, and D. J. Richardson, "A comparative evaluation of three approaches to specifying security requirements," in *Proceedings of the Twelfth Working Conference on Requirements Engineering: Foundation for Software Quality*, 2006.

[11] EmbUnit. (2012) Embunit unit testing tool. [Online]. Available: http://www.embunit.com

[12] Fraunhofer SIT. (2015) E-safety vehicle intrusion protected applications. [Online]. Available: www.evita-project.org

[13] M. Howard and S. Lepner, *The Security Development Lifecycle: SDL: A Process for Developing Demonstrably More Secure Software (Developer Best Practices)*. Microsoft Press, 2006.

[14] ISO, "26262, road vehicles–functional safety," *International Standard ISO/FDIS*, vol. 26262, 2011.

[15] ISO/IEC, "27000, information technology – security techniques – information security management systems – overview and vocabulary," *International Standard ISO/IEC*, 2014.

[16] L. Kohnfelder and P. Garg, "The threats to our products," *Microsoft Interface, Microsoft Corporation*, 1999.

[17] K. Koscher, A. Czeskis, F. Roesner, S. Patel, T. Kohno, S. Checkoway, D. McCoy, B. Kantor, D. Anderson, H. Shacham *et al.*, "Experimental security analysis of a modern automobile," in *Security and Privacy (SP), 2010 IEEE Symposium on*. IEEE, 2010, pp. 447–462.

[18] H. Krawczyk, M. Bellare, and R. Canetti, "Rfc 2104: Hmac: Keyed-hashing for message authentication," 1997.

[19] D. Kum, G.-M. Park, S. Lee, and W. Jung, "Autosar migration from existing automotive software," in *Control, Automation and Systems, 2008. ICCAS 2008. International Conference on*. IEEE, 2008, pp. 558–562.

[20] K. Lemke, C. Paar, and M. Wolf, *Embedded Security in Cars: Securing Current and Future Automotive IT Applications*. Springer-Verlag, 2006.

[21] M. S. Lund, B. Solhaug, K. Stølen, and S. (e-book collection), *Model-driven risk analysis: the CORAS approach*. Berlin; London: Springer, 2010.

[22] C. D. Manning, P. Raghavan, and H. Schütze, *Introduction to information retrieval*. Cambridge university press Cambridge, 2008, vol. 1.

[23] J. McDermott and C. Fox, "Using abuse case models for security requirements analysis," in *Computer Security Applications Conference, 1999.(ACSAC'99) Proceedings. 15th Annual*. IEEE, 1999, pp. 55–64.

[24] MISRA, "Guidelines for the use of the c language in critical systems."

[25] S. Myagmar, A. J. Lee, and W. Yurcik, "Threat modeling as a basis for security requirements," in *Symposium on requirements engineering for information security (SREIS)*, vol. 2005, 2005, pp. 1–8.

[26] A. L. Opdahl and G. Sindre, "Experimental comparison of attack trees and misuse cases for security threat identification," *Information and Software Technology*, vol. 51, no. 5, pp. 916–932, 2009.

[27] D. M. Powers, "Evaluation: from precision, recall and f-measure to roc, informedness, markedness and correlation," 2011.

[28] C. Robson, "Real world research. 2nd," *Edition. Blackwell Publishing. Malden*, 2002.

[29] P. Runeson and M. Höst, "Guidelines for conducting and reporting case study research in software engineering," vol. 14, no. 2, pp. 131–164, 2009.

[30] P. Saitta, B. Larcom, and M. Eddington, "Trike v. 1 methodology document [draft]," *URL: http://dymaxion. org/trike/Trike_v1_Methodology_Documentdraft. pdf*, 2005.

[31] B. Schneier, "Attack trees," *Dr.Dobb's Journal*, vol. 24, no. 12, pp. 21–29, 12 1999.

[32] A. Shostack, *Threat Modeling: Designing for Security.* US: John Wiley & Sons Ltd, 2014.

[33] ——, "Experiences threat modeling at microsoft," in *Modeling Security Workshop. Dept. of Computing, Lancaster University, UK*, 2008.

[34] SP. (2015) Heavens. [Online]. Available: www.sp.se/en/index/research/dependable_systems/heavens/Sidor/default.aspx

[35] M. Stevens, "On collisions for md5," Master's thesis, Eindhoven University of Technology, 2007.

[36] A. Van Lamsweerde *et al.*, "Requirements engineering: from system goals to uml models to software specifications," 2009.

[37] X. Wang, D. Feng, X. Lai, and H. Yu, "Collisions for hash functions md4, md5, haval-128 and ripemd." *IACR Cryptology ePrint Archive*, vol. 2004, p. 199, 2004.

[38] X. Wang, Y. L. Yin, and H. Yu, "Finding collisions in the full sha-1," in *Advances in Cryptology–CRYPTO 2005.* Springer, 2005, pp. 17–36.

[39] C. Xiaoling, "The future of connected cars," *China Today*, Feb 15 2013.

A

STRIDE-per-Element threats

The rows that are marked red are false positives threats.

Table A.1: Mapping of STRIDE to external entities [13].

#	External Entities	S	T	R	I	D	E
0.1	Application with e2e	X		X			
0.2	Application without e2e	X		X			
8.1	External User	X		X			
8.2	Normal ECU	X		X			

Table A.2: Mapping of STRIDE to processes [13].

#	Processes	S	T	R	I	D	E
1.1	RTE	X	X	X	X	X	X
2.1	COM	X	X	X	X	X	X
3.1	PduR	X	X	X	X	X	X
4.1	SecOC	X	X	X	X	X	X
5.1	CanIF	X	X	X	X	X	X
6.1	CAL	X	X	X	X	X	X
6.2	CPL	X	X	X	X	X	X
7.1	CAN	X	X	X	X	X	X

Table A.3: Mapping of STRIDE to data stores [13].

#	Data Stores	S	T	R	I	D	E
1.2	Transform Buffer		X		X	X	
2.2	Shadow Buffers		X		X	X	
2.3	I-PDU Buffer		X		X	X	
2.4	L-Pdu Buffer		X		X	X	
2.5	Transmit Buffer		X		X	X	
3.2	IPDU Buffer		X		X	X	
4.2	Input Buffer		X		X	X	
4.3	Output Buffer		X		X	X	
5.2	Transmit Buffer		X		X	X	
7.2	HW Buffer		X		X	X	
10.1	NvM		X		X	X	

Table A.4: Mapping of STRIDE to data flows [13].

Data Flows	S	T	R	I	D	E
0.1 ⇔ 1.1		X		X	X	
0.2 ⇔ 1.1		X		X	X	
1.1 ⇔ 2.1		X		X	X	
1.1 ⇔ 1.2		X		X	X	
1.1 ⇔ 4.1		X		X	X	
2.1 ⇔ 2.2		X		X	X	
2.1 ⇔ 2.3		X		X	X	
2.1 ⇔ 2.4		X		X	X	
2.1 ⇔ 2.5		X		X	X	
2.1 ⇔ 3.1		X		X	X	
3.1 ⇔ 3.2		X		X	X	
3.1 ⇔ 4.1		X		X	X	
3.1 ⇔ 5.1		X		X	X	
4.1 ⇔ 4.2		X		X	X	
4.1 ⇔ 4.3		X		X	X	
4.1 ⇔ 10.1		X		X	X	
4.1 ⇔ 6.1		X		X	X	
5.1 ⇔ 5.2		X		X	X	
5.1 ⇔ 7.1		X		X	X	
6.1 ⇔ 6.2		X		X	X	
6.1 ⇔ 6.3		X		X	X	
6.2 ⇔ 6.3		X		X	X	
7.1 ⇔ 7.2		X		X	X	
7.1 ⇔ 8.1		X		X	X	
7.1 ⇔ 8.2		X		X	X	

Table A.5: Elements with possible Spoofing threats.

Threat Type	Element Type	DFD Item Numbers
Spoofing	External entities	(0.1), (0.2), (8.1), (8.2)
	Processes	(1.1), (2.1), (3.1), (4.1), (5.1), (6.1), (6.2), (7.1)

Table A.6: Elements with possible Tampering threats.

Threat Type	Element Type	DFD Item Numbers
Tampering	Processes	(1.1), (2.1), (3.1), (4.1), (5.1), (6.1), (6.2), (7.1)
	Data stores	(1.2), (2.2), (2.3), (2.4), (2.5), (3.2), (4.2), (4.3), (5.2), (7.2), (10.1)
	Data flows	(0.1 ⇔ 1.1), (0.2 ⇔ 1.1), (1.1 ⇔ 2.1), (1.1 ⇔ 1.2), (1.1 ⇔ 4.1), (2.1 ⇔ 2.2), (2.1 ⇔ 2.3), (2.1 ⇔ 2.4), (2.1 ⇔ 2.5), (2.1 ⇔ 3.1), (3.1 ⇔ 3.2), (3.1 ⇔ 4.1), (3.1 ⇔ 5.1), (4.1 ⇔ 4.2), (4.1 ⇔ 4.3), (4.1 ⇔ 10.1),(4.1 ⇔ 6.1),(5.1 ⇔ 5.2), (5.1 ⇔ 7.1), (6.1 ⇔ 6.2), (6.1 ⇔ 6.3), (7.1 ⇔ 7.2), (7.1 ⇔ 8.1), (7.1 ⇔ 8.2)

Table A.7: Elements with possible Repudiation threats.

Threat Type	Element Type	DFD Item Numbers
Repudiation	External entities	(0.1), (0.2), (8.1), (8.2)
	Processes	(1.1), (2.1), (3.1), (4.1), (5.1), (6.1), (6.2), (7.1)

Table A.8: Elements with possible Information Disclosure threats.

Threat Type	Element Type	DFD Item Numbers
Information Disclosure	Processes	(1.1), (2.1), (3.1), (4.1), (5.1), (6.1), (6.2), (7.1)
	Data stores	(1.2), (2.2), (2.3), (2.4), (2.5), (3.2), (4.2), (4.3), (5.2), (7.2), (10.1)
	Data flows	(0.1 ⇔ 1.1), (0.2 ⇔ 1.1), (1.1 ⇔ 2.1), (1.1 ⇔ 1.2), (1.1 ⇔ 4.1), (2.1 ⇔ 2.2), (2.1 ⇔ 2.3), (2.1 ⇔ 2.4), (2.1 ⇔ 2.5), (2.1 ⇔ 3.1), (3.1 ⇔ 3.2), (3.1 ⇔ 4.1), (3.1 ⇔ 5.1), (4.1 ⇔ 4.2), (4.1 ⇔ 4.3), (4.1 ⇔ 10.1),(4.1 ⇔ 6.1),(5.1 ⇔ 5.2), (5.1 ⇔ 7.1), (6.1 ⇔ 6.2), (6.1 ⇔ 6.3), (7.1 ⇔ 7.2), (7.1 ⇔ 8.1), (7.1 ⇔ 8.2)

Table A.9: Elements with possible Denial Of Service threats.

Threat Type	Element Type	DFD Item Numbers
DoS	Processes	(1.1), (2.1), (3.1), (4.1), (5.1), (6.1), (6.2), (7.1)
	Data stores	(1.2), (2.2), (2.3), (2.4), (2.5), (3.2), (4.2), (4.3), (5.2), (7.2), (10.1)
	Data flows	(0.1 ⇔ 1.1), (0.2 ⇔ 1.1), (1.1 ⇔ 2.1), (1.1 ⇔ 1.2), (1.1 ⇔ 4.1), (2.1 ⇔ 2.2), (2.1 ⇔ 2.3), (2.1 ⇔ 2.4), (2.1 ⇔ 2.5), (2.1 ⇔ 3.1), (3.1 ⇔ 3.2), (3.1 ⇔ 4.1), (3.1 ⇔ 5.1), (4.1 ⇔ 4.2), (4.1 ⇔ 4.3), (4.1 ⇔ 10.1),(4.1 ⇔ 6.1),(5.1 ⇔ 5.2), (5.1 ⇔ 7.1), (6.1 ⇔ 6.2), (6.1 ⇔ 6.3), (7.1 ⇔ 7.2), (7.1 ⇔ 8.1), (7.1 ⇔ 8.2)

Table A.10: Elements with possible Elevation Of Privilege threats.

Threat Type	Element Type	DFD Item Numbers
EoP	Processes	(1.1), (2.1), (3.1), (4.1), (5.1), (6.1), (6.2), (7.1)

Table A.11: Spoofing threats against the system.

Element	Preconditions	Threat	Impact
0.1	The application know what other applications run on the ECU.	The application pretends to be another application to read the information sent to it.	The application read data that was meant for another application and the original application might not get the data.
0.2	The application know what other applications run on the ECU.	The application pretends to be another application to read the information sent to it.	The application read data that was meant for another application and the original application might not get the data.
8.1	An attacker have connected a external device to the CAN network.	The external user pretends to be another ECU and send commands to our ECU.	The ECU can be tricked into performing actions that was not intended.
8.2	Another ECU is compromised and the attacker can control it.	The external ECU pretends to be another ECU and send commands to our ECU.	The ECU can be tricked into performing actions that was not intended.

Table A.12: Tampering threats against the system, Part 1.

Element	Preconditions	Threat	Impact
1.1	A malicious user have access to the RTE module.	Tampering with the RTE module could lead to changes in the received or sent packets.	The application or another ECU get the wrong information.
2.1	A malicious user have access to the COM module.	Tampering with the COM module could lead to changes in the received or sent packets.	The application or another ECU get the wrong information.
3.1	A malicious user have access to the PduR module.	Tampering with the PduR module could lead to changes in the received or sent packets.	The application or another ECU get the wrong information.
4.1	A malicious user have access to the SecOC module.	Tampering with the SecOC module could lead to changes in the received or sent packets.	The application or another ECU get the wrong information.
5.1	A malicious user have access to the CanIF module.	Tampering with the CanIF module could lead to changes in the received or sent packets.	The application or another ECU get the wrong information.
6.1	A malicious user have access to the CAL module.	Tampering with the CAL module could lead to incorrect authentication or verification.	Incorrect packets get verified and sent to application or the incorrect Authentication information is attached to a packet.

Table A.13: Tampering threats against the system, Part 2.

Element	Preconditions	Threat	Impact
6.2	A malicious user have access to the CPL module.	Tampering with the CPL module could lead to incorrect authentication or verification.	Incorrect packets get verified and sent to application or the incorrect Authentication information is attached to a packet.
7.1	A malicious user have access to the CAN module.	Tampering with the CAN module could make the ECU send messages on the CAN network.	The ECU could try to manipulate another ECU to perform some action.
7.1	A malicious user have access to the CAN module.	Tampering with the CAN module could lead to changes in the received or sent packets.	The application or another ECU get the wrong information.
1.2	A malicious user have access to the Transform buffer.	Tampering with the Transform buffer could allow an attacker to change the packets sent from the ECU.	Packets that is to have a transform applied may be changed or removed.
2.2	A malicious user have access to the shadow buffer.	Tampering with the Shadow buffer could allow an attacker to change the message sent to a signal group.	The attacker may modify the messages sent by the COM module.
2.3	A malicious user have access to the I-PDU buffer.	Tampering with the I-PDU buffer could allow an attacker to change the message sent from the module.	The attacker may modify the messages sent by the COM module.
2.4	A malicious user have access to the L-PDU buffer.	Tampering with the L-PDU buffer could allow an attacker to change the message sent from the module.	The attacker may modify the messages sent by the COM module.

Table A.14: Tampering threats against the system, Part 3.

Element	Preconditions	Threat	Impact
2.5	A malicious user have access to the Transmit buffer.	Tampering with the transmit buffer could allow an attacker to change the message sent from the module.	The attacker may modify the messages sent by the COM module.
3.2	A malicious user have access to the IPDU Buffer.	Tampering with the IPDU buffer could allow an attacker to change the packets that is being sent to another network.	The messages that is being sent through the gateway may be changed or corrupted.
6.3	A malicious user have access to the CAL buffer.	Tampering with the CAL buffer could allow an attacker to verify packets that are supposed to be rejected.	The verification of packets that should not be verified.
7.2	A malicious user have access to the HW buffer.	Tampering with the hardware buffer could allow an attacker to modify packets sent to or from the ECU.	The packets sent or received by the ECU may be changed by the attacker.
7.2	A malicious user have access to the HW buffer.	Tampering with the hardware buffer could allow an attacker to add packets to be sent to or from the ECU.	The attacker could add packets and send to the CAN network, or make it seem like the ECU have received packets that the attacker added.
10.1	A malicious user have access to the NvM module.	Tampering with the NvM could allow an attacker to change the freshness value.	This could allow an attacker to easier manipulate the MAC and allow for modification of
0.2 $\Leftrightarrow$ 1.1	A malicious user have access to the data flow between the application and the RTE.	Tampering with this data flow could change the data sent to or from the application.	The application get or send the wrong information.

Table A.15: Tampering threats against the system, Part 4.

Element	Preconditions	Threat	Impact
1.1 ⇔ 2.1	A malicious user have access to the data flow between the RTE and COM.	Tampering with this data flow could change the data sent to or from the application.	The application get or send the wrong information
1.1 ⇔ 1.2	A malicious user have access to the data flow between the RTE and the transform buffer.	Tampering with this data flow could allow an attacker to change the data that is sent to the transform functionality.	Packets that is to have a transform applied may be changed or removed.
1.1 ⇔ 4.1	A malicious user have access to the data flow between the RTE and SecOC.	Tampering with this data flow could allow an attacker to manipulate the key that is used for the MAC, as well as the freshness value.	The attacker may modify the keys or freshness value that SecOC use for creating the MAC value. This could make it easier to spoof messages.
2.1 ⇔ 2.2	A malicious user have access to the data flow between the COM module and the shadow buffer.	Tampering with this data flow could allow an attacker to change the message sent to a signal group.	The attacker may modify the messages sent by the COM module.
2.1 ⇔ 2.3	A malicious user have access to the data flow between the COM module and the I-PDU buffer.	Tampering with this data flow could allow an attacker to change the message sent from the module.	The attacker may modify the messages sent by the COM module.
2.1 ⇔ 2.4	A malicious user have access to the data flow between the COM module and the L-PDU buffer.	Tampering with this data flow could allow an attacker to change the message sent from the module.	The attacker may modify the messages sent by the COM module.
2.1 ⇔ 2.5	A malicious user have access to the data flow between the COM module and the transmit buffer.	Tampering with this data flow could allow an attacker to change the message sent from the module.	The attacker may modify the messages sent by the COM module.

Table A.16: Tampering threats against the system, Part 5.

Element	Preconditions	Threat	Impact
2.1 ⇔ 3.1	A malicious user have access to the data flow between the COM and PduR module.	Tampering with this data flow could change the data sent to or from the application.	The application get or send the wrong information.
3.1 ⇔ 3.2	A malicious user have access to the data flow between the PduR module and the IPDU buffer.	Tampering with this data flow could allow an attacker to manipulate the message that is sent to other networks.	Allow the attacker to change messages that is to be relayed to other networks.
3.1 ⇔ 4.1	A malicious user have access to the data flow between the PduR and SecOC module.	Tampering with this data flow could change the data sent to or from the application.	The application get or send the wrong information.
4.1 ⇔ 4.3	A malicious user have access to the data flow between the SecOC module and the output buffer.	Tampering with this data flow could allow an attacker to change the data that is sent by the ECU.	The attacker could modify the packet before the authentication information is attached, making the modification undetectable.
4.1 ⇔ 10.1	A malicious user have access to the data flow between the SecOC module and the output buffer.	Tampering with this data flow could allow an attacker to change the data that is stored in persistent memory, such as the freshness value.	If the incorrect freshness value is stored in the persistent memory, SecOC wont be able to verify packets, and other ECUs wont be able to verify packets from this ECU.
4.1 ⇔ 6.1	A malicious user have access to the data flow between the CAL and SecOC module.	Tampering with the data flow from CAL to SecOC could allow an attacker to tell SecOC to verify packets that should not pass verification.	Incorrect packets get verified and sent to application or the incorrect Authentication information is attached to a packet.

Table A.17: Tampering threats against the system, Part 6.

Element	Preconditions	Threat	Impact
6.1 ⇔ 6.2	A malicious user have access to the data flow between the CAL and CPL module.	Tampering with this data flow could allow an attacker to verify packets that should not pass verification.	Incorrect packets get verified and sent to application or the incorrect Authentication information is attached to a packet.
6.1 ⇔ 6.3	A malicious user have access to the data flow between the CAL module and the CAL buffer.	Tampering with this data flow could allow an attacker to verify packets that should not pass verification.	Incorrect packets get verified and sent to application or the incorrect Authentication information is attached to a packet.
6.2 ⇔ 6.3	A malicious user have access to the data flow between the CPL module and the CAL buffer.	Tampering with this data flow could allow an attacker to verify packets that should not pass verification.	Incorrect packets get verified and sent to application or the incorrect Authentication information is attached to a packet.

Table A.18: Repudiation threats against the system.

Element	Preconditions	Threat	Impact
0.1		The application can send malicious data to the RTE and deny doing it.	The ECU have no way to prove what application sent the data.
0.2		The application can send malicious data to the RTE and deny doing it.	The ECU have no way to prove what application sent the data.
8.1		The external user can send malicious data to the ECU and deny doing it.	The external user can send data to the ECU and there is no way to prove that it did.
8.2	Another ECU is compromised and the attacker can control it.	Another ECU can send malicious data to the ECU and deny doing it.	The ECU can send data to the ECU and there is no way to prove that it did.

Table A.19: Information Disclosure threats against the system, Part 1.

Element	Preconditions	Threat	Impact
4.1	A malicious user have access to read the information stored in the SecOC module.	A malicious user read the freshness values.	The user can easier spoof messages if it knows the current freshness values.
4.1	A malicious user have access to read the information stored in the SecOC module.	A malicious user read the encryption keys.	The user can spoof messages and get them authenticated and verified.
6.1	A malicious user have access to read the information stored in the CAL module.	A malicious user read the encryption keys.	The user can spoof messages and get them authenticated and verified.
6.2	A malicious user have access to read the information stored in the CPL module.	A malicious user read the encryption keys.	The user can spoof messages and get them authenticated and verified.
6.3	A malicious user have access to read the information stored in the CAL buffer.	A malicious user read the encryption keys.	The user can spoof messages and get them authenticated and verified.
6.3	A malicious user have access to read the information stored in the CAL buffer.	A malicious user read the freshness values.	The user can easier spoof messages if it knows the current freshness values.
10.1	The malicious user have access to read the non volatile memory.	A malicious user can read the freshness values.	The user can easier spoof messages if it knows the current freshness values.
0.1 $\Leftrightarrow$ 1.1	-	A malicious user can read the data sent from the application.	The user can read data that is sent to or from the application.
0.2 $\Leftrightarrow$ 1.1	-	A malicious user can read the data sent from the application.	The user can read data that is sent to or from the application.

Table A.20: Information Disclosure threats against the system, Part 2.

Element	Preconditions	Threat	Impact
1.1 ⇔ 4.1	The malicious user have access to read the data flow between the RTE and SecOC.	A malicious user read the freshness values.	The user can easier spoof messages if it knows the current freshness values.
1.1 ⇔ 4.1	The malicious user have access to read the data flow between the RTE and SecOC.	A malicious user read the encryption keys.	The user can spoof messages and get them authenticated and verified.
4.1 ⇔ 10.1	The malicious user have access to read the data flow between the SecOC and NvM.	A malicious user read the freshness values.	The user can easier spoof messages if it knows the current freshness values.
4.1 ⇔ 6.1	The malicious user have access to read the data flow between the SecOC and CAL.	A malicious user read the encryption keys.	The user can spoof messages and get them authenticated and verified.
6.1 ⇔ 6.2	The malicious user have access to read the data flow between CAL and CPL.	A malicious user read the encryption keys.	The user can spoof messages and get them authenticated and verified.
6.1 ⇔ 6.3	The malicious user have access to read the data flow between CAL and CAL buffer.	A malicious user read the encryption keys.	The user can spoof messages and get them authenticated and verified.
6.2 ⇔ 6.3	The malicious user have access to read the data flow between CPL and CAL buffer.	A malicious user read the encryption keys.	The user can spoof messages and get them authenticated and verified.

Table A.21: Denial of Service threats against the system, Part 1.

Element	Preconditions	Threat	Impact
1.1	A malicious user flood the RTE with messages from or to the applications.	The RTE get overloaded and can't handle new requests.	None of the applications on the ECU will work.
2.1	A malicious user flood the COM module with messages.	The COM module is flooded with requests and stops working.	The Applications will not be able to communicate with other ECUs.
3.1	A malicious user flood the PduR with packets.	The PduR module is flooded with requests and stops working.	The Applications will not be able to communicate with other ECUs.
4.1	A malicious user flood the SecOC with requests.	The SecOC module is flooded with requests and stops working.	The ECU cant handle packets that require authentication or verification. The ECU can only handle unsecured packets.
5.1	A malicious user flood the CanIF with packets.	The CanIF module is flooded with requests and stops working.	The Applications will not be able to communicate with other ECUs over CAN.
6.1	A malicious user flood the CAL module with requests.	The CAL module is flooded with requests and stops working.	The ECU cant handle packets that require authentication or verification. The ECU can only handle unsecured packets.
6.2	A malicious user flood the CPL module with requests.	The CPL module is flooded with requests and stops working.	The ECU cant handle packets that require authentication or verification. The ECU can only handle unsecured packets.

Table A.22: Denial of Service threats against the system, Part 2.

Element	Preconditions	Threat	Impact
7.1	A malicious user flood the CAN network with packets, overloading the CAN module.	The CAN network get flooded with messages and stops working.	The ECU wont be able to communicate with other ECUs or sensors connected to the CAN network.
1.2	A malicious user flood the RTE with transform requests.	The transform buffer is flooded with data and stops working.	The transform functionality of the RTE will fail.
2.2	A malicious user flood the COM module with messages.	The shadow buffer is flooded with data and stops working.	The Applications will not be able to communicate with other ECUs.
2.3	A malicious user flood the COM module with messages.	The I-PDU buffer is flooded with data and stops working.	The Applications will not be able to communicate with other ECUs.
2.4	A malicious user flood the COM module with messages.	The L-PDU buffer is flooded with data and stops working.	The Applications will not be able to communicate with other ECUs.
2.5	A malicious user flood the COM module with messages.	The Transmit buffer is flooded with data and stops working.	The Applications will not be able to communicate with other ECUs.
3.2	A malicious user flood the PduR with packets.	The IPDU buffer is flooded with data and stops working.	The PduR can no longer work as a gateway between different networks.
4.2	A malicious ECU flood the system with packets to be verified.	The input buffer for SecOC is flooded with data and stops working.	SecOC can't verify any packets and the incoming packets is dropped.
4.3	A malicious application flood the system with packets to be authenticated.	The output buffer for SecOC is flooded with data and stops working.	SecOC can't authenticate any packets and the outgoing packets is dropped.

Table A.23: Denial of Service threats against the system, part 3.

Element	Preconditions	Threat	Impact
5.2	A malicious user flood the CanIF with packets.	The Transmit buffer is flooded with data and stops working.	The ECU can no longer send messages on the CAN network.
6.3	A malicious user flood the CAL module with messages that overload the CAL buffer.	The CAL buffer is flooded with data and stops working.	The ECU cant handle packets that require authentication or verification. The ECU can only handle unsecured packets.
7.2	A malicious user flood the CAN module with messages that overload the hardware buffer.	The hardware buffer is flooded with data and stops working.	The CAN module can't write packets to the buffer and will stop working.
10.1	A malicious user flood the NvM with requests to overload it.	The NvM module is overloaded and stops handling requests.	SecOC can't read or write freshness values to persistent memory.
0.1 $\Leftrightarrow$ 1.1	A malicious user flood the connection between the application and the RTE.	The connection between the application and the RTE get overloaded.	The specific application will fail.
0.2 $\Leftrightarrow$ 1.1	A malicious user flood the connection between the application and the RTE.	The connection between the application and the RTE get overloaded.	The specific application will fail.
1.1 $\Leftrightarrow$ 2.1	A malicious user flood the connection between the RTE and the COM.	The connection between the RTE and the COM modules get overloaded.	The ECU wont be able to communicate with other ECUs or sensors.
1.1 $\Leftrightarrow$ 1.2	A malicious user flood the RTE with transform requests.	The connection between the RTE and the transform buffer get overloaded.	The transform functionality of the RTE will fail.

Table A.24: Denial of Service threats against the system, Part 4.

Element	Preconditions	Threat	Impact
1.1 ⇔ 4.1	A malicious user flood the RTE with SecOC requests.	The connection between the RTE and SecOC get overloaded.	The RTE wont be able to provide Key & counter management services to the applications.
2.1 ⇔ 2.2	A malicious user flood the COM module with messages.	The connection between the COM module and the shadow buffer is overloaded.	The Applications will not be able to communicate with other ECUs.
2.1 ⇔ 2.3	A malicious user flood the COM module with messages.	The connection between the COM module and the I-PDU buffer is overloaded.	The Applications will not be able to communicate with other ECUs.
2.1 ⇔ 2.4	A malicious user flood the COM module with messages.	The connection between the COM module and the L-PDU buffer is overloaded.	The Applications will not be able to communicate with other ECUs.
2.1 ⇔ 2.5	A malicious user flood the COM module with messages.	The connection between the COM module and the Transmit buffer is overloaded.	The Applications will not be able to communicate with other ECUs.
2.1 ⇔ 3.1	A malicious user flood the connection between the COM and PduR module.	The connection between the COM and PduR module is overloaded.	The ECU can no longer send messages on the CAN network.
3.1 ⇔ 3.2	A malicious user flood the PduR with gateway requests.	The connection between the PduR and the IPDU buffer is overloaded.	The PduR can no longer work as a gateway between different networks.
3.1 ⇔ 4.1	A malicious user flood the PduR with requests to the SecOC module.	The connection between the PduR and SecOC modules is overloaded.	The ECU cant handle packets that require authentication or verification. The ECU can only handle unsecured packets.

Table A.25: Denial of Service threats against the system, part 5.

Element	Preconditions	Threat	Impact
3.1 ⇔ 5.1	A malicious user flood the CanIF with packets.	The connection between the PduR and CanIF modules is overloaded.	The ECU can no longer send messages on the CAN network.
4.1 ⇔ 4.2	A malicious user flood SecOC with messages to verify.	The connection between SecOC and the input buffer is overloaded.	SecOC can't verify any packets and the incoming packets is dropped.
4.1 ⇔ 4.3	A malicious user flood SecOC with messages to authenticate.	The connection between SecOC and the output buffer is overloaded.	SecOC can't authenticate any packets and the outgoing packets is dropped.
4.1 ⇔ 10.1	A malicious user flood SecOC with requests to write/read from NvM.	The connection between SecOC and NvM is overloaded.	SecOC can't load/store the freshness values from the persistent memory if this happens during start up or shutdown.
4.1 ⇔ 6.1	A malicious user flood SecOC with messages to verify or authenticate.	The connection between SecOC and CAL is overloaded.	The ECU cant handle packets that require authentication or verification. The ECU can only handle unsecured packets.
5.1 ⇔ 5.2	A malicious user flood CanIF with messages to send.	The connection between CanIF and the Transmit buffer is overloaded.	The ECU can no longer send messages on the CAN network.
5.1 ⇔ 7.1	A malicious user flood CanIF with messages to send.	The connection between CanIF and CAN is overloaded.	The Applications will not be able to communicate with other ECUs over CAN.

Table A.26: Denial of Service threats against the system, part 6.

Element	Preconditions	Threat	Impact
6.1 ⇔ 6.2	A malicious user flood SecOC with messages to verify or authenticate.	The connection between CAL and CPL is overloaded.	The ECU cant handle packets that require authentication or verification. The ECU can only handle unsecured packets.
6.1 ⇔ 6.3	A malicious user flood SecOC with messages to verify or authenticate.	Connection between the CAL module and the CAL buffer is overloaded.	The ECU cant authorize or verify packets.
6.2 ⇔ 6.3	A malicious user flood SecOC with messages to verify or authenticate.	Connection between the CPL module and the CAL buffer is overloaded.	The ECU cant authorize or verify packets.
7.1 ⇔ 7.2	A malicious user flood the CAN module with messages.	Connection between the hardware buffer and the CAN module is overloaded.	The CAN module can't write packets to the buffer and will stop working.
7.1 ⇔ 8.1	A malicious user have access to the CAN network and flood it with messages.	The CAN connection from the ECU is overloaded.	The ECU can't communicate with other ECUs.
7.1 ⇔ 8.2	A malicious user have access to the CAN network and flood it with messages.	The CAN connection from the ECU is overloaded.	The ECU can't communicate with other ECUs.

B

STRIDE-per-Interaction threats

The cells that are marked grey are false positives threats.

Table B.1: STRIDE-per-Interaction table: Threat Applicability

#	ELEMENT	INTERACTION	S	T	R	I	D	E
1	Process (7.1 CAN)	Process sends output to external interactor (8.1 External User)						
2		Process receives input to external interactor (8.1 External User)		X	X		X	X
3		Process sends output to external interactor (8.2 ECU)						
4		Process receives input to external interactor (8.2 ECU)		X	X		X	X
5		Process has outbound data flow to datastore (7.2 HW buffers)						
6		Process has inbound data flow from datastore (7.2 HW buffers)						

Table B.1: STRIDE-per-Interaction table: Threat Applicability (continued)

#	ELEMENT	INTERACTION	S	T	R	I	D	E
7		Process has sends output to another process (5.1 CANIF)						X
8		Process has inbound data flow from a process (5.1 CANIF)						X
9	Data store (7.2 HW Buffers)	Process has outbound data flow to datastore (7.2 HW buffers)					X	
10		Process has inbound data flow from datastore (7.2 HW buffers)				X		
11	Process (5.1 CANIF)	Process has sends output to another process (7.1 CAN)						X
12		Process has sends output to another process (3.1 PduR)						X
13		Process has inbound data flow from a process (7.1 CAN)						X
14		Process has inbound data flow from a process(3.1 PduR)						X
15		Process has outbound data flow to datastore (5.2 Transmit buffer)						
16		Process has inbound data flow from datastore (5.2 Transmit buffer)						
17	Data store (5.2 TransmitBuffer)	Process has outbound data flow to datastore (5.2 Transmit buffer)					X	

Table B.1: STRIDE-per-Interaction table: Threat Applicability (continued)

#	ELEMENT	INTERACTION	S	T	R	I	D	E
18		Process has inbound data flow from datastore (5.2 Transmit buffer)				X		
19	Process (3.1 PduR)	Process has sends output to another process (5.1 CANIF)						X
20		Process has sends output to another process (4.1 SecOC)						X
21		Process has sends output to another process (2.1 COM)						X
22		Process has inbound data flow from a process(5.1 CanIf)						X
23		Process has inbound data flow from a process (4.1 SecOC)				X		X
24		Process has inbound data flow from a process(2.1 COM)						X
25		Process has outbound data flow to datastore (3.2 I-PDU buffer)	X					
26		Process has inbound data flow from datastore (3.2 I-PDU buffer)	X					
27	Data store (3.2 I-PDU Buffer)	Process has outbound data flow to datastore (3.2 I-PDU buffer)					X	
28		Process has inbound data flow from datastore (3.2 I-PDU buffer)				X		

Table B.1: STRIDE-per-Interaction table: Threat Applicability (continued)

#	ELEMENT	INTERACTION	S	T	R	I	D	E
29	Process (4.1 SecOC)	Process has sends output to another process (3.1 PduR)				X		X
30		Process has sends output to another process (6.1 CAL)				X		X
31		Process has inbound data flow from a process(3.1 PduR)						X
32		Process has inbound data flow from a process (6.1 CAL)						X
33		Process has outbound data flow to datastore (4.2 Input Buffer)						
34		Process has outbound data flow to datastore (4.3 Output Buffer)						
35		Process has outbound data flow to datastore (10.1 NvM)			X	X	X	
36		Process has inbound data flow from datastore (4.2 Input Buffer)						
37		Process has inbound data flow from datastore (4.3 Output Buffer)						
38		Process has inbound data flow from datastore (10.1 NvM)			X	X	X	X
39	Data store (10.1 NvM)	Process has inbound data flow from datastore (10.1 NvM)		X				

Table B.1: STRIDE-per-Interaction table: Threat Applicability (continued)

#	ELEMENT	INTERACTION	S	T	R	I	D	E
40		Process has outbound data flow to datastore (10.1 NvM)				X		
41	Data store (4.2 Input Buffer)	Process has outbound data flow to datastore (4.2 Input Buffer)					X	
42		Process has inbound data flow from datastore (4.2 Input Buffer)				X		
43	Data store (4.3 Output Buffer)	Process has outbound data flow to datastore (4.3 Output Buffer)					X	
44		Process has inbound data flow from datastore (4.3 Output Buffer)				X		
45	Process (6.1 CAL)	Process has sends output to another process (6.2 CPL)						X
46		Process has sends output to another process (4.1 SecOC)						X
47		Process has inbound data flow from a process(6.2 CPL)				X		X
48		Process has inbound data flow from a process (4.1 SecOC)				X		X
49		Process has outbound data flow to datastore (6.3 CalBuffer)	X			X		
50		Process has inbound data flow from datastore (6.3 CalBuffer)	X					

Table B.1: STRIDE-per-Interaction table: Threat Applicability (continued)

#	ELEMENT	INTERACTION	S	T	R	I	D	E
51	Data store (6.3 Cal Buffer)	Process has outbound data flow to datastore (6.3 CalBuffer)					X	
52		Process has inbound data flow from datastore (6.3 CalBuffer)				X		
53	Process (6.2 CPL)	Process has sends output to another process (6.1 CAL)				X		X
54		Process has inbound data flow from a process(6.1 CAL)						X
55		Process has outbound data flow to datastore (6.3 CalBuffer)				X	X	
56		Process has inbound data flow from datastore (6.3 CalBuffer)				X		
57	Process (2.1 COM)	Process has outbound data flow to datastore (2.2 ShadowBuffer)	X					
58		Process has inbound data flow from datastore (2.2 ShadowBuffer)	X					
59		Process has outbound data flow to datastore (2.3 I-PDU Buffer)	X					
60		Process has inbound data flow from datastore (2.3 I-PDU Buffer)	X					
61		Process has outbound data flow to datastore (2.4 L-PDU Buffer)	X					

Table B.1: STRIDE-per-Interaction table: Threat Applicability (continued)

#	ELEMENT	INTERACTION	S	T	R	I	D	E
62		Process has inbound data flow from datastore (2.4 L-PDU Buffer)	X					
63		Process has sends output to another process (1.1 RTE)						X
64		Process has sends output to another process (3.1 PduR)						X
65		Process has inbound data flow from a process(1.1 RTE)						X
66		Process has inbound data flow from a process (3.1 PduR)						X
67		Process has outbound data flow to datastore (2.5 Transmit Buffer)	X					
68		Process has inbound data flow from datastore (2.5 Transmit Buffer)	X					
69	Data store (2.5 TransmitBuffer)	Process has outbound data flow to datastore (2.5 Transmit Buffer)					X	
70		Process has inbound data flow from datastore (2.5 Transmit Buffer)				X		
71	Data store (2.2 Shadow Buffers)	Process has outbound data flow to datastore (2.2 ShadowBuffer)					X	
72		Process has inbound data flow from datastore (2.2 ShadowBuffer)				X		

Table B.1: STRIDE-per-Interaction table: Threat Applicability (continued)

#	ELEMENT	INTERACTION	S	T	R	I	D	E
73	Data store (2.4 L-PDU Buffer)	Process has outbound data flow to datastore (2.4 L-PDU Buffer)					X	
74		Process has inbound data flow from datastore (2.4 L-PDU Buffer)				X		
75	Data store (2.3 I-PDU Buffer)	Process has outbound data flow to datastore (2.3 I-PDU Buffer)					X	
76		Process has inbound data flow from datastore (2.3 I-PDU Buffer)				X		
77	Process (1.1 RTE)	Process receives input to external interactor (0.1 E2E with protection)	X	X	X		X	X
78		Process receives input to external interactor (0.2 E2E without protection)	X	X	X	X	X	X
79		Process sends output to external interactor (0.1 E2E with protection)			X		X	
80		Process sends output to external interactor (0.2 E2E without protection)				X	X	
81		Process has sends output to another process (2.1 COM)						X
82		Process has inbound data flow from a process(2.1 COM)						

Table B.1: STRIDE-per-Interaction table: Threat Applicability (continued)

#	ELEMENT	INTERACTION	S	T	R	I	D	E
83		Process has outbound data flow to datastore (2.5 Transform Buffer)	X					
84		Process has inbound data flow from datastore (2.5 Transform Buffer)	X					
85	Data store (1.2 Transform Buffer)	Process has outbound data flow to datastore (2.5 Transform Buffer)					X	
86		Process has inbound data flow from datastore (2.5 Transform Buffer)				X		
87	External Interactor (0.1 Application with E2E Protection)	External interactor gets input from process.	X					
88		External interactor passes input to process.	X					
89	External Interactor (0.2 Application without E2E Protection)	External interactor gets input from process.	X					
90		External interactor passes input to process.	X					
91	External Interactor (8.1 External user)	External interactor gets input from process.						
92		External interactor passes input to process.						
93	External Interactor (8.2 ECU)	External interactor gets input from process.						
94		External interactor passes input to process.						
95	Data flow (Request/Receive data)	Crosses machine boundary.				X		

Table B.1: STRIDE-per-Interaction table: Threat Applicability (continued)

#	ELEMENT	INTERACTION	S	T	R	I	D	E
96	Data flow (Commands/Responses)	Crosses machine boundary.		X		X	X	

Table B.2: STRIDE-per-Interaction Threats

#	ELEMENT	INTERACTION	S	T	R	I	D	E
1	Process (7.1 CAN)	Process sends output to external interactor (8.1 External User)						
2		Process receives input to external interactor (8.1 External User)		Dataflow is tampered by an attacker which lead to information disclosure by CAN.	CAN denies getting data from External User.		CAN crashes/stop due to External User interaction.	CAN impersonate External User and use its privilege.
3		Process sends output to external interactor (8.2 ECU)						

Table B.2: STRIDE-per-Interaction Threats (continued)

#	ELEMENT	INTERACTION	S	T	R	I	D	E
4		Process receives input to external interactor (8.2 ECU)		Dataflow is tampered by an attacker which lead to information disclosure by CAN.	CAN denies getting data from ECU.		CAN crashes/ stops due to ECU interaction.	CAN impersonate ECU and use its privilege.
5		Process has outbound data flow to datastore (7.2 HW buffers)						
6		Process has inbound data flow from datastore (7.2 HW buffers)						
7		Process has sends output to another process (5.1 CANIF)						Canif is impersonate CAN and use its privilege.

Table B.2: STRIDE-per-Interaction Threats (continued)

#	ELEMENT	INTERACTION	S	T	R	I	D	E
8		Process has inbound data flow from a process (5.1 CANIF)						CAN is impersonate Canif and use its privilege.
9	Data store (7.2 HW Buffers)	Process has outbound data flow to datastore (7.2 HW buffers)					HW buffer cannot be written to (deadlock/timeou	
10		Process has inbound data flow from datastore (7.2 HW buffers)				HW buffers reveals information.		
11	Process (5.1 CANIF)	Process has sends output to another process (7.1 CAN)						CAN impersonate Canif and use its privilege.

Table B.2: STRIDE-per-Interaction Threats (continued)

#	ELEMENT	INTERACTION	S	T	R	I	D	E
12		Process has sends output to another process (3.1 PduR)						PduR impersonate CanIf and use its privilege.
13		Process has inbound data flow from a process (7.1 CAN)						Canif impersonate CAN and use its privilege.
14		Process has inbound data flow from a process(3.1 PduR)						Canif impersonate PduR and use its privilege.
15		Process has outbound data flow to datastore (5.2 Transmit buffer)						
16		Process has inbound data flow from datastore (5.2 Transmit buffer)						

Table B.2: STRIDE-per-Interaction Threats (continued)

#	ELEMENT	INTERACTION	S	T	R	I	D	E
17	Data store (5.2 TransmitBuffer)	Process has outbound data flow to datastore (5.2 Transmit buffer)					Transmit buffer cannot be written to (deadlock/timeout).	
18		Process has inbound data flow from datastore (5.2 Transmit buffer)				Transmit buffer reveals information.		
19	Process (3.1 PduR)	Process has sends output to another process (5.1 CANIF)						Canif impersonate PduR and use its privilege.
20		Process has sends output to another process (4.1 SecOC)						SecOC impersonate PduR and use its privilege.

Table B.2: STRIDE-per-Interaction Threats (continued)

#	ELEMENT	INTERACTION	S	T	R	I	D	E
21		Process has sends output to another process (2.1 COM)						COM impersonate PduR and use its privilege.
22		Process has inbound data flow from a process(5.1 CanIf)						PduR impersonate CanIf and use its privilege.
23		Process has inbound data flow from a process (4.1 SecOC)				The contents can be revealed in case of weak authentication (easily guessable credentials)		PduR impersonate SecOC and use its privilege.

Table B.2: STRIDE-per-Interaction Threats (continued)

#	ELEMENT	INTERACTION	S	T	R	I	D	E
24		Process has inbound data flow from a process(2.1 COM)						PduR impersonate COM and use its privilege.
25		Process has outbound data flow to datastore (3.2 I-PDU buffer)	I-PDU buffer is spoofed, and PduR writes in wrong place.					
26		Process has inbound data flow from datastore (3.2 I-PDU buffer)	I-PDU buffer is spoofed, wrong data is written into the buffer.					
27	Data store (3.2 I-PDU Buffer)	Process has outbound data flow to datastore (3.2 I-PDU buffer)					I-PDU buffer cannot be written to (deadlock/timeou	

Table B.2: STRIDE-per-Interaction Threats (continued)

#	ELEMENT	INTERACTION	S	T	R	I	D	E
28		Process has inbound data flow from datastore (3.2 I-PDU buffer)				I-PDU buffer reveals information.		
29	Process (4.1 SecOC)	Process has sends output to another process (3.1 PduR)				The contents can be revealed in case of weak authentication (easily guessable credentials)		PduR impersonate SecOC and use its privilege.

Table B.2: STRIDE-per-Interaction Threats (continued)

#	ELEMENT	INTERACTION	S	T	R	I	D	E
30		Process has sends output to another process (6.1 CAL)				The contents can be revealed in case of weak authentication (easily guessable credentials)		CAL can impersonate SecOC and use its privileges.
31		Process has inbound data flow from a process(3.1 PduR)						SecOC impersonate PduR and use its privilege.
32		Process has inbound data flow from a process (6.1 CAL)						SecOC impersonate CAL and use its privilege.

Table B.2: STRIDE-per-Interaction Threats (continued)

#	ELEMENT	INTERACTION	S	T	R	I	D	E
33		Process has outbound data flow to datastore (4.2 Input Buffer)						
34		Process has outbound data flow to datastore (4.3 Output Buffer)						
35		Process has outbound data flow to datastore (10.1 NvM)			NvM denies writing data received from SecOC.	The data from NvM to the process is sniffed by an attacker.	SecOC is corrupted by access denied to NvM.	
36		Process has inbound data flow from datastore (4.2 Input Buffer)						
37		Process has inbound data flow from datastore (4.3 Output Buffer)						

Table B.2: STRIDE-per-Interaction Threats (continued)

#	ELEMENT	INTERACTION	S	T	R	I	D	E
38		Process has inbound data flow from datastore (10.1 NvM)			SecOC denies receiving data from NvM.	The content of NvM to SecOC is revealed because of improper data protection.	SecOC is corrupted by access denied to NvM.	NvM is remotely executing code for SecOC.
39	Data store (10.1 NvM)	Process has inbound data flow from datastore (10.1 NvM)		NvM is corrupted.				
40		Process has outbound data flow to datastore (10.1 NvM)				NvM reveals information.		
41	Data store (4.2 Input Buffer)	Process has outbound data flow to datastore (4.2 Input Buffer)					Input Buffer cannot be written to (deadlock/timeou	

Table B.2: STRIDE-per-Interaction Threats (continued)

#	ELEMENT	INTERACTION	S	T	R	I	D	E
42		Process has inbound data flow from datastore (4.2 Input Buffer)				Input Buffer reveals information.		
43	Data store (4.3 Output Buffer)	Process has outbound data flow to datastore (4.3 Output Buffer)					Output Buffer cannot be written to (deadlock/timeou	
44		Process has inbound data flow from datastore (4.3 Output Buffer)				Output Buffer reveals information.		
45	Process (6.1 CAL)	Process has sends output to another process (6.2 CPL)						CPL can impersonate CAL and use its privileges.

Table B.2: STRIDE-per-Interaction Threats (continued)

#	ELEMENT	INTERACTION	S	T	R	I	D	E
46		Process has sends output to another process (4.1 SecOC)						SecOC impersonate CAL and use its privileges.
47		Process has inbound data flow from a process(6.2 CPL)				The contents can be revealed in case of weak authentication (easily guessable credentials)		CAL can impersonate CPL and use its privileges

Table B.2: STRIDE-per-Interaction Threats (continued)

#	ELEMENT	INTERACTION	S	T	R	I	D	E
48		Process has inbound data flow from a process (4.1 SecOC)				The contents can be revealed in case of weak authentication (easily guessable credentials)		SecOC impersonate CAL and use its privileges.
49		Process has outbound data flow to datastore (6.3 CalBuffer)		CalBuffer is spoofed and CAL writes in wrong place.			Credentials stored in CalBuffer is stolen because of weak credential storage.	

Table B.2: STRIDE-per-Interaction Threats (continued)

#	ELEMENT	INTERACTION	S	T	R	I	D	E
50		Process has inbound data flow from datastore (6.3 CalBuffer)		Incorrect data is delivered to CAL from spoofed CalBuffer.				
51	Data store (6.3 Cal Buffer)	Process has outbound data flow to datastore (6.3 CalBuffer)					Calbuffer cannot be written to (deadlock/timeou	
52		Process has inbound data flow from datastore (6.3 CalBuffer)				CalBuffer reveals information.		

Table B.2: STRIDE-per-Interaction Threats (continued)

#	ELEMENT	INTERACTION	S	T	R	I	D	E
53	Process (6.2 CPL)	Process has sends output to another process (6.1 CAL)				The contents can be revealed in case of weak authentication (easily guessable credentials)		CAL can impersonate CPL and use its privileges.
54		Process has inbound data flow from a process(6.1 CAL)						CPL impersonate CAL and use its privileges.
55		Process has outbound data flow to datastore (6.3 CalBuffer)				Credentials stored in CalBuffer is stolen because of weak credential storage.	Calbuffer cannot be written to (deadlock/timeout)	

Table B.2: STRIDE-per-Interaction Threats (continued)

#	ELEMENT	INTERACTION	S	T	R	I	D	E
56		Process has inbound data flow from datastore (6.3 CalBuffer)				CPL gets to read the information from CalBuffer it's not authorized to get.		
57	Process (2.1 COM)	Process has outbound data flow to datastore (2.2 ShadowBuffer)		ShadowBuffer is spoofed and COM writes in wrong place.				
58		Process has inbound data flow from datastore (2.2 ShadowBuffer)		Incorrect data is delivered to COM from spoofed ShadowBuffer.				

Table B.2: STRIDE-per-Interaction Threats (continued)

#	ELEMENT	INTERACTION	S	T	R	I	D	E
59		Process has outbound data flow to datastore (2.3 I-PDU Buffer)		I-PDU buffer is spoofed, wrong data is written into the buffer.				
60		Process has inbound data flow from datastore (2.3 I-PDU Buffer)		Incorrect data is delivered to COM from spoofed I-PDU Buffer.				
61		Process has outbound data flow to datastore (2.4 L-PDU Buffer)		L-PDU Buffer is spoofed and COM writes in wrong place.				

Table B.2: STRIDE-per-Interaction Threats (continued)

#	ELEMENT	INTERACTION	S	T	R	I	D	E
62		Process has inbound data flow from datastore (2.4 L-PDU Buffer)		Incorrect data is delivered to COM from spoofed L-PDU Buffer.				
63		Process has sends output to another process (1.1 RTE)						RTE impersonate COM and use its privileges.
64		Process has sends output to another process (3.1 PduR)						PduR impersonate COM and use its privileges.
65		Process has inbound data flow from a process(1.1 RTE)						COM impersonate RTE and use its privileges.

Table B.2: STRIDE-per-Interaction Threats (continued)

#	ELEMENT	INTERACTION	S	T	R	I	D	E
66		Process has inbound data flow from a process (3.1 PduR)						COM impersonate PduR and use its privileges.
67		Process has outbound data flow to datastore (2.5 Transmit Buffer)		Transmit buffer is spoofed, and COM writes to the wrong place.				
68		Process has inbound data flow from datastore (2.5 Transmit Buffer)		Incorrect data is delivered to COM from spoofed Transmit Buffer.				
69	Data store (2.5 TransmitBuffer)	Process has outbound data flow to datastore (2.5 Transmit Buffer)					Transmit Buffer cannot be written to (deadlock/timeou	

Table B.2: STRIDE-per-Interaction Threats (continued)

#	ELEMENT	INTERACTION	S	T	R	I	D	E
70		Process has inbound data flow from datastore (2.5 Transmit Buffer)				Transmit Buffer reveals information		
71	Data store (2.2 Shadow Buffers)	Process has outbound data flow to datastore (2.2 ShadowBuffer)					ShadowBuf cannot be written to (deadlock/timeou	
72		Process has inbound data flow from datastore (2.2 Shadow-Buffer)				Shad-owBuffer reveals information		
73	Data store (2.4 L-PDU Buffer)	Process has outbound data flow to datastore (2.4 L-PDU Buffer)					L-PDU Buffer cannot be written to (deadlock/timeou	

Table B.2: STRIDE-per-Interaction Threats (continued)

#	ELEMENT	INTERACTION	S	T	R	I	D	E
74		Process has inbound data flow from datastore (2.4 L-PDU Buffer)				L-PDU buffer reveals information		
75	Data store (2.3 I-PDU Buffer)	Process has outbound data flow to datastore (2.3 I-PDU Buffer)					I-PDU Buffer cannot be written to (deadlock/timeou	
76		Process has inbound data flow from datastore (2.3 I-PDU Buffer)				I-PDU buffer reveals information		
77	Process (1.1 RTE)	Process receives input to external interactor (0.1 E2E with protection)	Information disclosure by E2E with protection when RTE is spoofed.	Data flow get unprotected signals tampered with by an attacker.	RTE disclaims receiving data from "E2E with protection".		RTE crashes/stodue to E2E with protection interaction.	RTE impersonate "E2E without protection" and use its privileges.

Table B.2: STRIDE-per-Interaction Threats (continued)

#	ELEMENT	INTERACTION	S	T	R	I	D	E
78		Process receives input to external interactor (0.2 E2E without protection)	Information disclosure by E2E without protection when RTE is spoofed.	Dataflow get unprotected signals that is tampered with by an attacker.	RTE disclaims receiving data from "E2E without protection".	Unprotected signals is sniffed by an attacker.	RTE is interrupted of an "external agent". RTE crashes and interrupt the data flow.	"E2E without protection" passes data that allow it to change flow of execution of RTE. RTE also impersonate "E2E without protection" and use its privileges.
79		Process sends output to external interactor (0.1 E2E with protection)			"E2E with protection" disclaims receving data from RTE.		RTE is interrupted of an "external agent".	

Table B.2: STRIDE-per-Interaction Threats (continued)

#	ELEMENT	INTERACTION	S	T	R	I	D	E
80		Process sends output to external interactor (0.2 E2E without protection)					RTE is interrupted of an "external agent".	COM impersonate RTE and use its privileges.
81		Process has sends output to another process (2.1 COM)						RTE impersonate COM and use its privileges.
82		Process has inbound data flow from a process(2.1 COM)						
83		Process has outbound data flow to datastore (2.5 Transform Buffer)		Transform buffer is spoofed and RTE writes in wrong place.				

Table B.2: STRIDE-per-Interaction Threats (continued)

#	ELEMENT	INTERACTION	S	T	R	I	D	E
84		Process has inbound data flow from datastore (2.5 Transform Buffer)		Incorrect data is delivered to RTE from spoofed Transform Buffer.				
85	Data store (1.2 Transform Buffer)	Process has outbound data flow to datastore (2.5 Transform Buffer)					Transform Buffer cannot be written to (deadlock/timeou	
86		Process has inbound data flow from datastore (2.5 Transform Buffer)				Transform Buffer reveals information.		

Table B.2: STRIDE-per-Interaction Threats (continued)

#	ELEMENT	INTERACTION	S	T	R	I	D	E
87	External Interactor (0.1 Application with E2E Protection)	External interactor gets input from process.	Application with E2E Protection is confused about the identity with RTE.					
88		External interactor passes input to process.	RTE is confused about the identity of the Application with E2E Protection.					

Table B.2: STRIDE-per-Interaction Threats (continued)

#	ELEMENT	INTERACTION	S	T	R	I	D	E
89	External Interactor (0.2 Application without E2E Protection)	External interactor gets input from process.	RTE is confused about the identity of the Application without E2E Protection.					
90		External interactor passes input to process.		Application without E2E Protection is confused about the identity with RTE.				
91	External Interactor (8.1 External user)	External interactor gets input from process.						
92		External interactor passes input to process.						

Table B.2: STRIDE-per-Interaction Threats (continued)

#	ELEMENT	INTERACTION	S	T	R	I	D	E
93	External Interactor (8.2 ECU)	External interactor gets input from process.						
94		External interactor passes input to process.						
95	Data flow (Request/ Receive data)	Crosses machine boundary.				The contents of the data flow are sniffed on the wire.		
96	Data flow (Commands/ Responses)	Crosses machine boundary.		Data flow is tampered by an attacker.		The contents of the data flow are sniffed on the wire.	The data flow is interrupted by an external entity.	