Technical University Ingolstadt
Faculty of Electrical Engineering and Computer Science

BACHELOR THESIS

Security Evaluation for the Real-time Operating System VxWorks 7 for Avionic Systems

by

Kevin Klaus Gomez Buquerin

Faculty of Electrical Engineering and Computer Science

Supervisors of the bachelor thesis:	Prof. Dr.-Ing. Hans-Joachim Hof
	Prof. Dr.-Ing. Ernst-Heinrich Göldner
	Dr. Andreas Schweiger

Begin date:	October 09, 2017
Submission date:	January 11, 2018
Study programm:	Aviation and Automotive Computer Engineering

Ingolstadt, May 3, 2018

Declaration

I hereby declare that this thesis is my own work, that I have not presented it elsewhere for examination purposes and that I have not used any sources or aids other than those stated. I have marked verbatim and indirect quotations as such.

Ingolstadt, 11. January 2018

Kevin Klaus, Gomez Buquerin

First Name, Surname

Abstract

In previous years, the threat landscape for avionic systems, especially with a military background, became more dangerous. Additional, the complexity for aircraft systems is increasing as well as the number of threats, attacks, and vulnerabilities against computer systems. From this it follows, core units such as real-time operating systems need to be secure to provide shelter against possible cyber security threats.

This thesis discusses and tests avionic relevant categories separated into common threats, attacks, and vulnerabilities for the real-time operating system VxWorks 7. The security of the RTOS is evaluated on a simulated target using varying security configurations of the real-time operating system. Possible security technologies and mechanisms provided by VxWorks are viewed and evaluated, regarding their reaction against threats, attacks, and vulnerabilities.

Issues appeared with basic attacks and vulnerabilities such as buffer overflows or string vulnerabilities, where negative impact on the system is noticeable. Good security practices are observed in security areas such as cryptography and privilege management.

The study reveals security issues as well as good protective technologies and mechanisms for the real-time operating system VxWorks 7 in the area of avionic systems.

Furthermore, additional security technologies and mechanism, which can provide further protection against common attacks, threats, and vulnerabilities, are introduced.

Acknowledgements

I want to thank Prof. Dr.-Ing. Hans-Joachim Hof for the incredible and enriching talks about this topic, the support during challenging tasks, and the extensive mentoring.

Furthermore, I want to thank Dr. Andreas Schweiger from Airbus Defence and Space for the amazingly good conversations and discussions about this thesis as well as the granted freedom to structure and approach this topic.

Special thanks to Stefan Harwarth and Rainhard Kain from *Wind River* for making it possible to use the products in my thesis for educational purposes.

Thank you to my family and my girlfriend for supporting me throughout my bachelor.

Glossary

Attack Vector	Used path by an attacker to gain access to a computer system.
Brute-force Attack	An attacker attempts as many requests (e.g. password tries) as possible. The better the password, the longer the brute-force takes.
Buffer Overflow	Overwrite of data, which lies beyond the intended buffer size.
Buffer Underflow	Overwrite of data, which underflows the intended buffer size.
Code Injection	Take advantage of a vulnerability, which grants the ability to introduce source code into an application.
Data Leakage	Unintentional lose and leakage of data within a computer system.
Denial of Service	Make a computer system unavailable for a certain amount of time. Causes a downtime of the system.
Eavesdropping	Listening to a conversation without alerting the communicating entities.
Exploit	An application, which takes advantage of a vulnerability within a computer system to obtain malicious intentions.
Malware	Malicious software with the goal to perform harmful actions.

Man-in-the-middle	An attacker is able to position himself between two entities to gather and modify data, which is transmitted between both.
Social Engineering	Forcing an expected reaction by an other individual using targeted manipulation.
SQL Injection	Cause an unintended SQL execution using crafted SQL queries.
Trojan	An unnoticed application within a computer system, which gathers information about the target system.
Virus	Malicious software, which tries to replicate itself and spread even more.
Worm	Used to spread malicious code from computer to computer, for example over the network.

Abbreviations

API	Application Programming Interface
ARM	Instruction set architecture for processors
ASLR	Address Space Layout Randomization
C	Programming language C defined in ISO/IEC 9899:2011 [1]
CLI	Command Line Interface
CPU	Central Processing Unit
CVE	Common Vulnerabilities and Exposures
C++	Programming language C++ defined in ISO/IEC 14882:2014 [2]
DAL	Development Assurance Level
DoS	Denial of Service
I/O	Input/Output
IP	Internet Protocol
OS	Operating System
QSP	Wind Rivers Quick-Start Platform
RTOS	Real-time Operating System
VIP	VxWorks Image Project
VSB	VxWorks Source Build Projects

List of Figures

1 Concept for the Future Aircraft Systems 13

2 Viewed Security Areas 15
3 Vulnerability, Threat and Attack 17
4 Wind River's Simics 20

5 Security Evaluation Approach 23

6 Workbench Format String Vulnerability Warning 29

7 Simulation Setup 40
8 VxWorks 7 Kernel Image 45
9 Simics Control QSP PowerPC Board 48
10 QSP PowerPC Board Architecture 48

11 VxWorks 7 Login Policies 50
12 VxWorks 7 Password Policies 51
13 Initial VxWorks 7 User *vxworks* 52
14 Size of the string "A" on the VxWorks 7 machine 57
15 Buffer Overflow Test – 44 character copy 58
16 Buffer Overflow Test – 45 character copy 59
17 Buffer Overflow Test – File system ejected 60
18 Format String Test – Stack Leak 62
19 Stack to the defined time stamp 63
20 Format String Test – Leak local data 64
21 Memory Management Unit Output 65
22 Generation of a 2048 bit RSA Private Key 68
23 VxWorks Cryptography Configuration for VSB 69
24 Symbol Table System Function 72

List of Tables

6.1 Security Evaluation for VxWorks Version 7 39

7.1 Final Security Evaluation Matrix 76

Contents

1 Introduction **12**

2 Background **14**
2.1 Relevant Information Security Aspects 14
2.2 Threat, Vulnerability, and Attack 16
2.3 Real-time Operating System VxWorks 18
2.4 Workbench and Simics 19

3 Related Work **21**

4 Evaluation Approach **22**

5 Analysis **24**
5.1 Developers security awareness 24
5.2 Secure Software Development in C and C++ 25
5.2.1 Strings and User Input 27
5.2.2 Pointer Subterfuge 29
5.2.3 Integers 30
5.2.4 Formatted Output (Format Strings) 32
5.2.5 File I/O 34
5.3 Specific Characteristics for Aircraft Systems 34
5.4 Score Definition and Evaluation Structure 35

6 Design and Implementation **36**
6.1 Representation Method 36
6.2 Relevant Threats, Vulnerabilities, and Attacks 37
6.3 Infrastructure 40
6.4 Workbench configuration 41
6.5 Simics configuration 47

7 Evaluation **49**
7.1 E1 – Improper Authentication (*CWE-287*) 49
7.2 E2 – Privilege Management (*CWE-264/266/269*) 53
7.3 E3 – Malware Protection 53
7.4 E4 – Buffer Overflow (*CWE-121/122/123*) 55
7.5 E5 – String Vulnerabilities (*CWE-133/134*) 60
7.6 E6 – Secure Cryptography Algorithm 67
7.7 E7 – Storage of Sensitive Data 69
7.8 E8 – Command Injection (*CWE-77/78*) 71
7.9 E9 – Integrity check of external Software 72
7.10 Summary 74

8 Conclusion and Outlook **77**

Chapter 1

Introduction

According to the *Bundesamt für Sicherheit in der Informationstechnik* (BSI) of Germany, the total number of critical software vulnerabilities in common software products[1] increased and partly doubled from Q2/15 - Q1/16 to Q2/16 - Q1/17 [3].

This increase is also visible in operating systems (OS). Due to the architectural position, which is between the application/user layer and the hardware layer, control over the OS can lead to full control over the computer system. From 2016 to 2017, the number of known vulnerabilities for the major operating system vendors (*Apple*, *Microsoft*, and *Linux*) has increased. The number for *Apple's* Mac OS X has risen from 215 up to 236 [4], for *Microsoft's* Windows 10 from 172 to 255 [5], and for the community developed Linux Kernel from 217 to 407 [6]. This exhibits a clear trend for weaknesses in operating systems.

Real-time operatings systems (RTOS) are a core unit of common avionic systems and connect the hardware layer with the application layer. State of the art avionic systems became more complex over the past years. The percent of functionality provided by software has increased from 8% in 1960 up to 80% in 2000 for U.S. fighter jets [7]. New mechanisms and technologies expand the scope of possible attacks and increases the number of attack vectors[2].

This increasing complexity becomes clearly visible in the concept of future aircraft systems as illustrated in Figure 1. The leading and manned aircraft communicates with the ground station, satellites, and other aircraft

[1] *Adobe* Reader, *Adobe* Flash, *Apple* OS X, *Google* Chrome, Linux Kernel, *Mozilla* Firefox, *Oracle* Java/JRE, *Microsoft* Windows, *Microsoft* Internet Explorer and *Microsoft*

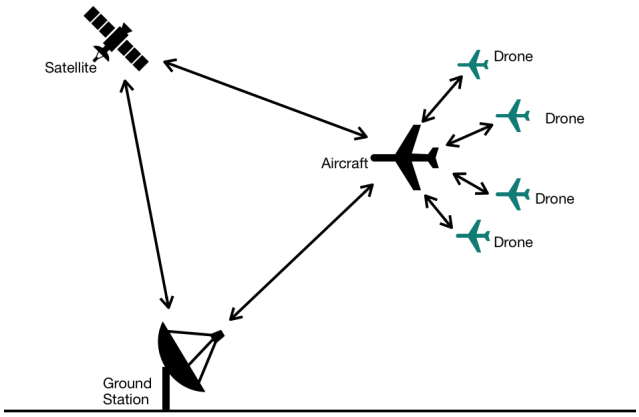

Figure 1: Concept for the Future Aircraft Systems

or drones. As a consequence, the connectivity of the overall system increases dramatically, which results in more possible attack vectors.

Avionic systems are not always secure. The security researcher Hugo Teso was able to exploit vulnerabilities in the flight-control-system as well as the communication system for state of the art aircraft systems [8]. This gave him access to valuable mission data and other resources. In his released work he identifies clear security issues in avionic communication systems.

This leads to the question if state of the art RTOS provide a sufficient security foundation for avionic systems.

The thesis is structured as follows: Chapter 2 gives a short overview of the essentials to understand the argumentation and implications for the rest of the thesis. After the fundamentals follows a list of related work, which already discusses security for VxWorks in Chapter 3. Chapter 4 illustrates the approach, which is chosen to evaluate VxWorks 7 in the field of security. Following from this chapter, is the analysis of the research question and how it is going to be answered in Chapter 5. The chosen design and implementation to perform a security evaluation of VxWorks 7 is shown in Chapter 6. The security evaluation itself, where all relevant topics are assessed, is located in Chapter 7. Chapter 8 summarizes the results and gives and outlook for further work.

Office

[2]Path which an attacker takes to gain access to a computer system.

Chapter 2

Background

To understand the main body of the thesis, prerequisite and contextual information is necessary. This chapter offers an overview of several important topics, which are relevant in this bachelor thesis.

2.1 Relevant Information Security Aspects

The U.S. code law defines information security in the following way: "*The term "information security" means protecting information and information systems from unauthorized access, use, disclosure, disruption, modification, or destruction.*" [9]. The definition displays, that information security is a broad area in computer science. To protect computer systems for the enumerated terms, protective security mechanisms are necessary.

Several vulnerabilities, attacks, technologies, and mechanisms are viewed in this thesis as depicted in Figure 2.

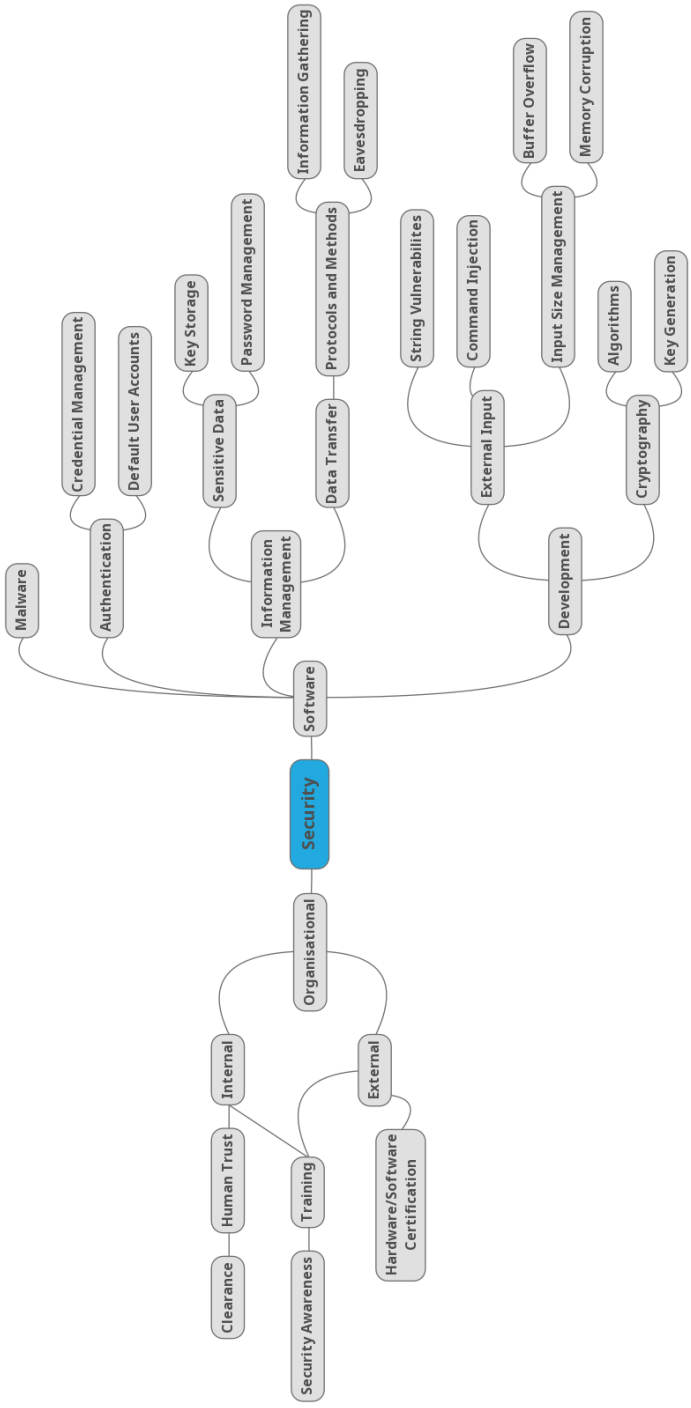

Figure 2: Viewed Security Areas

The displayed scope shows the most relevant areas for avionic systems:

- Organisational
  - → Internal – Threats from internal entities including personnel, which can be prevented by humans trust, clearances, training, and a sufficient security awareness.
  - → External – Threats from external entities including off the shelf hardware and software. Hardware and software certifications as well as training and security awareness can harden a system in reference to this topic.
- Software based
  - → Malware – Increasing complexity and internet of things (IoT) similarities in avionic systems.
  - → Authentication – Secure access control between aircraft, ground station, satellites, and other entities. This topic is separated into creditable management and the issue with default user accounts.
  - → Information management – Secure storage and handling of sensitive data with topics such as the storage of passwords or keys, the used protocols and methods, which can be attacked by information gathering and eavesdropping.
  - → Development – Secure software development for aerospace software. Development can be separated into external input, which can lead to string vulnerabilities, command injections, and input size management, which can further be subdivided into buffer overflows and memory corruption.

2.2 Threat, Vulnerability, and Attack

The three terms threat, vulnerability, and attack are mentioned multiple times in the course of this thesis. Figure 3 displays the dependence between the terms.

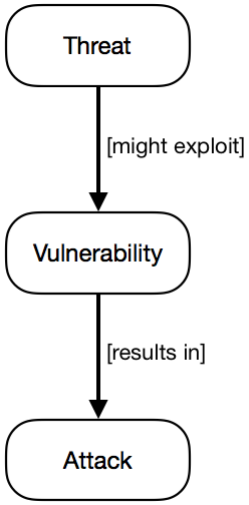

Figure 3: Vulnerability, Threat and Attack

A threat is a potential security flaw, which affects the integrity of a computer system. It can result in the exploitation of a vulnerability (if one is present) within the system [10]. Eckert presents five major threats to computer systems [11]:

- Act of nature beyond control, which is not viewed in this thesis because of the small impact a developer has regarding this topic. Furthermore, the RTOS relies on properly function hardware, which could not be given, if the nature would influence the hardware significantly.
- Carelessness is the result of poor written software and covered in security mistakes by programmers (Chapter 5).
- Intention is present, if the developer is not trustworthy. Furthermore, attackers have the intention to target the system using vulnerabilities and attacks. This topic is covered in developers security awareness (Chapter 5.1) as well as in the security evaluation (Chapter 7).
- Technical failure is not considered in this thesis because a developer has no impact on this topic and the reaction of hardware failures of the overall avionic system is not viewed.
- Organisational shortcomings are covered in developers security awareness (Chapter 5.1) and is the result of poor assignment of security

clearances.

Vulnerabilities reduce the overall security of a computer system. Examples are Buffer Overflows, Injection Vulnerabilities including Code and Command Injections, as well as Data Leakage Vulnerabilities, which are all explained in the glossary. If an attacker is able to detect a vulnerability, he is usually able to attack the system.

An attack against a computer system is mostly the result of a vulnerability. Typically an attack occurs in the form of an exploit. Exploits can lead to the gain of unintended privileges and informations[3], compromise of the system, the altering of data, or a crash. Examples are Man-in-the-middle attacks, Eavesdropping, SQL Injection Attacks, and Denial of Service (DoS) Attacks. All attacks are explained in the glossary.

2.3 Real-time Operating System VxWorks

An aircraft is limited in space to carry hardware devices such as computers. This leads in the need of rather small hardware elements. Those computers are called Electronic Control Units (ECU). An ECU is an embedded device with limited resources. Operating systems for desktop computers such as Windows, Linux, or macOS are not able to perform smoothly with limited resources or fulfil the real-time requirements. They are not usable on such a device due to the constant display of user interfaces, frequent appropriation of multiple networking or connectivity technologies, and similar. Instead, an ECU uses a real-time operating system such as VxWorks.

Kevin D. Morgan characterizes RTOS with the following attributes: determinism[4], responsiveness[5], user control[6], reliability[7], and fail-safe operation[8] [12][13].

[3]Such as user credentials, military mission data, and similar.

[4]The performance of fixed operations at predetermined times, the predictability for results, and the characteristic that the outcome is repeatable.

[5]The time how long it takes for the system to acknowledge an interrupt.

[6]The determination and monitoring for users permissions over tasks, objects and resources.

[7]The need to respond to events in real time and the characteristic that all services are provided as required. If this is not guaranteed, it might result in damage of the system and environment.

[8]The fail of the system leads to preservation of as much data and services as possible.

VxWorks is a RTOS developed by *Wind River*[9] [14]. The RTOS mainly supports C, C++, and Java as programming languages for software development. This thesis discusses VxWorks on the newest Version 7. The reason to choose VxWorks 7 over the already used, certified, and aerospace specific VxWorks 653[10] is to select a RTOS with dedicated security features to protect future avionic systems against security threats.

2.4 Workbench and Simics

Wind River's Workbench Version 4 is an application, which grants the ability to deploy VxWorks, *Wind River* Linux, and the Diab Compiler[11]. The application uses the integrated development enviroment (IDE) "Eclipse" and provides several components including support for File Transfer Protocol (FTP) and Trivial File Transfer Protocol (TFTP) servers, configuration/build tools, debug and analysis tools, language support for the programming language Ada[12] (while using the AdaCore [17], which allows compilation of Ada source code on a VxWorks 7 system), and more. The Workbench configuration is presented in Section 6.4. [18]

Wind River's Simics is used to simulate multiple target systems, which are often used by aerospace companies for development purposes. As shown in Figure 4, the application consists of two windows: Simics Control to select targets and start or pause the simulation, and the Simics Command Line Interface (CLI) to manage the simulated boards including Ethernet connections and debug functionalities. If a target is selected, the application creates a Serial Console window, which enables interaction with the simulated board running the chosen OS. Figure 4 displays the Serial Console[13] window simulating VxWorks 7 on a Quick-Start Platform (QSP) with *PowerPC* architecture. Details to the Simics configuration and the QSP are presented in Chapter 6.5.

[9] *Wind River* is specialized in developing software for embedded devices. VxWorks is one of their core products. With over 1.5 billion deployed devices VxWorks is the world's leading real-time operating system.

[10] RTOS with an ARINC 653 architecture, which is an internationally accepted specification for aerospace systems [15].

[11] *Wind River's* own compiler for safty critical systems.

[12] Objected-oriented programming language defined in ISO/IEC 8652:2012 [16].

[13] Allows interaction with the simulated operating system (here VxWorks 7).

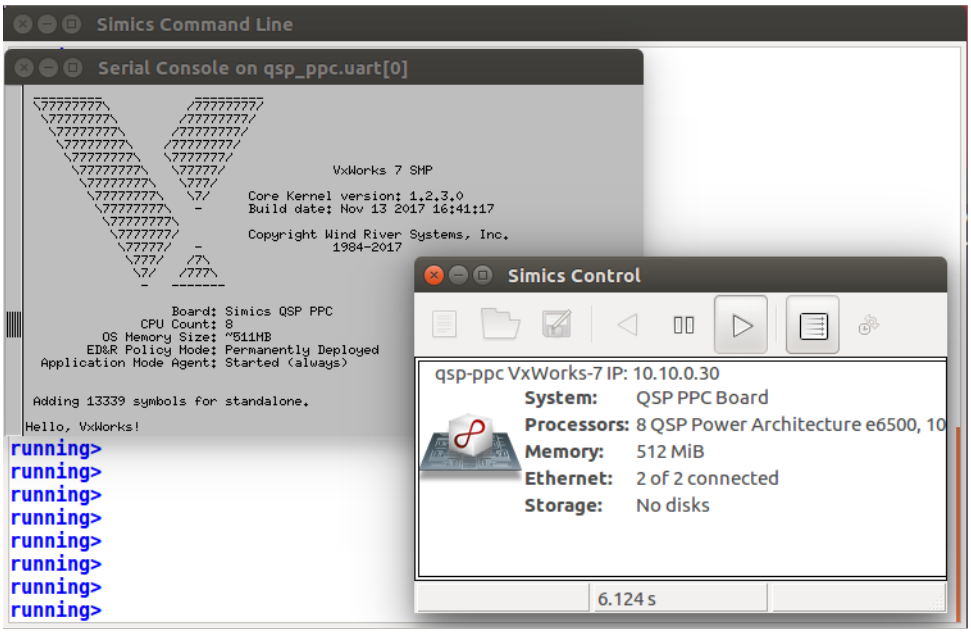

Figure 4: Wind River's Simics

Chapter 3

Related Work

Since VxWorks is one of the most popular RTOS [19], security analysis and penetration tests has already been done by security researchers.

Aditya K. Sood was concerned with VxWorks 5.x and found several security vulnerabilities [20]. The researcher acknowledged that the stack protection of VxWorks was not stringent enough, since the default stack guard was only 4 kilobyte (KB) big[14]. Additionally, the password encryption algorithm of the viewed VxWorks version was not secure enough. Encrypted passwords were brute-force-able.

Furthermore, the software company Rapid7 looked at VxWorks 6.x and found weak password hashing within the RTOS [21]. Passwords are used to enable a login, if remote access to the device is needed. This access is provided using the File Transfer Protocol (FTP) or Telnet. The researchers were able to dump memory using the hacking tool Metasploit[15] and read passwords in clear-text representation. The passwords were stored in hashed form within the RTOS. However, unencrypted parts of the firmware image contained clear-text passwords. The researcher was able to login using the captured passwords.

Both researchers show clear security issues within VxWorks. However, the research was performed on older versions of the real-time operating system. No publicly available research for VxWorks 7 is accessible, which leads to the need of a security evaluation of the latest version. This thesis discusses and evaluates the security for the current version 7.

[14] Chapter 6.4 gives further explanation regarding kernel hardening features and the stack guard.

[15] An application, which provides several security vulnerabilities to test the robustness of a computer system

Chapter 4

Evaluation Approach

Figure 5 illustrates the methodology to perform the evaluation of the real-time operating system VxWorks 7.

The security awareness of software developers is introduced in Chapter 5.1 followed by the presentation of secure software development aspects in Chapter 5.2. Chapter 5.3 points out the avionic characteristics in order to define avionic relevant security categories, which are introduced in Chapter 6. Those are separated into multiple threats, vulnerabilities, and attacks, which are presented in Chapter 6.2. The mentioned threats, vulnerabilities, and attacks are tested on the simulated VxWorks 7 target as well as evaluated using vulnerable code or manual audits in Chapter 7. Afterwards, the reaction of VxWorks regarding the tested points is monitored and analysed. If the RTOS is able to prevent the attack from happening or prevent further damaged, a positive[16] score is given. This score itself is separated into two appropriate sub items[17]. On the other hand, if VxWorks 7 is not able to protect the system from damage, a negative[18] score is given. The corresponding scores are listed into the evaluation matrix Table 7.1 in Chapter 7.10.

[16] "*Mechanism exists*"
[17] "*Full protection*" and "*Only extenuates*"
[18] "*Mechanism does not exist*"

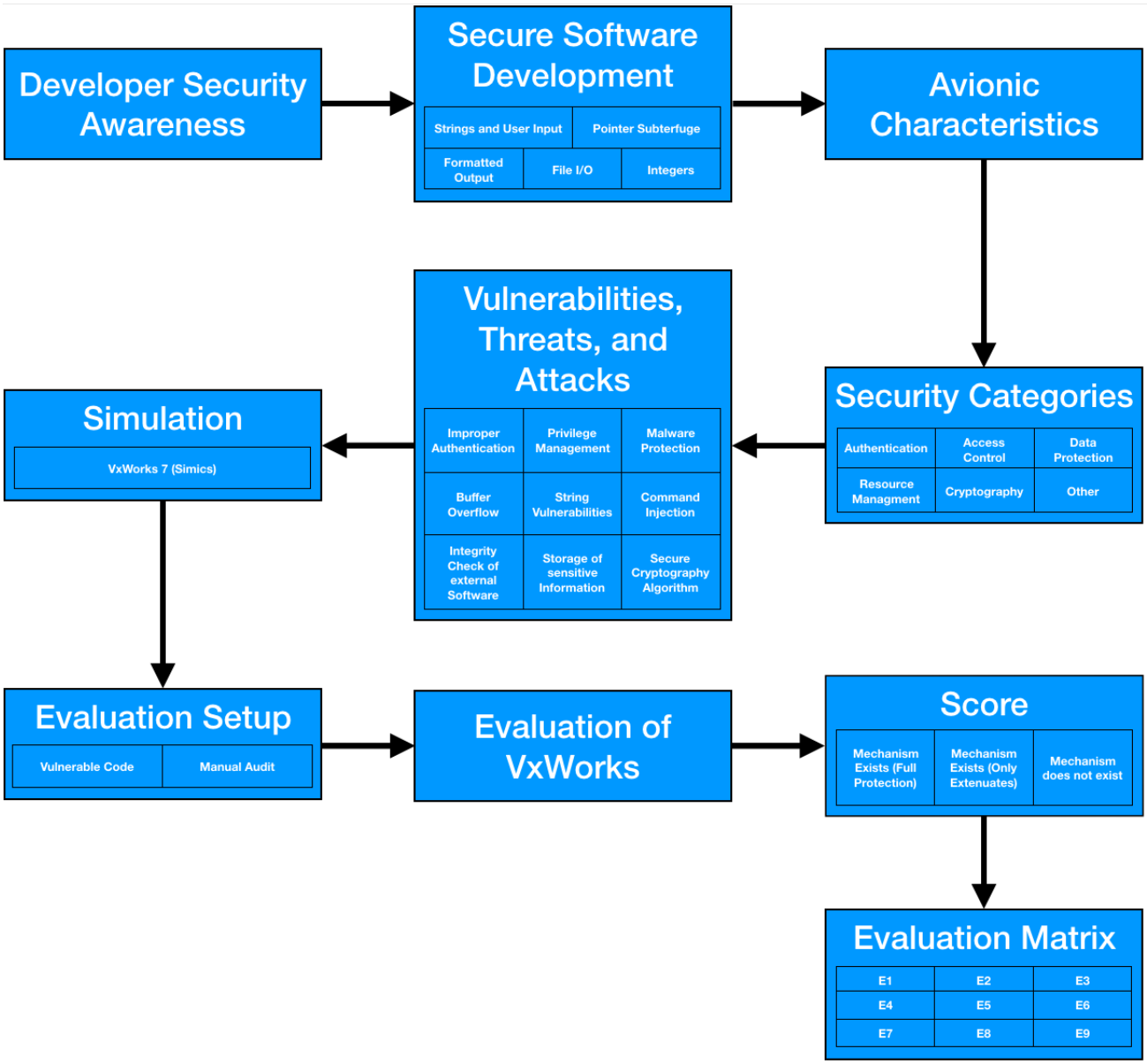

Figure 5: Security Evaluation Approach

Chapter 5

Analysis

To answer the question, if the real-time operating system VxWorks 7 offers protective technologies and methods to be usable in avionic systems, a further analysis of this topic is necessary. This chapter displays the areas for secure software development, which is necessary to protect the RTOS from unwanted security incidents as well as the characteristics, which are important for aircraft systems. Afterwards, an assessment of the evaluation parameters and the structure to evaluate VxWorks 7 is presented.

5.1 Developers security awareness

To maintain secure software development and protect the system in the first place, programmers need to be trustworthy to ensure security in the work place as well as the private life. Security awareness is important for software development departments, since programmers are able to add back doors or intentionally make software prone to vulnerabilities. This can be extremely valuable for an attacker. An assailant is able to gain this information by paying, blackmailing the individual, or through social engineering. [22]

Due to the amount of sensitive information and importance to provide secure software, military aircraft software development centres have a strong security infrastructure. This is realized using clearances for the personnel. Several talks with officials and background checks of the person are necessary to get a clearance. Furthermore, employees are only able to gain information they need to know. Clearances ensure organisational security within the software development department for an aircraft. However, a clearance does not prevent programmers from making software security errors.

The University of North Carolina at Charlotte (UNC) published a paper [23], which discusses the reasons for programmers making security errors. The researchers from the UNC interviewed multiple experienced software developers. Many of the developers had a basic understanding of software security, while differences appeared in performing practices to ensure security. The developers display the problem of deadlines, available project budget, requirements set by the customers, and the lack of knowledge.

These interviews show a clear lack of security awareness and knowledge of experienced software developers as well as issues in development departments.

5.2 Secure Software Development in C and C++

Software written in VxWorks' supported languages[19] needs to provide a sufficient software security foundation. These fundamentals result in secure software, which is lesser prone to common vulnerabilities and robust against various software conditions and inputs.

This thesis discusses the programming languages C and C++, since JSF++ [24], which is based on C and C++ is commonly used in U.S. military aricraft software development. Java is not used in military aircraft because the Java byte code is executed in the Java Virtual Machine (JVM). This results in an increase of the execution time. Furthermore, Java is not a hardware-near programming language like C, C++, or Ada. Hence, influence on hardware devices takes further steps and more time to achieve. There is a possibility to execute Java byte code directly and not using the JVM. Version 6 of the GNU Compiler Collection (GCC) compiler, called GNU Compiler for Java (GCJ) supports this feature [25]. However, this technology is not used in common aircraft software. The programming language Ada [16], which is common in military aircraft, is by default[20] not supported in VxWorks 7 and therefore not considered in this thesis.

There is a broad range of security mistakes programmers can do, which might lead to vulnerabilities and consequential exploits or attacks. If there is no focus on security during software development, the chance of applications

[19] C, C++, and Java.

[20] With the use of a technolgoy called AdaCore [17] Ada can be used in VxWorks 7. Although, this technology is not used due to licensing problems.

containing vulnerabilities and errors is higher than with a focus on security [26].

The following areas of the programming languages C and C++ commonly lead to security vulnerabilities. They are based on Robert C. Seacord's book "*Secure Coding in C and C++*" [27]:

- Strings and User Input
- Pointer Subterfuge
- Dynamic Memory Management
- Integer Security
- Formatted Output
- Concurrency
- File I/O

Dynamic memory management and concurrency are both left out. The two areas are usually not relevant in current aircraft software development and not that much used compared to the other displayed areas.

Due to security and safety reasons, dynamic memory management is currently not allowed and not used in aircraft software development. Resources are statically allocated to minimize calculation time. If the memory is allocated dynamically, the processing time is higher, and it is harder to fulfil real-time requirements in an aircraft. Furthermore, if allocated memory is not freed correctly, the memory could be insufficient. Other programs would be incapable in execution because the memory is not sufficient enough.

Concurrency defines the computation of multiple executions simultaneously using common resources. Since current aircraft does not yet support multi core processors as well as shared memory over several processors and processes, concurrency is not relevant for avionic systems. However, aircraft software development uses multiple tasks to introduce parallelism in avionic systems but without using shared memory. As a result, concurrency is not viewed in this thesis.

Once software developers consider the demonstrated mitigation techniques presented in Robert C. Seacord's mentioned book, the errors and vulnerabilities are less likely to occur. In addition, constant source code audit and software updates prevent vulnerabilities to become acquainted in the future.

5.2.1 Strings and User Input

External data including user input can be unsafe. Further, it can be costly to verify the source of external data. Therefore, it is safer to assume that all external data is unsafe and not trustworthy in the first place. Security programming errors with strings and user input can lead to the following vulnerabilities and attacks:

Stack Smashing: The OS saves local data and return addresses on the stack[21]. If source code writes past the intended declared size, it is called a stack smashing attack [28]. Such an attack can be the result of a buffer overflow vulnerability.

Buffer Overflow: Eckert defines a buffer overflow attack as a programming error, which results in the overflow of a variable with fixed length[22] [11]. Therefore, more data than intended is written to a variable without checking if the size of the variable exceeds. Usually a buffer overflow vulnerability leads to the control of one or more pointers such as the program instruction pointer. Those can be used to change the control flow of the program.

Code Execution: The attacker has the ability to execute custom code. This is extremely powerful since the attacker can choose the code he or she wants to execute. Code execution attacks can lead to the gain of root privileges, Denial of Service (DoS), ability to read or write data, and more. This attack is usually the result of a buffer overflow or command injection.

Arc Injections: An arc injection attack is also known as "*return-to-libc*". For the programming language C, *Libc* provides the standard library for Linux [29]. During "*return-to-libc*" attacks, control is transferred to code that already exists. For example in libraries or application programming interfaces (API) [27]. This is useful, if security mechanisms like the non-executable stack bit[23] (NX) is enabled, because the attacker can locate the instruction he or she wants to execute on the heap instead of using the stack[24].

[21] This is a reservered amount of memory to store this data.
[22] A buffer can be an allocated array in C (`char arr[64]`)
[23] Prevents instructions from being executed on the stack.
[24] Usually the stack is easier to access and to use compared to the heap.

Return Oriented Programming (ROP): If an attacker executes existing code from the program for unintended reasons [30], it is called ROP attack. Since software contains assembler code there are a lot existing code snippets. Those code snippets can be used to execute specific instructions, which can lead to dangerous program execution including shellcode[25].

There are multiple mitigation techniques to weaken or even prevent the mentioned vulnerabilities and errors to occur during software development, while using strings and user input. In order to provide decent security for C or C++ software, while using strings and handling user input, the use of secure functions comes in handy. Secure functions in C can be `strncpy_s()` or `strncat_s()`. Those functions are elements of the default library of the C programming language and provide an additional parameter which determines the size of the destination buffer. For example: `strncpy_s(dest, dest_size, src, count)` where `dest` is a pointer to the array to copy to, `dest_size` the maximum size of the destination buffer, `src` is a pointer to the array to copy from, and `count` the maximum number of characters which should be copied.

The development environment can offer help for software developer to write secure software. *Wind River's* Workbench provides warnings or errors for the developed project as displayed in Figure 6. If some of the source code files contain a format string vulnerability, a compiler warning is displayed during the project build process. The Workbench checks for code segments where no explicit format string is in use. For example in `printf(user_input)`, where the user input directly gets printed without checking its content or size.

[25] Usually, shellcode is byte code with the purpose of executing a shell. Nowadays it is used in more ways inlcuding breaking out of a chroot shell, creating a file, and proxying system calls. [31]

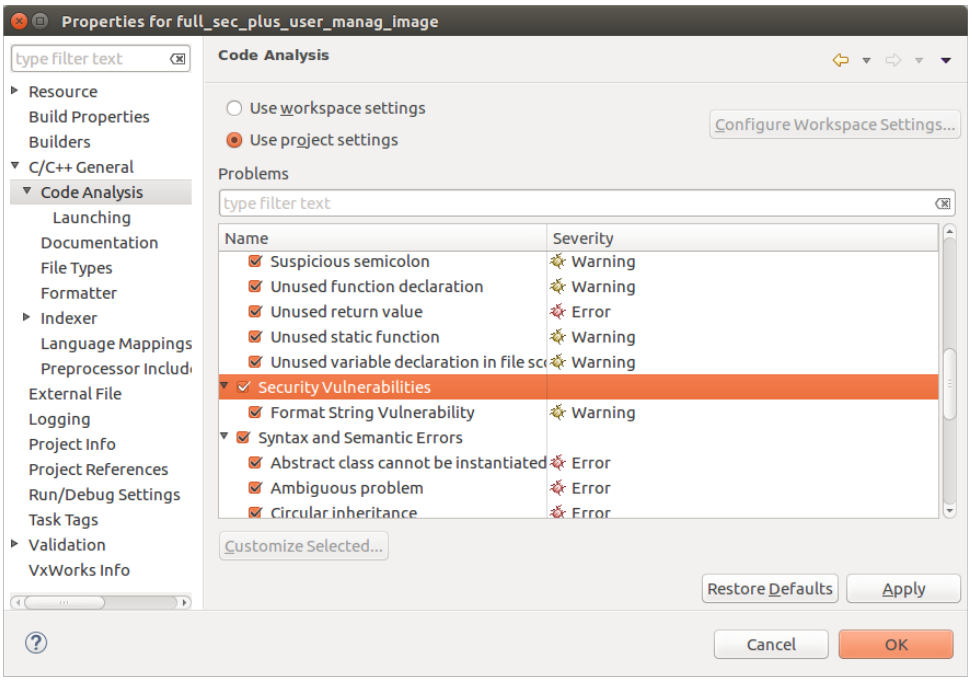

Figure 6: Workbench Format String Vulnerability Warning

5.2.2 Pointer Subterfuge

The process during a buffer overflow, which results in an overwrite of a pointer, is called pointer subterfuge. This overwrite is either a pointer to an object[26] (*pointer-to-object*) or a pointer to a function[27] (*pointer-to-function*) [27]. The overwrite of pointers can lead to code flow changes. If the execution order of instructions changes or even the attacker controls this alterations, it can cause massive damage and might result in security issues.

Attackers are mostly able to read pointers in clear text. Usually in hexadecimal representation such as "`0x23b1e990`". Therefore, an attacker is able to overwrite pointer addresses since those are legible. The encryption of pointers during compilation would complicate the ability to use pointer to redirect code execution. Merely the encrypted representation of a pointer is visible if this mechanism is used. Secure encryption algorithms and robust

[26]Points to assigned memory including the heap. This memory is allocated during the run time of the software with C functions such as `malloc()` or `calloc()`.

[27]Refers to a function which is located in a memory segment such as the stack.

key management can provide durability against pointer subterfuge vulnerabilities.

Wind River's Workbench does not provide security mechanisms or technologies to prevent any kind of pointer subterfuge attcks.

5.2.3 Integers

Integers in C and C++ have a fixed size and therefore boundary conditions. If an integer value exceed its size, it can result in an unexpected value.

Conversion and truncation errors can occur if the C/C++ developer does not validate integer sizes. Both errors are a result of incorrect or inadequate integer allocation.

Conversion errors occur, if an integer value is only checked one way. The following code displays such an error:

```
1 void foo(int var1)
2 {
3     if(var1 <= MAX_SIZE)
4         // continue processing, e.g. allocate memory using var1
5     else
6         // handle error to allocate too much memory
7 }
```

The intention of the function `foo` is to check if the variable `var1` exceeds the maximum intended size (`MAX_SIZE`). If the integer value is higher as requested, error handling takes place. The security issue here is the insufficient boundary check. The if-clause only handles errors for values higher than the maximum intended size (`var1 <= MAX_SIZE`), but not values which are negative. The query `if(var1 <= MAX SIZE)` is for negative values true.

Truncation errors arise, if integer values are too high:

```
1 void foo(int var1, int var2)
2 {
3     if(var1 <= MAX_SIZE & var1 >= 0 & var2 <= MAX_SIZE & var2 >=
      0)
4         int res = var1 + var2;
5         // continue processing, e.g. allocate memory using res
6     else
7         // handle error to allocate too much memory
8 }
```

In this example the developer checks each variable of the function `foo` for the highest and lowest boundaries. However the `res` value will be truncated

if the sum of `var1` and `var2` is too high. The maximum value to fit into a 32-bit singed integer is 2147483648 [32]. The following code snippet shows a integer truncation on a 32-bit UNIX machine:

```
#include<stdio.h>

int main()
{
    int var1 = 2147483647;
    int var2 = 1;
    printf("Variable var1: %d\n", var1);
    printf("Variable var2: %d\n", var2);

    int res = 0;
    res = var1 + var2;
    printf("Variable res: %d\n", res);
    return 0;
}
```

If this C code gets compiled, without compiler optimisations, the following overflow is visible:

```
admin@lab:~$ gcc -O0  int_overflow.c -o int_overflow
admin@lab:~$ ./int_overflow
Variable var1: 2147483647
Variable var2: 1
Variable res: -2147483648
```

The intended value of the `res` variable is 2147483648 and not -2147483648.

In secure software development it is important to select the correct data type (e.g. `int`, `float`, `short`) and the properly abstract data type (e.g. `unsigned`, `signed`). The data type needs to be able to handle all possible intended values and does not allow unwanted merits to occur. For example: A C developer should not pick the data type `short` for values, which can contain a higher number than 32,767. A data type including `int` or `double` is advisable. Furthermore, there is no reason to pick the C data type `int` to store, for example, the height over ground, since this value can not be negative. In this example the developer should use an `unsigned int` to store the value. Before the call of any function, the range of the input parameters should be validated. Every side for the boundaries should be checked. The developer needs to ensure that only intended values are passed to the function. For example:

```
int addNumbers(int var1, int var2)
{
    int res = 0;
```

```
4     res = var1 + var2;
5     return res;
6 }
```

Another mitigation strategy is arbitrary-precision arithmetic. In the course of this, a new integer type can be introduced, which is bound to the memory of the host system. This implies that an integer value can not be bigger than the disposable space of the computer system. GMP[28] provides a framework for the C programming language. This framework offers integer functions to initialize integer objects. The function `void mpz_init2(mpz_t x, mp_bitcnt_t n)`, which initializes the integer `x` with space for `n`-bit numbers [33]. Hence, the framework offers a robust security implementation for integers due to limited space.

Wind River's Workbench does not support the developer to prevent security mistakes relating integers.

5.2.4 Formatted Output (Format Strings)

Format strings are the input variables for formatted output functions such as `printf()`, `sprintf()`, or `scanf()` within the C programming language. Depending on the conversion specifier[29] of the format string the input variable is presented in a different way. If the inputs are not checked for integrity, an attack is able to leak data or even write data to the stack.

Format strings can lead to a buffer overflows. For example:

```
1 char buf[64];
2 sprintf(buf, "Some text %s", var1);
```

If `var1` is bigger than 52 bytes[30], the character buffer `buf` will overflow.

Morover, stack content can be leaked, if formatted output functions are not used properly. With C functions such as `printf(var1)`, stack data can be displayed, if the variable contains the format string `%p`. This will result in the print of a pointer on the stack.

Furthermore, if the variable `var1` is something like `ABCD%n`, the attacker is able to write to a specific address[31] on the stack. This can result in control flow changes.

[28] The GNU Multiple Precision Arithmetic Library https://gmplib.org/.

[29] For example `s` for a character string, `d` for an integer, or `p` for a pointer.

[30] 64 bytes - 12 characters of the "*Some text %s*" string

[31] Here, `0x41424344` which is the hexadecimal representation of `ABCD`

An integrity check of the input for formatted output functions provide a sufficient security aspect. Black- and White-Listing can be helpful in this case. The surest implementation to examine the input variable is white listing. That implies to block every input and only allow certain inputs. Blacklisting is when every input is accepted and a few are specifically blocked.

A black list can be implemented like this:

```
1 char *blacklist = "%!@#$%^&*()_+[]{}\|";
2 for(int i = 0; blacklist[i] != '\0'; i++)
3 {
4   if(strstr(var1, blacklist[i]) == NULL)
5     // further execution
6   else
7     break;
8 }
```

The character pointer `blacklist` contains multiple character, which should not be element of `var1`. If `var1` contains any of the `blacklist` characters, the program stops due to the `break` in line 7.

One possible white list implementation in C could look like this:

```
1 char *whitelist = "abcdefghijklmnopqrstuvwxyz"
2 for(int i = 0; whitelist[i] != '\0'; i++)
3 {
4   if(strstr(var1, whitelist[i]) != NULL)
5     // further execution
6   else
7     break;
8 }
```

This white list only allows lower case characters. If `var1` contains an element of the character pointer `whitelist`, the program continues execution. Otherwise, the program will stop execution, due to the `break` instruction in line 7.

The formatted output functions themselves provide a security mechanism. Developers can limit the number of bytes written by using the format string like `sprintf(buf, "Some text %.52s", var1)`. The use of `%.52s` results in a limit of 52 bytes, which will be written to the `buf` variable.

In addition, compiler modifications provide features to make the use of format string functions safer. Format guards can be implemented to dynamically check output functions. The number of provided format strings will be counted and compared with the intended number. For example: if `printf("Some text:  %s", var1)` is used, the intended number of format

strings is 1 (`%s`). Therefore, if `var1` contains more strings like `%x` or `%p`, the format guard will notice this and reject the function call. This can be done manually too, if the developer includes check for every format string input. Such a manual verification is presented in the following code snipped:

```
1 if(strstr(var1, "%") == NULL)
2     // further execution
3 else
4     break;
```

The `if` statement will check the variable `var1` for the character `%`. This example can be refined and extended to provide more protections against malicious format string function inputs.

As mentioned in Section 5.2.1 and displayed in Figure 6, Wind River's Workbench provides Format String vulnerability warnings or errors.

5.2.5 File I/O

The read and write to files is a popular attack vector for hackers. Files can contain valuable information including passwords and system information. Hence, during C/C++ development the focus on file permissions and privileges is important. The introduction of privilege management policies is essential for developers. The UNIX file-system introduces such policies by determining files with tree states: read, write, and executable. The dispensation of the least privileges for each user and each file can prevent security issues from occurring.

VxWorks supports file system management and privileges similar to UNIX. That implies that file system objects can be flagged as readable, writeable and executable. Further examination to VxWorks 7 privilege management is introduced in Section 7.2.

5.3 Specific Characteristics for Aircraft Systems

To just pick common vulnerabilities is not sufficient enough because aircraft systems bring specific characteristics. Those limit the ability of the computer system and consequential the vulnerabilities, threats, and attacks which occur or happen. Avionic software is subject to restrictions and regulations including the aviation standard DO-178C [34]. Therefore, technologies such as dynamic memory management or shared memory is not

applicable. On the other hand, technologies for instance authentication, access control, cryptography, or data protection is more important compared to a standard desktop system. From this it follows, to select avionic specific threats, vulnerabilities, and attacks to evaluate the security of the real-time operating system VxWorks 7.

5.4 Score Definition and Evaluation Structure

The three scores[32] are selected due to VxWorks' 7 reaction to vulnerable programs and audits. The vulnerable programs are well-known code snippets executing predefined and existing functions to force reactions such as buffer overflows or vulnerable writes to memory sections. The audits amount to code reviews and examinations of provided technologies, which are known to harden the system such as code signing or malware defences.

If the RTOS provides mechanisms or technologies, which prevent the tested threats, attacks, or vulnerabilities from happening, the score is put within the "*mechanism exists*" column. Furthermore, analysis such as tests or comparisons with other (real-time) operating systems are performed to determine if the provided mechanism is sufficient enough to grant protection. The result of this examination resolves to a mark in the corresponding column "*full protection*" or "*only extenuates*". If VxWorks does not offer any mechanism to prevent security incidents, the mark is placed in the "*mechanism does not exist*" column. Additionally, existing technologies are described, which could be used in the appropriate context.

[32] "*Mechanism exists with full protection*", "*mechanism exists but only extenuates*", and "*mechanism does not exit*".

Chapter 6

Design and Implementation

After a further analysis of the research question as well as the presentation of security awareness and secure programming, the design and implementation, the used course of action and components to evaluate the security of VxWorks 7 is presented in this chapter.

6.1 Representation Method

To clearly present multiple vulnerabilities, threats, and attacks and verify if VxWorks is vulnerable, the chosen representation method is a matrix as displayed in Table 6.1 on page 39.

The rows will be sorted in different categories of threats, vulnerabilities, and attacks. The selected categories are based on a paper by Afzali [35]. This publication describes a model based approach to categorize multiple security criteria and corresponding mechanisms in a hierarchical structure. Most of the presented classes (categories) for operating system security are usable for the security evaluation for VxWorks, while categories like logging and auditing, protection of meta data, and secure backup/restore are not suitable for a real-time operating system for avionic systems. The columns will depict, if VxWorks is prone to the vulnerabilities, threats, and attacks. A mark ("*X*") will be placed in the "*Mechanism exists*" or "*Mechanism does not exist*" cells. A tag in the "*Mechanism exists*" column determines, if VxWorks on Version 7 uses a security mechanism, which does not allow the viewed vulnerability to be exploited. A mark in the "*Mechanism does not exist*" column represents, if the real-time operating system does not support any security mechanism and the flaw can result in a security issue for an aircraft system. The "*Evaluation*"-column describes further explana-

tions. Those cells set a parameter ("*E1*", "*E2*", "*E3*", and so forth). These parameters will result in further explanations regarding the weak point and which mechanism is used or could be used by VxWorks to provide a security mechanism for the viewed criteria.

This method results in a clear depiction of weaknesses for operating systems and VxWorks 7 liability regarding them.

6.2 Relevant Threats, Vulnerabilities, and Attacks

To pick relevant vulnerabilities for a real-time operating system like VxWorks it is important to choose only weak spots, which are usable for the viewed area, here avionic systems. The picked vulnerabilities are the most common vulnerabilities from reliable and community-developed sources *Common Vulnerabilities and Exposures* (CVE) database [36] and *Common Weakness Enumeration* (CWE) book [37]). Both sources provide a extensive collection on threats, vulnerabilities, and attacks against vendor products, including operating systems as well as real-time operating systems. Furthermore, the picked vulnerabilities are the most relevant security issues for avionic software due to the following reasons:

As displayed in Figure 1, future aircraft systems introduce an extensive amount of communication between several entities including aircraft, ground stations, satellites, and drones. Hence, reliable authentication mechanisms are necessary to provide secure communication. CWE-287 describes improper authentication as an attack where the attacker pretends to be someone else with other privileges.

Military aircraft systems work with highly sensitive and secret data. This data should be protected from unintentionally access because access to such information can result in military mission relevant disadvantages and danger to the aircraft and environment. Privilege management is described in CWE-264/266/269 and shows the management of assignments, which cover access control.

Malware is an increasing threat to embeeded systems. The BSI shows a clear increase of existing malware from 100 million in 2012 to approximate 620 million in 2017 [3]. Malware can infect IoT and embedded devices including aircraft systems. Even though these systems are usually not connected to the internet, Malware can gain access using other ways such as the transfer via USB during the read-out of mission data or even over satellite or ground station connections.

Buffer overflows are common vulnerabilities in current applications, which

includes avionic software. Available complicated and resource expensive protection frameworks are not adaptable in an aircraft because of the limited processing power. Since resources are limited within aircraft systems, buffers are usually very small and overflows are likely to occur if the software is not developed with a focus on security.

CWE-121/122/123 define string vulnerabilities. Those are security issues regarding user/external input and format functions are likely to occur in software. The vulnerabilities can occur in aircraft software as well, since strings are usual data types.

Data within an aircraft system is highly sensitive and secret. This information should be treated with special attention. If data is transferred in clear-text representation, information can be leaked, which can result in military disadvantages. RTOS can provide secure cryptography algorithms in order to grant the ability to decrypt and encrypt data for secure transfer.

Following from encryption and decryption of sensitive information, improper storage of information including cryptographic keys can lead to security issues. Separated storage locations and similar technologies can impede the leakage of sensitive data and make the aircraft system more secure.

The possibility to execute system commands is a powerful ability for an attacker. Command injections base on those functions and are defined within CWE-77/78. These vulnerabilities grant an attacker full control over a system. Hence, avionic systems should be protected against unintentional execution of commands. The RTOS can provide such protection using security technologies and mechanisms such as user input control.

Software updates and modifications are common within avionic aircraft systems. Each client has different requirements to the military aircraft. Therefore, the software needs to be changed regularly. The integrity of new software units should be ensured. The RTOS can provide check mechanisms including code signing to protect the aircraft from malicious software.

Category	**Threat, Vulnerability or Attack**	**Mechanism exists**			**Mechanism does not exist**	**Evaluation**
		Name	Full protection	Only extemuates		
Authentication	Improper Authentication (CWE-287)					
Access Control	Privilege Management (CWE-264/266/269)					
Data Protection	Malware Protection					
Resources management	Buffer Overflow (CWE-121/122/123)					
	String vulnerabilities (CWE-133/134)					
Cryptography	Secure cryptography algorithm					
	Storage of sensitive information					
Other	Command Injection (CWE-77/78)					
	Integrity check of external software					

Table 6.1: Security Evaluation for VxWorks Version 7

6.3 Infrastructure

Wind River provides tools to realise the creation of kernel images, virtualization, and simulation of the real-time operating system VxWorks 7 as well as the virtualization and simulation of several targets. The used set-up is displayed in Figure 7.

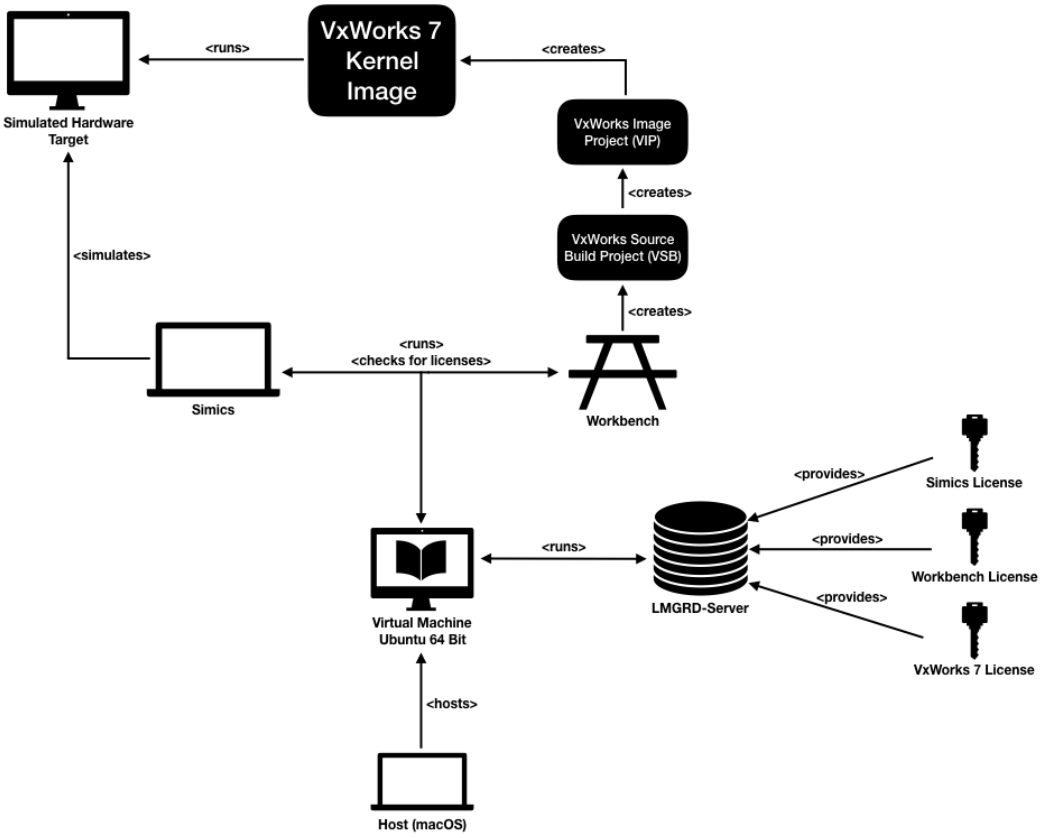

Figure 7: Simulation Setup

A Ubuntu 64-bit virtual machine (VM) is applied to run all necessary applications. This VM is hosted by the virtualization program *VirtualBox* from *Oracle* on an macOS[33] host system. The Ubuntu machine runs three applications. *Wind River's* Simics (Chapter 2.4), Workbench (Chapter 2.4), and a License Server Manager called LMGRD, which is a daemon

[33] *Apples* operating system for *Apple* computers.

for *FLEXnet* Licensing and dynamically provides floating and network licenses [38]. Three licenses are embedded within the LMGRD server: Simics, Workbench, and VxWorks 7 license in one combined license file. Every time, one of the programs verifies a license, they obtain them from the LMGRD server and check its validity.

All entities grand the ability to simulate VxWorks 7 with multiple configurations or features enabled (see Section 6.4). The hardware boards can be switched on demand as well as combined and linked together to simulate a whole system of multiple computers, which interact with each other, similar to a real aircraft system. The simulation of whole computer systems is not part of this thesis, because VxWorks 7 as a part of an avionic system is considered.

6.4 Workbench configuration

Wind River's Workbench is used to generate a VxWorks 7 image, which will then be used to run on a simulated target using Simics, as illustrated in Figure 7. To generate a VxWorks 7 image, two steps are necessary: First, a VxWorks Source Build Project (VSB) is created. Second, a VxWorks Image Project (VIP), based on the VSB, is generated. The VIP creates the needed VxWorks 7 image as a Executable and Linking Format (ELF) 32-bit PowerPC executable file.

To create an image with all necessary security mechanisms and features provided by VxWorks 7, a VSB is required. If options are excluded they are fully deleted from the configuration and not only disabled. The following bullet points displays the VSB options with a listing of the picked configuration:

- *VxWorks Global Configuration Options for BSP qsp_ppc* – This option describes basic configuration parameters including the endian configuration[34], the compiler version, floating point configuration, and more. Those settings are left on default since there is no need to change options for the board support package for the QSP.
- *ARCH selection(s) for PPCE6500* – These options allow to edit PowerPC libraries. They are left on default because there is no need to change these options for the viewed security evaluation.

[34] Arrangements of bits in a computer system.

- *VxWorks Kernel Configuration Options* – It contains VxWorks library configuration flags and options. This can not be edited in the selected VSB[35].

- *os* – The *os*-parameter offers operating system specific configurations, including language, debug options, utilities (JavaScript Object Notation (JSON) library, VxWorks shell, etc.), firmware options (Advanced Configuration and Power Interface[36] (ACPI) and VxWorks Flattened Device Tree[37] (FDT)), and more. The configurations are left on default since there is no relevance relating to the tested threats, attacks, and vulnerabilities.

- *rttools* – Contains runtime analysis tools to analyse VxWorks 7 during processing including additional debug parameter, a target runtime analysis tool, and similar. The default configuration enables all necessary tools including debug options and the ability to analyse VxWorks 7 during the runtime.

- *security* – This contains all security relevant configuration options including encryption libraries, hash algorithms, encryption APIs, user management options, and more. Since the VSB configuration does not result in the creation of the kernel image itself, nearly every option is enabled within the security configuration to enable them for the VIP. All networking options like Simple Certificate Enrolment Protocol (SCEP), which enables network certificate handling, as well as options, which are architecture specific[38], are not enabled since those networking technologies are not used within current aircraft systems.

- *connectivity* – Connection dependent options including Controller Area Network (CAN) support, Universal Serial Bus (USB) system support, and similar are included within this parameter. Those configuration are left on default since no such connection is considered in this thesis.

- *storage* – Storage configurations like file-system-support, -applications, and similar are part of this option. All options are left on the default

[35] It is not clear why it is not editable, since no information is displayed.

[36] Enable OS-directed motherboard device configuration and power management. [39]

[37] Copy of a binary blob in the kernel memory space, which describes the hardware resources of a computer system. [40]

[38] Options like OP Trusted Execution environment (OP-TEE) can not be implemented since they require the ARM architecture.

values, which allows the storage of cryptographic keys and user data used within the *security* configuration parameters.

- *net* – All networking options are left on default, which enables all parameters including Ethernet network device support, IP networking stack, Simple Network Management Protocol (SNMP), Management Information Base (MIB) for networking stack, Network Information Center (NIC), and networking basic library. Real-time network stack is not enabled since this option is only usable with the ARM architecture.

- *ipc* – The inter process communication options are left on default. This allows socket-oriented messaging communication between several applications in multi-node or multi-OS systems. Those configurations are not considered in this thesis.

- *ui* – User interaction options including audio devices, frame buffer interface for pictures, font libraries, image libraries, and more are left on default. As a consequence, everything is disabled and only event device support is enabled to manage all input devices and events.

- *app* – WebCLI and Extensible Markup Language (XML) are part of this configuration parameter. Both are not necessary for the security evaluation performed in this thesis.

- *vip_profiles* – VIP profiles can be included by using this parameter. The VIP configurations are set from the generation of a VIP project. The parameter is enabled by default but not explicitly used.

- *VxWorks User Library Configuration* – This configuration parameter can be used to add user-side libraries and user custom applications. By default, this option is enabled but no dedicated use of this parameter during the thesis is done.

The chosen VSB configuration and creation enables the generation of VxWorks 7 images using a VIP. If options are excluded, they are fully deleted from the image file and not only disabled. The corresponding configuration is separated in the following items:

- *Application components* – Includes parameters such as an application initialization parameter to add a function, which will be called by start-up, post-, and pre-kernel initializations, and other start-up facilities. All options except application initialization are not used since they are not necessary for a security evaluation.

- *C++ components* – No C++ components are used in the VIP configuration since there is no dedicated C++ software development for the security evaluation in this thesis.
- *Development tools* – This options introduces several tools for development purposes including the kernel shell, runtime analysis, system viewer, VxWorks debug library, and more. The options are enabled to use the necessary debug functionalities:
  - → *Application Mode Agent components* – Target Communication Framework (TCF) based communication system to enable debugging, profiling, and code patching at runtime.
  - → *Downloadable kernel modules compiler support routines* – Adds the ability to use GNU compiler support routines for C code.
  - → *Kernel shell* – Enables the kernel shell to interact with VxWorks.
  - → *Kernel-write* – Adds the function `printf()` to the kernel and symbol table.
  - → *Show routines* – Options like Address Space Allocator to display the address space of a program, routines for the memory to show memory partitions, the routines for information about symbols and symbol tables, and more. This parameter allows to display memory during debugging and testing.
  - → *Loader components* – For VxWorks kernel object management to load libraries and add downloaded objects into the running target.
  - → *Symbol table components* – Adds a data structure, called symbol table, for compiler options, which is needed to display information about the compiler and grant the compiler the ability to compile the programs source code to a binary file e.g. application.
  - → *Memory Error Detection and Reporting* – If memory errors occur, VxWorks 7 will report those errors and display information about the error.
  - → *VxWorks debug library* – To support runtime debugging.
  - → *spy* – A utility to measure per-task CPU utilization. This is useful to measure and notice DoS attacks and if a program is situated in an endless loop, which can cause a delay in program execution.
  - → *tip serial line connection utility* – To manage serial connection lines e.g. an escape character for the CLI.

- *Hardware* – The listed hardware components are for example the BSP configuration variants, device drivers, peripherals, memory, and more. All options except the bus library system are added, since this feature is not used in this security evaluation.
- *Network components* – Implements the ISO/OSI model layers[39] [41], and other network components. All options are added.
- *Obsolete components* – This parameter can not be edited and added to the VIP, since it will be removed next release due to a note in the configuration parameters.
- *Operating system* – The *operating system* parameter adds options including ANSI C programming language components, I/O systems, the kernel, a random number generator, and more. All configurations are added since these parameters are part of this security evaluation.
- *Security* – Adds components like cipher algorithms, hash algorithms, OpenSSL, protected storage, kernel hardening features, and user management. These parameters are enabled to test the corresponding security mechanisms, which are tested in this thesis.
- *Storage* – Within the *storage* parameter a file system to store cryptographic keys and user passwords is added in a read-only memory (ROM) file system as well as a random-access memory (RAM) disk.
- *User interface components* – This parameter can not be added since it is not added within the VSB and therefore the option is not addable in the VIP.

With all the chosen VIP configuration a VxWorks 7 binary is generated as shown in Figure 8.

```
vxWorks: ELF 32-bit MSB executable, PowerPC or cisco 4500, version 1 (SYSV), sta
tically linked, not stripped
```

Figure 8: VxWorks 7 Kernel Image

For the evaluation parameter "*Buffer Overflow (CWE-121/122/123)*" and "*String vulnerabilities (CWE-133/134)*" two different workbench configuration are selected. One with all kernel hardening features enabled and

[39] VxWorks 7 uses a custom ISO/OSI model with 5 layers: physical, data link, network, transport, application

one without any security features. The use of two configurations allows the distinction of VxWorks 7 with a light-weight and default[40] configuration as well as the RTOS with dedicated security features enabled. The used configuration is mentioned for each evaluation parameter if necessary. If options are excluded, they are fully deleted from the image file and not only disabled. Kernel hardening features provide the following security mechanisms:

- *Guard pages for the interrupt stack* – Provide additional memory[41] for the stack, which handles interrupts. Adds the guard to prevent buffer overflows and buffer underflows.
- *Enable all kernel hardening features* – Provides protection to certain kernel components (text segment write protection, write protection of the exception vector table, task stack overflow and underflow detection, non-executable task stacks, non-executable heap and data sections, NULL pointer dereference detection for kernel tasks).
- *Guard pages for kernel task stacks* – Adds additional memory[42] to protect the kernel tasks stacks against buffer overflows and buffer underflows.
- *Non-executable memory protection* – Provides the characteristic that only the memory pages containing the *.text*[43] (and *.rodata*[44]) ELF sections will be executable. All other memory pages, e.g. pages containing the *.data*[45] ELF section, task stack, and heaps are marked non-executable.
- *Non executable stacks* – Exception and execution stacks for Real-time Transfer Protocol (RTP) and kernel tasks will be marked non-executable.
- *Write protection for program text* – Provides protection against writes within the *.text* segment of the program.
- *Write protection for the vector table* – Provides security against writes within the vector table[46].

[40] No special security features.
[41] Size can be chosen by the developer. Default size is 4096 bytes.
[42] No information by *Wind River* relating to the size.
[43] Contains machine instructions.
[44] Read-only data section.
[45] Static data for the application including strings and static variables.
[46] Contains a list of pointers, which contain interrupt information.

6.5 Simics configuration

Common hardware boards used in avionic software development, such as the P4080DS, support the following flash memory systems as discribed in *Wind River's* the corresponding fact-sheet [42]:

- 128 Mega Byte (MB) Not-Or-(NOR)-flash.
- Three Inter-Integrated Circuit (I^2C) controllers supporting Erasable Programmable Read-Only Memory (EEPROM).
- 16 MB Serial Peripheral Interface Bus (SPI)-based EEPROM memory.
- One Secure Digital (SD) media card slot.

However, each memory system is capable of performing a limited amount of program and earase cycles. NOR-flash allows 100.000 program/erase cycles [43]. EEPROM memory is capable of approximalty 10^6 write-erase operations [44]. The maximum of possible read and write cycles for SD media cards depends on the used flash technology. For example, one of the biggest SD card vendors *SanDisk* entitle the endurance for their SD cards with approximalty 100,000 write and erase cycles [45].

The limited number of possible flash cycles results in the difficulty to test multiple kernel images on a hardware board, since the memory can not be flashed ad libitum. The change of broken flash memory can be costly. Hence, the hardware and software virtualization of targets like the mentioned development board and VxWorks 7 lead to a solution for this problem.

The picked board to test the security mechanisms and technologies provided by VxWorks 7, is the Quick-Start Platform (QSP) PowerPC Board. Figure 9 shows the target selected in the Simics-Control-Window. At this time, the already mentioned hardware board P4080DS can not be simulated while running VxWorks 7. *Wind River* does not provide VxWorks 7 Simics start-up scripts yet. These scripts are used to select the RTOS kernel image as well as a Device Tree Blob[47] (DTB). Therefore, the scripts are necessary to simulate the RTOS on the P4080DS.

[47] Likely and Boyer desribe DTB in [46] as a tree structure, which initialises the hardware configuration including information about the CPUs, memory banks, buses, and peripherals.

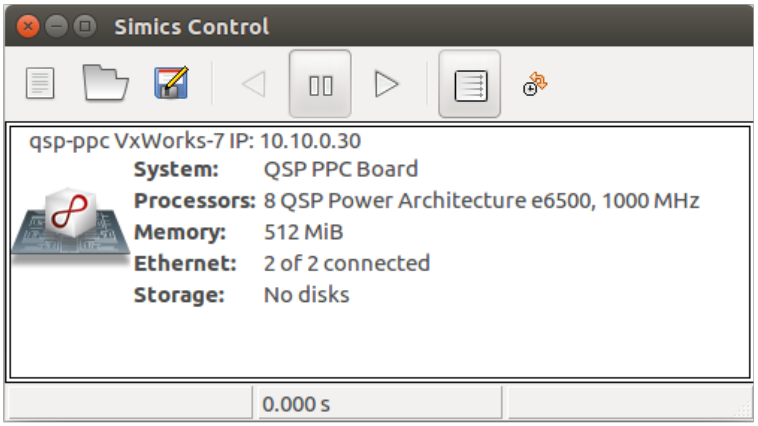

Figure 9: Simics Control QSP PowerPC Board

QSP was introduced by Windriver in 2012. QSP boards do not physically exist. Its architecture is similar to existing boards. As displayed in Figure 10, a QSP target supports all necessary units compared to development boards like the P4080DS. As well as the P4080DS, a QSP target supports serial and ethernet connections, an interrupt controller, multiple CPUs, timers, and more [47]. Hence, the QSP can be used as a replacement for the P4080DS due to its similarities with the mentioned development system board. [48]

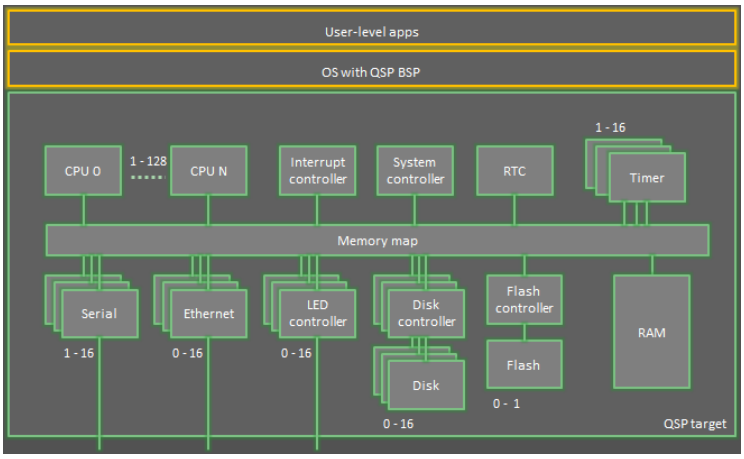

Figure 10: QSP PowerPC Board Architecture

Chapter 7

Evaluation

After the introduction of a set up, which allows to evaluate the security of VxWorks 7, the security evaluation can be realized. Each, in Chapter 6, mentioned threat, vulnerability, and attack is listed in the following sections and analysed. The sections are structured the following way: First, a representation of the problem. Followed by the provided or possible technologies and mechanisms. Afterwards, the approach to test the relevant parameters and to finish each evaluation parameter, the reaction of VxWorks is presented. Chapter 7.10 recaps the evaluation results, provides a general overview, and displays the final matrix (Table 7.1).

7.1 E1 – Improper Authentication (*CWE-287*)

CWE-287 defines the weakness class *Improper Authentication*, which discusses authentication within a computer system. It is introduced in the architecture, design, and implementation phase. Due to the medium and high likelihood for an exploit *CWE-287* has a high importance. [49]

Common threats regarding authentication are the use of default credentials, weak passwords and poor login policies. This becomes apparent in incidents like the *Twitter* account hack from 2009 [50]. An 18-year-old hacker was able to hijack multiple *Twitter* accounts by guessing and brute-forcing passwords. In this time *Twitter* did not restrict the number of failed login requests, which made it brute-force-able. Furthermore, the users were allowed to use weak passwords like "happiness".

To test this parameter, a manual audit is chosen. The basic authentica-

tion schemes and methods are inspected and evaluated.

VxWorks 7 introduces several policies for the login and the passwords used within the RTOS. By default none of the policies is enabled, which is a security issues. *DO-178C* [34] defines no specific security part in the development cycle of an aircraft. Although, existing security standards such as DO-326 [51], DO-355 [52], DO-356 [53], ISO-15408-1 [54], ISO-15408-2 [55], and ISO-15408-3 [56] examine security. The main focus on this documents is organisational security as well as basic techniques to ensure security. There is no specific focus on software security. Therefore, there might not be a focus on password and login policies for VxWorks 7 during the development. This can lead to major security issues, if an attacker is able to brute force or even manually guess credentials.

Furthermore, the developer has the ability to add custom login and password policies manually/programmatic, which can harden the whole aircraft system. If avionic system units (e.g. ground stations) or other aircraft need to authentication themselves to be able to communicate with each other, trust and policies can harden the process.

As shown in Figure 11, multiple parameters can be set to harden the login process. Login policies allow to set a maximum number for failed login attempts. If the maximum number is exceeded, the account is disabled for X^{48}-minutes.

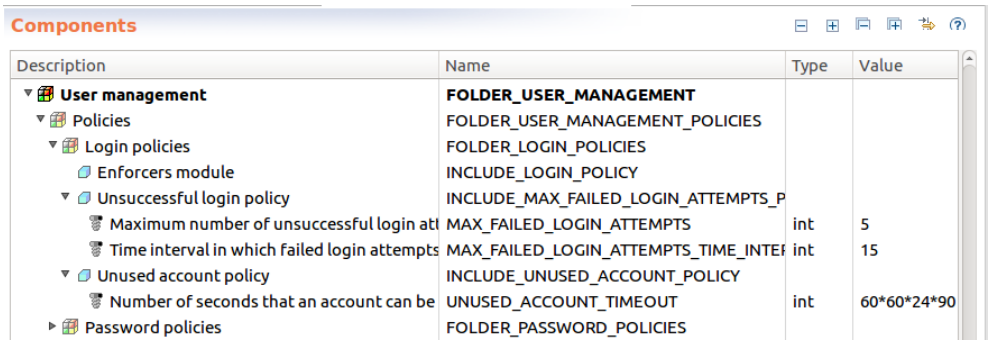

Figure 11: VxWorks 7 Login Policies

Moreover, several configurations can be set to define the complexity for passwords as shown in Figure 12. Password policies can be used to force the user to choose sophisticated and secure passwords. Multiple attributes

[48] Dependent on the chosen time interval.

including the minimum number of lower case characters, numbers, special characters, and upper case characters can be set. Furthermore, the minimum and maximum password length can be defined.

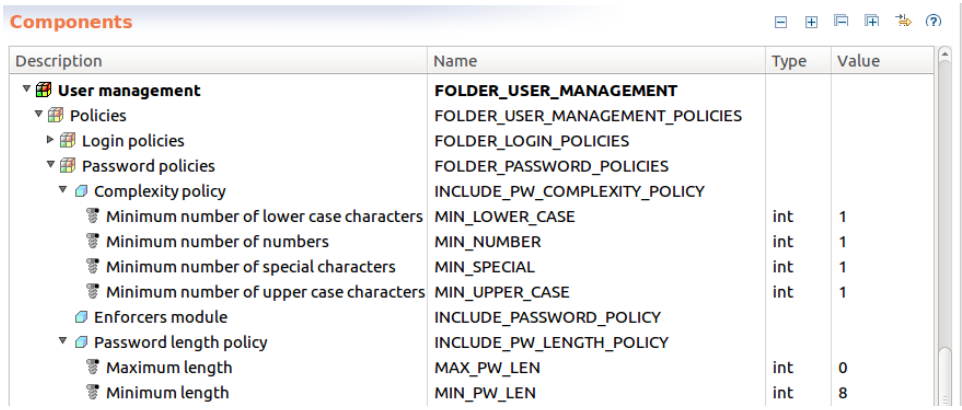

Figure 12: VxWorks 7 Password Policies

The manual audit revealed that the initial user on a VxWorks 7 machine is granted administrative privileges [57] and is called *vxworks* as shown in Figure 13. The user name and the name of the image are the same. *Administrative* privileges allow the authorized user to gain full control of the system. The corresponding password is equal to the user name.

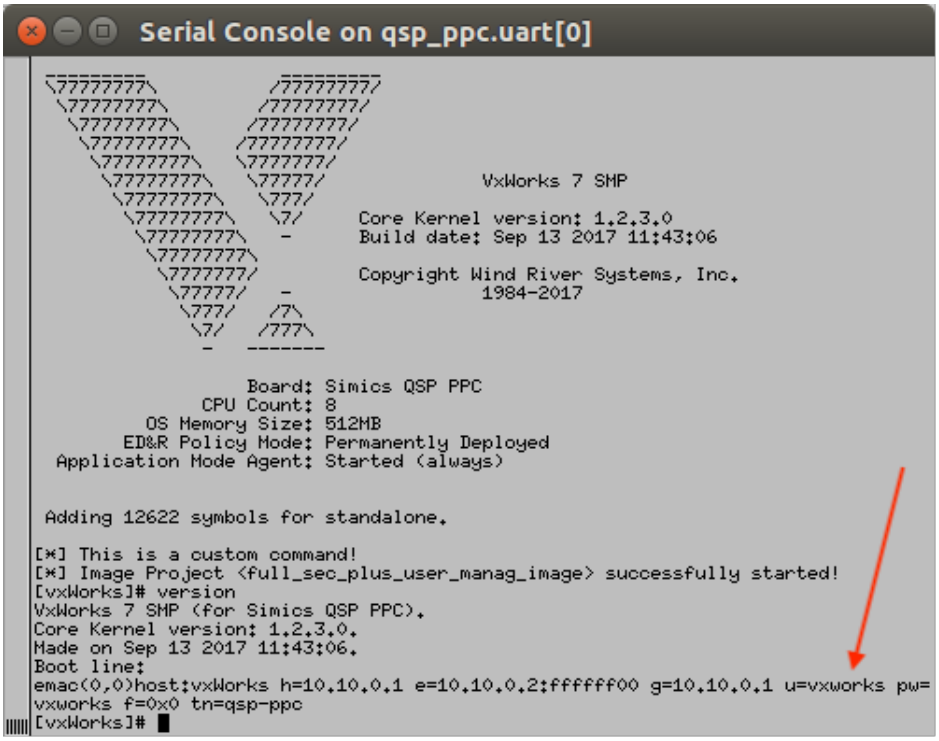

Figure 13: Initial VxWorks 7 User *vxworks*

Default credentials are an example for bad security awareness. Publicly available lists [58][59][60] contain a massive amount of default credentials for applications, including VxWorks. These credentials can be used to gain unitentional access to information. This becomes apparent in the reserach of the security website *KrebsonSecurity*, which reported that the credit reporting company *Equifax*[49] used the username and password combination `admin/admin` for their login on a website for employees of an Argentina office [62]. As a consequence, default credentials should be changed.

From this evaluation it follows that VxWorks 7 offers protection mechanism, but not by default. Therefore, the evaluation parameter is located within the "*Mechanism exist*"-column and "*Only extenuates*"-column.

[49]The servers of *Equifax* were breached in May 2017, which resulted in the theft of sensitive private data of 143 million customers [61].

7.2 E2 – Privilege Management (*CWE-264/266/269*)

Privileges are necessary to maintain access control within a computer system. Access control defines the constraint for entities like users or files to access other objects. The likelihood of exploit is marked as medium by CWE [63]. Wrong privilege assessment and management can result in major security issues.

During this evaluation, manual audits are used. VxWorks 7 introduces multiple file system formats and storage mediums. As mentioned in Section 6.4 the ROM and RAM file system are used in this thesis. By default, ROM is granted read-only privileges and RAM full[50] privileges. A database containing the different users is stored within the RAM file system. ROM contains encryption relevant data including the Key Store. As a consequence, user credentials can only be read. However, sensitive cryptographic information is not protected by default, which can result in security issues.

The first to login and consequential default user in VxWorks 7 is granted administrative rights. Hence, privilege management needs to be done by the developer.

In 1975, Saltier and Schröder defined the concept of least privileges as follows: *Every program and every user of the system should operate using the least set of privileges necessary to complete the job.* [64]. Following this rule, access control is less prone to attacks and vulnerabilities.

As a result of this, the mark is set within the "*Mechanism exists*" and "*Full protection*" column. Although, VxWorks 7 allows to create a user database where users with different privileges for diverse file systems and regions can be defined. As a result VxWorks 7 allows to define a more precise management for access control because users and file systems privileges can be customized. From this evaluation it follows that VxWorks 7 offers plenty of security mechanism to prevent attacks against the privilege management.

7.3 E3 – Malware Protection

J. Powers defines malware as malicious code with the goal to compromise the integrity of a computer system using automated exploitation techniques

[50] Read, write and execution.

[65]. Hence, malware exploits a known vulnerability fully automated, which means, no further analysis by the attacker are necessary for the compromise of a system. This makes malware very dangerous. Therefore, appropriate malware protection is necessary.

The total number of attacks using malware had increased over the past years. Based on *McAfee* Labs research, the number climed from approximatly 5.5 billion in Q1 2016 to 6.8 billion in Q1 2017 [66]. This shows a trend in an increase of malware attacks. Furthermore, a rise is visible in 2017th cyber security incidents including WannaCry[51] [67] or Petya[52] [68].

Malware defences in common operating systems including Windows and Linux are either signature-based [69] or behaviour based [70] and are realised using applications similar to anti virus software.

Signature based is detection regarding the structure or signature of a file. If the viewed file shows malware characteristics, it is marked as malicious and further examined.

During behaviour based malware detection, the viewed program is fully or partially run in an isolated environment such as a sandbox. If the program shows malicious intentions, it will not be loaded on the main system.

Both implementations are realised in state of the art anti virus software. However, regarding a paper published by the anti virus testing company *Anti-Virus Comparative* [71], the malicious code detection rate for an off-line search is on average 94% and for an on-line search on average 99.3%.

Military aircraft would be only capable using an on-line search because it is very resource consuming and those are limited within a military aircraft. From this it follows that on average 0.7% of malicious code would not be noticed. The U.S. fighter jet F-35 has more than 8 million lines of code over all used computers [72]. 0.7% would be 56.000 lines of code, which results in a lot of space for malware to be hidden in.

An off-line search can be realised using external computing power. Although it needs to be ensured, that the software on the system does not change while uploading it to the aircraft or during the flight.

Certainly, malware gets better over the past year. New mechanisms to hide malware from defense technologies like changing the signatures of mal-

[51]The data of computer systems running *Microsoft* Windows was encrypted.

[52]Encrypt files of a hard drive for *Microsft* Windows computer systems.

ware[53] [73] or malware encryption[54] [74] get introduced.

In a similar frequency to new malware obfuscation techniques, improved detection mechanisms are introduced. For example, Al-Saleh introduces an add-on to check network transfered malware upon arrival using a whitelist apporach [75]. This mechanism can be used in a military aircraft environment to check software before it gets loaded on to the aircraft.

Furthermore Aaraj shows a framework to detect malicious code on embedded devices [76]. The author runs the viewed program in a testing environment and in the real environment. It ensures to check the programs intentions and check for malicious behaviour. This approach is based on behaviour based malware detection and can be useful for military aircraft systems due to its lightweight and low use of resources. The drawback is the need of two separate environments. The shown implementation is effective because it detected all tested[55] malware instances.

VxWorks 7 does not provide any malware protection. However, the RTOS offers the possibility to write pre-boot and kernel scripts, which can check malware using the mentioned mechanisms. The development and implementation of such mechanisms is very time and resource consuming. Therefore, the mark will be set within the "*Mechanism does not exist*"-column.

7.4 E4 – Buffer Overflow (*CWE-121/122/123*)

Common (real-time) operating systems are vulnerable to buffer overflow attacks. This is noticable in statistics published by CVE. In 2017, 37 overflow vulnerabilties in the Linux Kernel [77] and 46 in *Microsoft* Windows 10 [78] were published. Furthermore, in QNX[56] real-time operating system, 7 total overflows [79] and in Wind Rivers VxWorks, 1 overflow [80] is listed in the CVE database. These numbers display that buffer overflow vulnerabilities are widespread in OS and RTOS.

For this evaluation vulnerable code is used. Each code snipped is separately introduced.

[53] Malware usually gets identified by its signature.

[54] The malicious code is encrypted. Anti virus software is not able to decrypt it and check for malicious intentions.

[55] This includes 15 publicly known malware such as *Zeus-1* and *Stuxnet-2*.

[56] https://blackberry.qnx.com/

VxWorks 7 implements kernel hardening features, which are further examined in Chapter 6.4. First, the scripts are tested on the RTOS without kernel hardening features enabled. Afterwards, the features are active and the code snippets are tested again to display, if the security mechanisms provide any shelter.

To test possible buffer overflow attacks against VxWorks 7, the start-up script was modified with the following C code snipped:

```
1 [...]
2 char * input;
3 char buf[10];
4 input = "AAAA";
5 printf("Stack before strcpy:\n%p\n%p\n%p\n%p\n%p\n%p\n%p\n%p\n%p\n%p
    \n%p\n%p\n%p\n%p\n%p\n%p\n%p\n%p\n%p\n%p\n%p\n%p\n%p\n%p\n%p\
    n%p\n%p\n\n");
6 strcpy(buf, input);
7 printf("Wrote <%s>\n\n", buf);
8 printf("Stack after strcpy:\n%p\n%p\n%p\n%p\n%p\n%p\n%p\n%p\n%p\n%p\
    n%p\n%p\n%p\n%p\n%p\n%p\n%p\n%p\n%p\n%p\n%p\n%p\n%p\n%p\n%p\n
    %p\n%p\n\n");
9 [...]
```

Using the C function `strcpy`, the content of the variable `input` is copied into a character buffer called `buf` with 10 elements. Each element has the size of a character (`char`) with 4 bytes, as displayed in Figure 14.

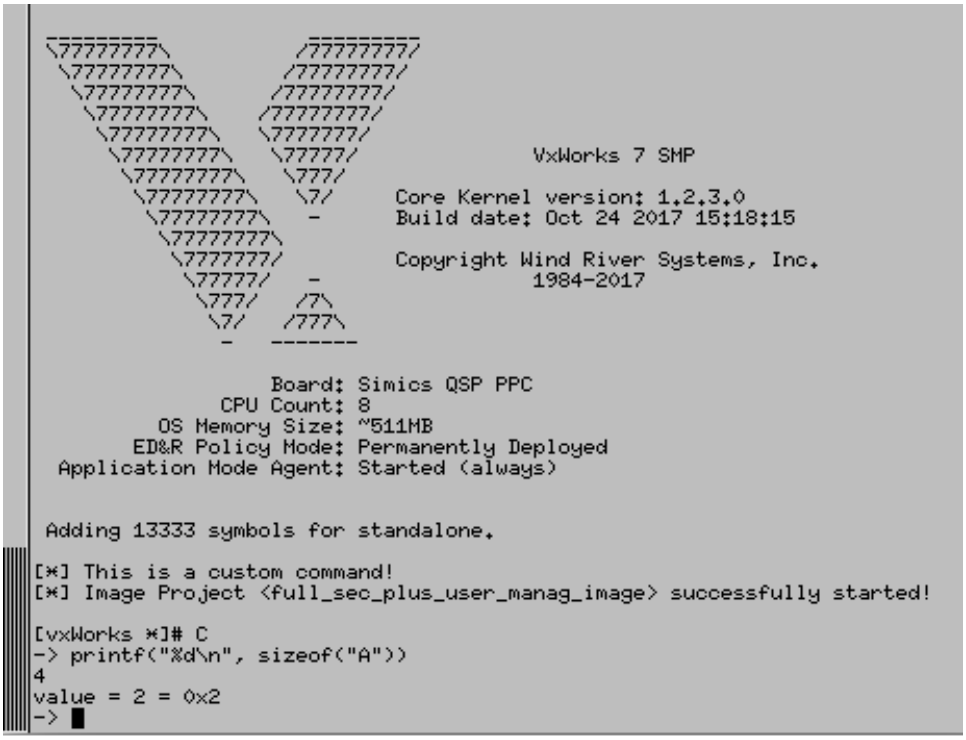

Figure 14: Size of the string "A" on the VxWorks 7 machine

The `input` variable can contain any possible string[57]. Furthermore, data copied into a variable with fixed size (e.g. a buffer) does not need to be a string. It can be in any possible supported format. Although, strings are being used in the following test.

The values of the `input` variable is changed to a string containing 44 ASCII characters. As displayed in Figure 15, VxWorks 7 opens the usual shell since no buffer overflow occurred.

[57]For example: A basic string as used in the following evaluation or communication relevant information.

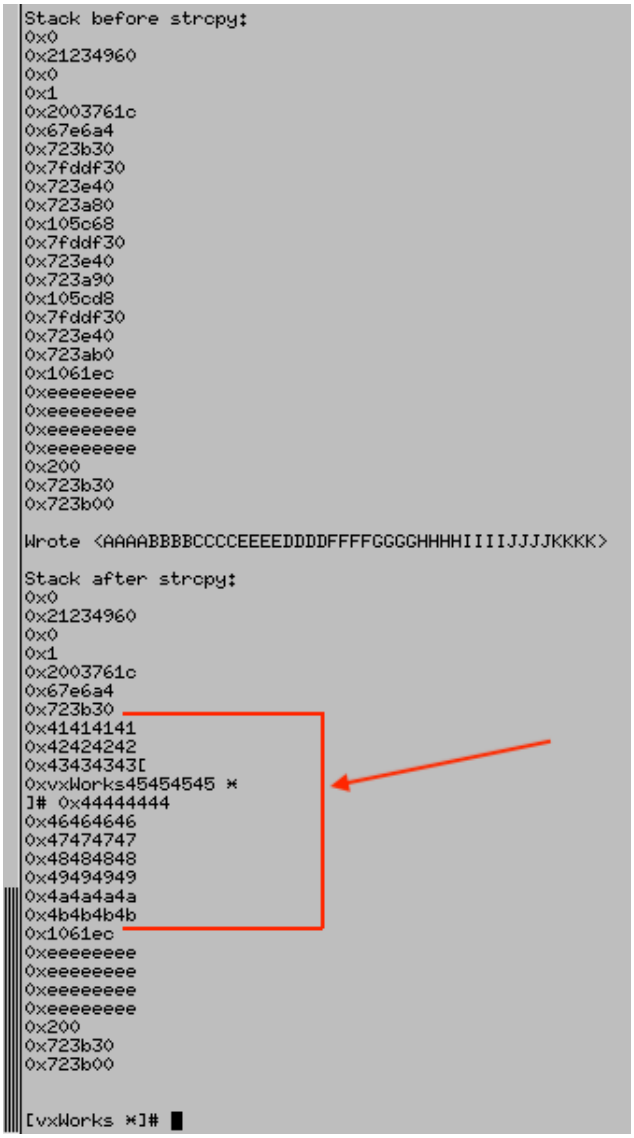

Figure 15: Buffer Overflow Test – 44 character copy

The next step is to overflow the stack. As depicted in Figure 16, the input is now 45 characters, which results in an overflow of the `buf` variable:

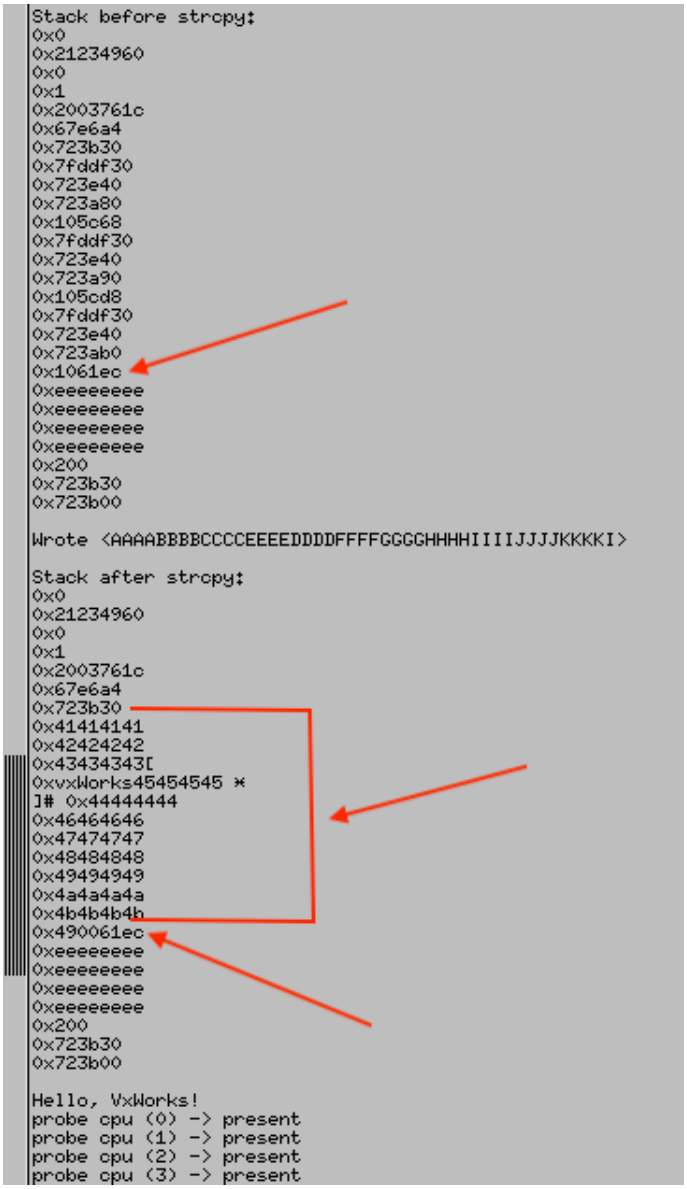

Figure 16: Buffer Overflow Test – 45 character copy

This overflow is present because the buffer has only space for 44 elements, since 44 divided by the number of elements (10) is 4. As already mentioned before and displayed in Figure 14, the size of one ASCII character (e.g. "`A`") on this machine is 4 bytes. From this it follows, that the buffer overflow

happens on the 45th element. As displayed in Figure 15 and 16 the pointer `0x1061ec` gets overwritten. This pointer is called return pointer and points to the next instruction, which will be executed on the stack.

Due to the buffer overflow, the usual VxWorks shell does not open since the valid instruction at address `0x1061ec` (which points to a valid memory address) has changed to `0x490061ec` (does not point to a valid instruction). This test shows that VxWorks 7 does not support any additional protection against a basic buffer overflow without using the kernel hardening features.

The next step is to enable the kernel hardening features and test the above mentioned script[58] again. As before, the shell does not open and the stack layout looks exactly the same (Figure 16). As displayed in Figure 17 the RAM file system is ejected to prevent further damage from happening. Although, VxWorks 7 needs to be booted manually in order to use it again.

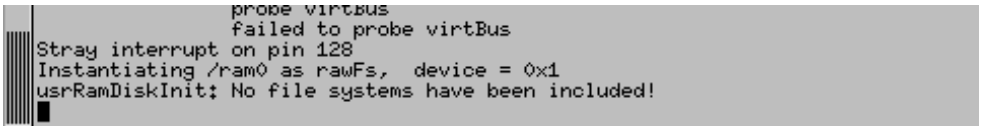

Figure 17: Buffer Overflow Test – File system ejected

From this it follows that buffer overflows can crash the system. The tests are positive whether the kernel hardening features are enabled or disabled. An attacker is not able to write at arbitrary memory regions due to memory protections. Secure software development prevents buffer overflows from happening. Therefore, the evaluation parameter is located within the "*Mechanism exists*"-column and "*Only extenuates*"-column.

7.5 E5 – String Vulnerabilities (*CWE-133/134*)

Improper use of format specifiers within C format functions such as `printf()` can result in data leakage and code flow changes.

A format function takes format specifiers, which indicate how the corresponding variable is displayed. For example: In `printf("%d %s", var1, var2)` two different variables (`var1` and `var2`) are displayed. The parameter `var1` is represented as an integer due to the format specifier `%d` and `var2` as a string pointer due to `%s`. If the number of variables is lower than the

[58] The buffer overflow script with an input of 45 characters.

number of format specifiers, the format function underflows the stack, otherwise it will overflow the stack [81].

VxWorks 7 implements kernel hardening features, which are further examined in Chapter 6.4. First, the scripts are tested on the RTOS without kernel hardening features enabled. Afterwards, the features are active and the code snippets are tested again, to display, if these security mechanisms provide any shelter.

To test format string vulnerabilities on the VxWorks 7 system, the startup script is modified in different ways. The following implementation is used:

```
1 [...]
2 char * input = "test";
3 printf(input);
4 [...]
```

The character pointer `input` is modified and can contain all different strings. Furthermore, the content of the variable is directly passed to `printf()`. As mentioned in Section 7.4, the variable can contain any thinkable string. The code snipped is vulnerable to a format string attack, if the attacker is able to modify the `input` variable. To simulate a worst case scenario, it is assumed that the attacker has control over the content of the `input` variable.

The first test is to leak data from the stack. A leak of stack data can debilitate the security of the aircraft system. The size of variables or memory areas, addresses, and more can be used to gain information about the system and its functionality.

Avionic software is flagged with the utmost secrecy due to the military background. Such leaks can make reverse engineering more simple and more efficient, which allows easier attacks since the attacker has a lot of knowledge about the system.

In the following test, the value of the `input` variable will be changed to multiple `%p`'s, which results in the display of several pointers:

```
1 [...]
2 char * input = "\n%p\n%p\n%p\n%p\n%p\n%p\n%p\n%p\n%p\n%p\n%p\n%p\n%p
    \n%p\n%p\n%p\n%p\n%p\n%p\n%p\n%p\n%p\n%p\n%p\n%p\n%p\n%p\n%p\n%p\
    n%p\n%p\n%p\n%p\n%p\n%p\n%p\n%p\n%p\n%p\n%p\n%p\n%p\n%p\n%p\n%p\n
    %p\n%p\n%p\n%p\n%p\n%p\n%p\n%p\n%p\n%p\n%p\n%p\n\n";
3 printf(input);
4 [...]
```

Figure 18 shows the execution of the code snipped:

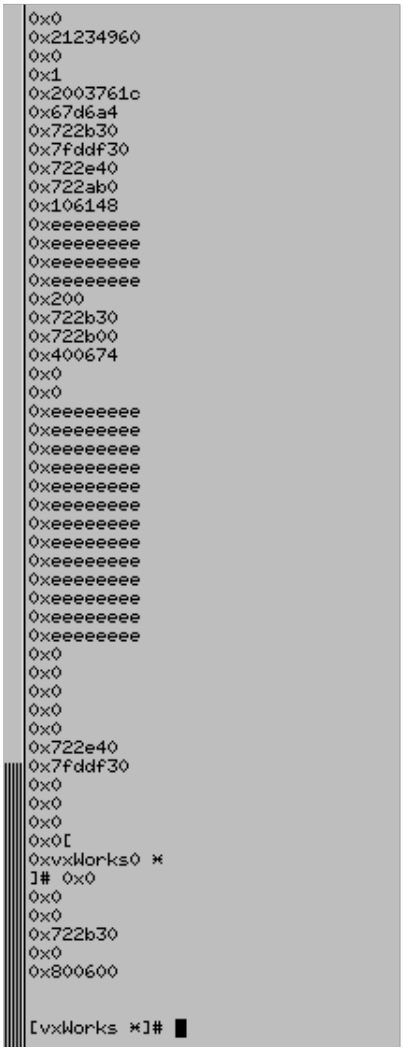

Figure 18: Format String Test – Stack Leak

This example reveals multiple pointers of the stack including "`0xeeeeeeee`", which is used to fill up the stack at the boot process of the VxWorks. There is no explanation from *Wind River* why the stack of the RTOS is filled up with this pointer.

The next step is to leak local variables because static data and locally

stored data can contain valuable information[59]. To accomplish this, the start-up script is modified in the following way:

```
[...]
char * a = "AAAA";
char * input;
char * b = "BBBB";
input = "\n%p\n%p\n%p\n%p\n%p\n%p\n%p\n%p\n%p\n%p\n%p\n%p\n%p\n%p\n%
    p\n%p\n%p\n%p\n%p\n%p\n%p\n%p\n%p\n%p\n%p\n%p\n%p\n%p\n%p\n%p\n%p
    \n%p\n%p\n%p\n%p\n%p\n%p\n%p\n%p\n%p\n%p\n%p\n%p\n%p\n%p\n%p\n%p\
    n%p\n%p\n%p\n%p\n%p\n%p\n%p\n%p\n\n";
char * c = "CCCC";
printf(input);
char * d = "DDDD";
[...]
```

Four character pointers, `a`, `b`, `c`, and `d` are introduced, which contain static data. The `printf()`-call is used before the variable `d` is introduced. Therefore, only content of `a`, `b`, and `c` should be displayed on the stack because the variable `d` is not part of the stack at this time stamp. Figure 19 displays the stack at this point in time:

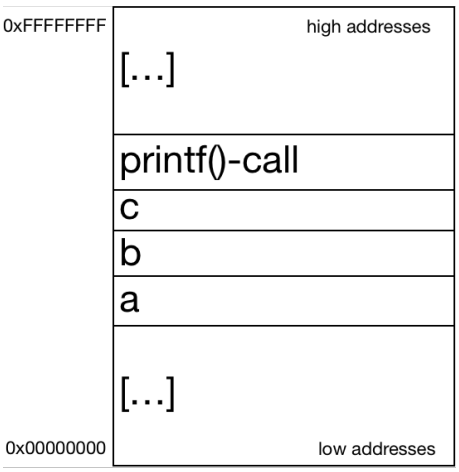

Figure 19: Stack to the defined time stamp

Figure 20 displays the corresponding output:

[59]Instead of the used test values, the variable can contain valuable information such as passwords or keys.

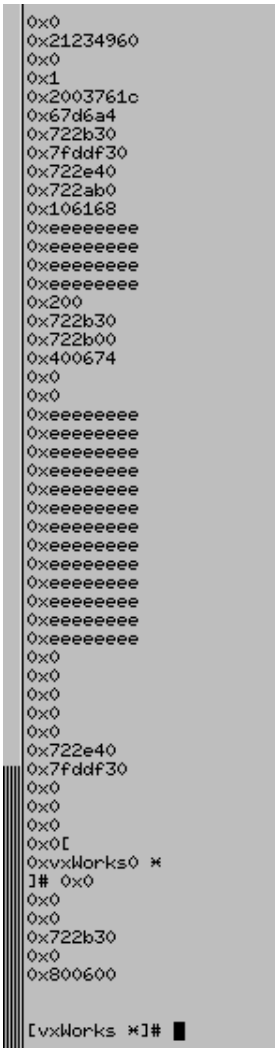

Figure 20: Format String Test – Leak local data

Since `%p` is used within the `printf()`-call, the content of the variables `a`, `b`, and `c` should be displayed as pointers. Therefore, `0x41414141`, `0x42424242`, and `0x43434343` (which is the corresponding pointer representation) should be visible. As seen in Figure 20, non of the mentioned pointer is apparent.

The next step is to increase the number of printed addresses to display more of the stack. Although, the expected values still do not show.

From this it follows that the VxWorks 7 Memory Management Unit

(MMU) does not allow the user[60] to read the content of the pointers. The command `vmContextShow` displays, which memory areas are protected from reading, writing, or execution. Figure 21 shows that all memory areas are read-protected. This is visible in the *PROT*-column where *R* means read protected, *W* means write protected, and *X* means execution protected. Therefore, the content of the values is not visible.

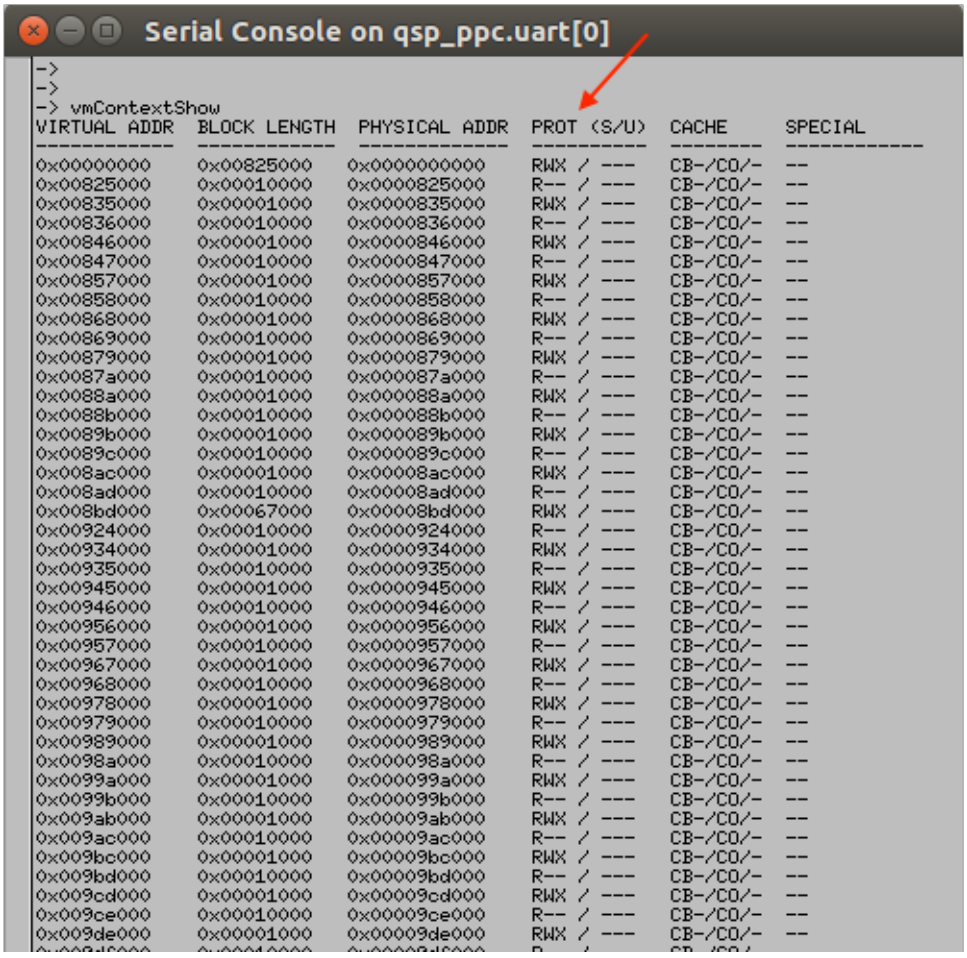

Figure 21: Memory Management Unit Output

After reading data, the next step is to write data. The following code segment is used to write data on the stack. The format specifier `%n` is used

[60]Default user with administrative privileges.

to write the number of characters which have been written so far [82].

```
1 [...]
2 char * new_input = "AAAA%n%n%n%n%n%n%n%n%n%n%n%n%n%n%n%n%n%n\n\n";
3 printf(new_input);
4 [...]
```

This write of data cause the system to crash. VxWorks 7 can not read the data and jump to the provided addresses[61], which causes a reboot of the system. The RTOS will reboot all the time, since the start-up script is executed for every start. A manual stop of the system is necessary. The use of an arbitrary format string write in the start-up script causes a downtime of the system, which can be interpreted as a DoS. This weakness is critical for an aircraft system because it relies on the functionality of all services.

A protection mechanisms is presented by the security researcher Crispin Cowan, who introduces a framwork called *FormatGuard* to protect a computer system from format string vulnerabilities. The resreacher describes the framwork the following way: "*The FormatGuard defends against format bug attacks by comparing the actual arguments presented to* `printf` *against the number of arguments called by the format string.*" [83]. This means that the format specifiers are compared against the provided variables. If both are not equal, the *FormatGuard* will raise an error and stop further execution. This mechanism could be used for avionic software. This decision depends on the use case and the available resources.

Furthermore, VxWorks 7 provides advanced stack protection while the kernel hardening features are enabled. Therefore, the next step is to test the RTOS with those features enabled.

First, the leak of local data is tested. To accomplish this, the start-up script is modified in the same way as mentioned before. As expected and already visible, VxWorks 7 does not leak local data.

In the first place, the arbitrary write of data on the stack caused a DoS, while no kernel hardening features were enabled. Since those features are now enabled, VxWorks 7 does not reboot all the time because it ejects the RAM file system and stops processing. A manual reboot is necessary to start the RTOS again. This prevents further damage from happening.

[61] The provided address is `0x41414141`, which is "`AAAA`" in ASCII representation.

As a result of this, VxWorks 7 does provide protection against format string bugs, vulnerabilities, and attacks. The kernel hardening features shelter the RTOS against a DoS, which results by a vulnerable stack write. Furthermore, aid during software development is provided by *Wind River's* Workbench regarding format string bugs. Therefore, the evaluation parameter is set to "*Mechanism exist*" and "*Full protection*".

7.6 E6 – Secure Cryptography Algorithm

Secure cryptography algorithm harden the system and make sure that sensible data is protected and can be stored securely.

This chapter introduces cryptography libraries mentioned in the *Cryptography Libraries Programmer's Guide* [57] for VxWorks 7. The RTOS offers plenty of cryptography libraries to provide encryption and decryption of data. This collection is based on OpenSSL 1.0.2 and supports multiple cryptography algorithms. Besides the algorithms from OpenSSL 1.0.2 including Tripple DES, MD5, SHA, RSA (as shown in Figure 22), and more, VxWorks 7 includes the Advanced Encryption Standard (AES) key wrap and the AES cypher-based message authentication code (CMAC).

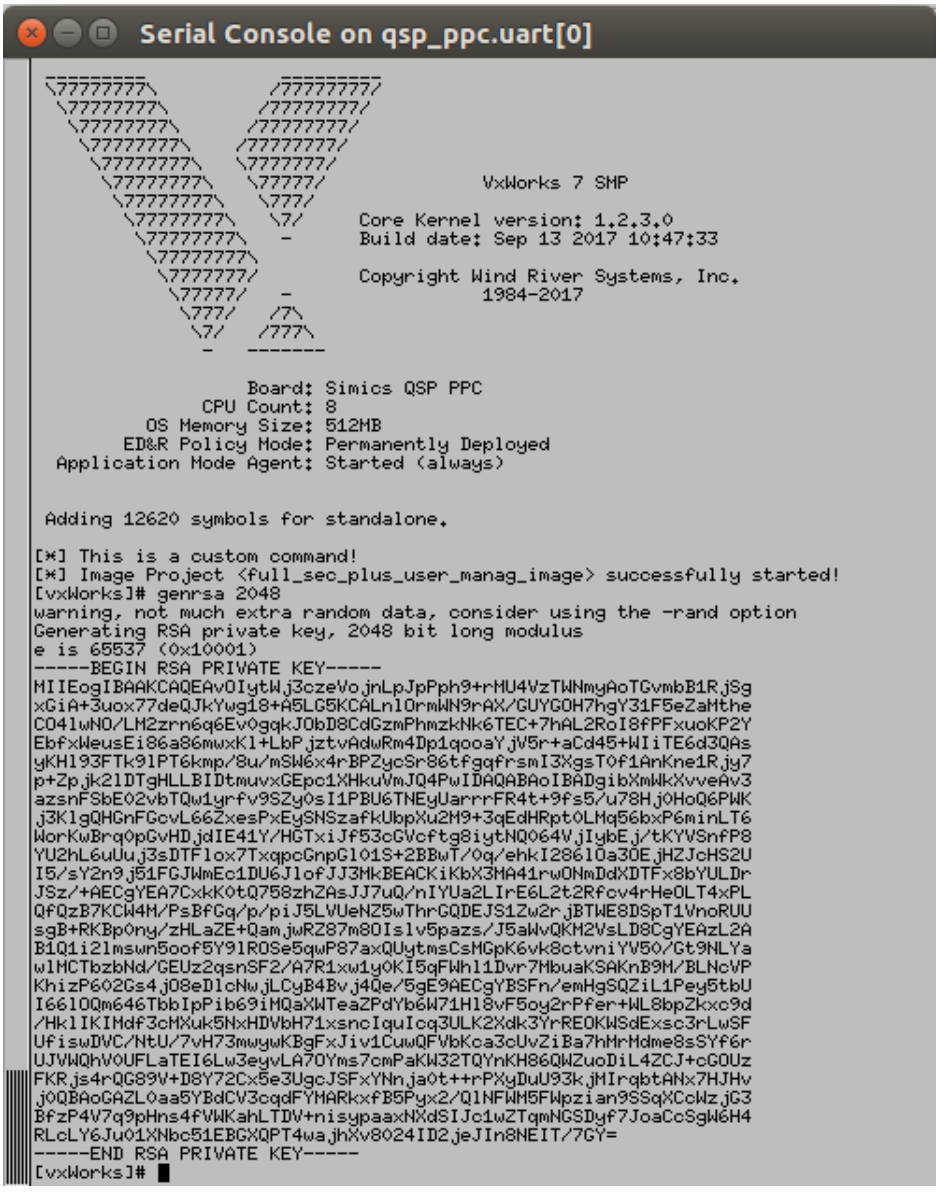

Figure 22: Generation of a 2048 bit RSA Private Key

VxWorks 7 can be configured to provide the mentioned algorithms. To add those facilities to the real-time operating system, multiple options in the VxWorks Source Build (VSB) and the VxWorks Image Project (VIP) must be set as shown in Figure 23. The RTOS does not support the cryptography

libraries by default. Therefore, VxWorks 7 with default settings is insecure relating to secure cryptographic algorithms.

Figure 23: VxWorks Cryptography Configuration for VSB

More libraries and configurations lead to more cryptographic security, but the VxWorks image is not lightweight any more and needs more storage. The VxWorks 7 binary with cryptography libraries enabled is 27.595954 MB. Without those libraries the size is 27.496172 MB. This is a difference of 99.792 KB (0.362237%), which can be a lot for an embedded device[62].

In conclusion, VxWorks 7 provides a good range of cryptographic algorithms, which are easy to use and publicly considered robust and secure such as RSA [84] and AES [85]. From this it follows to set the score for this evaluation parameter to "*mechanism exist*" and "*full protection*".

7.7 E7 – Storage of Sensitive Data

Sensitive data is information whose leakage can result in damage to the overall system, the affected entities[63], or exploitation of the RTOS. It can either be user specific data including passwords or cryptographic based information such as decryption keys. Exposure of sensitive data can lead to an unintended disclosure of information, which can result in security issues.

This parameter is evaluated using a manual audit.

[62] The development board P4080DS from NXP has a default storage of 16 MB [42].

[63] For example: other aircraft, drones, ground stations, or communication channels.

Within VxWorks 7, cryptographic keys and passwords are stored in multiple sections of the RTOS. On the one hand, in the Key Store and on the other hand in the Secrets Repository.

Public key pairs generated by cryptographic algorithms inlcuding Rivest Shamir Adleman (RSA) [84], Digital Signature Algorithm (DSA) [86], and Elliptic Curve Cryptography (ECC) [87] are stored in the Key Store.

For the storage of passwords and symmetric encryption keys[64] [88] VxWorks 7 implements the Secrets Repository. The data is encrypted in separate files, contrary to the Key Store, which does not encrypt user passwords and uses clear text to represent the information.

Key-encrypting password (KEP) providers are used to encrypt data in the Secrets Repository. VxWorks 7 supports the hardware based Trusted Platform Module 2.0 (TPM), software based Binary Large Object (BLOB), costume KEP providers[65], and a software based password obfuscater.

TPM 2.0 seals strings including passwords with TPM 2.0 hardware, which is defined by the *ISO/IEC 11889-1* standard [89].

BLOB keys are used to save storage space and database transactions processing time since large binary data is stored in single entities within a database system [90]. A BLOB key provides confidentially and integrity protection due to the use of headers and private-public key algorithms [57].

VxWorks' password obfuscator uses the Password-Based Cryptography Specification (PKCS Number 5) [91], which covers key derivation functions, encryption schemes, message-authentication schemes, and more.

The cryptographic libraries for VxWorks 7 provide a trusted key store for public key certificates. Those certificates provide the ability to check the ownership of a public key value. Such a value can either be trusted or untrusted. This technology can be used to validate the integrity of software including packages, add-ons, plug-ins, and updates by adding a certificate to a software module.

[64] Generated by algorithms including Data Encryption Standard (DES) and CAST-128.
[65] This can be a self implemented encryption password.

In summary VxWorks 7 provides two strong mechanisms to store sensitive data. The Key Store is more lightweight but does not encrypt data by default and the Secrets Repository is not that lightweight but does encrypt data. It is up to the developer and system architect to use them properly and implement them as needed. This evaluation parameter is marked within it "*Mechanism exists*"-column. Since the Key Store contains passwords ins clear-text, the "*Only extenuates*"-column will be marked.

7.8 E8 – Command Injection (*CWE-77/78*)

CWE 77-78 defines command injections as external input received by an application, which can lead to the execution of unintended commands [92]. C functions like `system(command)` can execute system specific commands. If an attacker is able to gain control[66] over the `command` variable, he has the ability to execute system commands. This results in a security issue for the prone application.

Secure software development can prevent command injection attacks and vulnerabilities. This topic is further discussed in Chapter 5.2. This parameter is examined using manual code audits.

The following Python code displays the misuse of a system command on a 32-bit UNIX system:

```
1 import sys
2 # import call which allows to execute system commands
3 from subprocess import call
4
5 # read a file and write the result to standard output
6 cat = "cat"
7
8 # get the user input
9 user_input = sys.argv[1]
10
11 # execute system command
12 call([cat, user_input])
```

If the script is executed, the following output is printed:

```
1 admin@lab: ~ $ python call.py file
2 This file contains some text.
```

The Python script works as intended since it just prints the content of the file name `file`. However, since the attacker controls the input he is able to misuse the script and inject more system commands:

[66] For example by exploiting a buffer overflow vulnerability.

```
admin@lab: ~ $ python call.py file;echo "attacker was here"
This file contains some text.
attacker was here
```

The attacker is able to execute all possible system commands, from printing some text up to reading passwords.

The RTOS does not explicitly use system functions in any available start-up or kernel scripts. Certainly, the symbol table contains the standard C function `system`, as displayed in Figure 24.

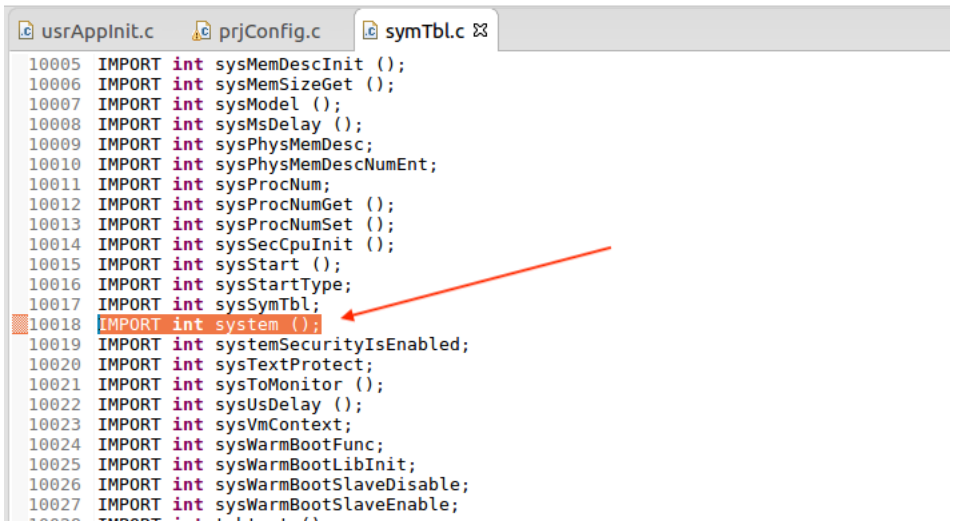

Figure 24: Symbol Table System Function

VxWorks 7 does not provide any publicly mentioned security features preventing command injections. However, every custom developed application, which uses system commands, can be prone to this attack. A focus on user input during software development is necessary to prevent a command injection vulnerability. Chapter 5 displays common security mistakes by programmers. Therefore, the evaluation parameter will be placed within the "*Mechanism does not exist*"-column.

7.9 E9 – Integrity check of external Software

Within a publicly released presentation [93], the U.S. Air Force describes the process for updating software of the weapon system of their military fighter

jets. Software developed within their *Aircraft System Software Development Source Center* is checked multiple times using virus scanners[67]. The transfer and update mediums are CD, DVD, or Floppy. The result of a manual audit is that there is no indication for an integrity check of external software such as separate code signing in the described process. If the software on such a disk would be changed in a malicious way and furthermore could not be identified as a virus, an attacker would be able to compromise the system without any notice.

Operating systems and framworks can provide code signing and integrity checks using separate technologies and mechanisms. For example, *Microsoft* Windows offers the ability to sign code with trusted certificates to indicate the integrity of the signed source code [94]. If the code would change after the signing process, the certificate is not valid any more.

Before the upload of software to a military aircraft, the software should be checked. It is even more secure to check the software on the aircraft itself. The RTOS can provide such mechanisms. VxWorks 7 does not provide any technology to check the integrity of external added software.

Although, *Wind Rivers* Workbench offers the developer to edit the start-up script. From this it follows that the signing and validating of certificates is realizable on a VxWorks 7 system. A possible code signing implementation can look like this:

1. Securely transfer and store the public/private key pair.
2. Hash the original code using the public key with a hash function such as SHA.
3. Store the hash from the original code in the Key Store or Secrets Repository.

On the other hand, a possible code verification implementation can look like this:

1. Decrypt the signed code using the private key.
2. Compare the hash of the signed code with the hash of the original code.
3. If both are equal, nothing has changed. If they are not equal, the code has changed.

[67] No information about the virus scanners is publicly accessible.

The certificates can securely be stored within the Secrets Repository, which is further described in Chapter 7.7.

Since VxWorks 7 does not offer any mechanism and technology to check the integrity of external software, but *Wind Rivers* Workbench provides the ability to edit the start-up script, a mark will be set within the "*Mechanism does exist*"-column and "*Only extenuates*"-column.

7.10 Summary

Evaluation Parameter E1: Chapter 7.1 describes the threat of improper authentication. By default, VxWorks 7 does not provide protective mechanisms against those threats by using publicly available credentials. Although, the RTOS offers custom login and password policies, which can be added manually by the developer to secure the system regarding this topic.

Evaluation Parameter E2: The evaluation of Chapter 7.2 describes privilege management. VxWorks 7 offers a robust and secure management of privileges using two modes, privileged and user mode, with separated access control. Furthermore, by default two file systems, ROM and RAM, provide dedicated storage of sensitive data. Although, the correct assignment of privileges regarding resources is up to the developer and system architect.

Evaluation Parameter E3: Chapter 7.3 specifies the protection against malware. Such protection technologies can be code signing or signature handling. VxWorks 7 does not provide any mechanisms or frameworks to protect the system against this threat. The RTOS grants the ability to edit start-up scripts, which make these technologies realizable, certainly with high effort.

Evaluation Parameter E4: This parameter examines buffer overflow vulnerabilities. It is possible to smash the stack using a program with a common overflow of a defined buffer. VxWorks 7 crashes, if it notices the overflow. Although, the MMU protects the RTOS by ejecting the file system and preventing further damage. Shelter against buffer overflows are based on secure software development in the corresponding language as described in Chapter 5.2 since the kernel hardening features offer protection regarding this threat.

Evaluation Parameter E5: String vulnerabilities are defined in Chapter 7.5. *Wind River's* Workbench offers warnings and errors regarding those vulnerabilities while developing software. The RTOS itself does not provide any mechanism, which protects VxWorks 7 from string vulnerabilities. It is

possible to DoS the RTOS exploiting a string vulnerability. Secure software development is described in Chapter 5.2 and can prevent those vulnerabilities and attacks from occurring.

Evaluation Parameter E6: Chapter 7.6 describes secure cryptography algorithms. This topic can easily result in threats or attacks, if not treated well enough. VxWorks 7 offers a rich number of cryptographic libraries and algorithms, which are easy to use and enabled by default. Furthermore, at this point in time all algorithms and libraries are considered as robust and secure by the public opinion.

Evaluation Parameter E7: The storage of sensitive data is examined in Chapter 7.7. VxWorks 7 introduces two storage locations for sensitive data. On the one hand the Secrets Repository and on the other hand the Key Store. Both provide the ability to store data in multiple representation formats. Although, by default information within the Key Store is stored in clear-text, which makes it readable for every user on the system.

Evaluation Parameter E8: This Chapter 7.8, specifies command injection attacks. The result of an evaluation relating to this parameter is the absence of any security mechanism provided by VxWorks 7. Command injections are possible since these functions are part of the symbol table. Secure software development with proper input handling can prevent these attacks.

Evaluation Parameter E9: Chapter 7.9 examines the integrity check of external software. VxWorks 7 does not provide any technology to implement signing of source code or applications. Although, through the editing of the start-up scripts it is possible to implement this mechanism.

Table 7.1 displays the filled and final matrix, which is the outcome of the security evaluation for VxWorks 7 for avionic systems.

Category	**Threat, Vulnerability or Attack**	**Mechanism exists**			**Mechanism does not exist**	**Evaluation**
		Name	Full protection	Only extenuates		
Authentication	Improper Authentication (CWE-287)	Authentication and Login Policies		X		E1
Access Control	Privilege Management (CWE-264/266/269)	User database and robust privilege assignment	X			E2
Data Protection	Malware Protection				X	E3
Resources management	Buffer Overflow (CWE-121/122/123)	Guard Stack Protection		X		E4
	String vulnerabilities (CWE-133/134)	Kernel hardening features		X		E5
Cryptography	Secure cryptography algorithm	Cryptographic libraries	X			E6
	Storage of sensitive information	Key Store and Secrets Repository		X		E7
Other	Command Injection (CWE-77/78)				X	E8
	Integrity check of external software	Edit Start-up Script		X		E9

Table 7.1: Final Security Evaluation Matrix

Chapter 8

Conclusion and Outlook

The purpose of the security evaluation for the real-time operating system VxWorks 7 for avionic systems is to determine if the RTOS offers protective mechanisms, which shelter the system against common threats, attacks, and vulnerabilities.

Secure software development is necessary to offer a sheltered avionic system. Although, not every developer provides a sufficient security awareness and knowledge. At this point, the real-time operating system can protect the system from security critical software errors as well as mistakes. Therefore, common threats, vulnerabilities, and attacks are tested to present technologies and mechanisms provided by the RTOS to protect an avionic system.

The result is the generation of the final evaluation matrix. VxWorks 7 shows good basic security mechanisms to protect the system against common threats, attacks, and vulnerabilities in the relevant categories as well as the viewed area[68]. Although, issues appeared in malware protection and command injection vulnerabilities, where no mechanism or technology is provided. Security mechanisms offer basic protection against issues within the privilege management, buffer overflows, and secure cryptography algorithms. Other technologies can (through implementing appropriate mechanisms) protect the system against the viewed threats, attacks, and vulnerabilities. It was even possible to cause a DoS of the RTOS by exploiting string vulnerabilities.

Further safeguarding mechanisms and technologies can be implemented to harden a VxWorks 7 system.

[68] Avionic systems.

Continued research can be performed to ensure security for VxWorks 7. This can be accomplished by performing penetration tests or further security evaluations on specific areas of the RTOS such as authentication technologies or key management on a used and implemented system. Further, security assessments of secure implementations and configurations for specific circumstances can be evaluated to ensure the creation of secure VxWorks 7 images.

Additionally, security evaluations and penetration tests can be performed on physical hardware devices, since this thesis discusses a virtual environment. Those tests can produce additional benefit to provide secure systems.

Bibliography

[1] I. O. for Standardization, "Information technology – programming languages – c," standard, International Organization for Standardization, Geneva, CH, 12 2011. Last accessed 06. August 2017.

[2] I. O. for Standardization, "Information technology – programming languages – c++," standard, International Organization for Standardization, Geneva, CH, 12 2014. Last accessed 06. August 2017.

[3] F. O. for Information Security (BSI), "Die lage der it-sicherheit in deutschland 2017," resreport, Federal Office for Information Security (BSI), Godesberger Allee 185-189, 53175 Bonn, Aug. 2017. Item Number: BSI-LB17/506.

[4] "CVE Details apple mac osx - vulnerability trends over time." `http://www.cvedetails.com/product/156/Apple-Mac-Os-X.html?vendor_id=49`, 2017. Last accessed 01. August 2017.

[5] "CVE Details microsoft windows 10 - vulnerability trends over time." `http://www.cvedetails.com/product/32238/Microsoft-Windows-10.html?vendor_id=26`, 2017. Last accessed 01. August 2017.

[6] "CVE Details linux kernel - vulnerability trends over time." `http://www.cvedetails.com/product/47/Linux-Linux-Kernel.html?vendor_id=33`, 2017. Last accessed 01. August 2017.

[7] A. West, "Nasa study on flight software complexity." `https://www.nasa.gov/pdf/418878main_FSWC_Final_Report.pdf`, 2001. Last accessed: 14. November 2017.

[8] H. Teso, "Aircraft hacking – practical aero series." `https://conference.hitb.org/hitbsecconf2013ams/materials/`

D1T1%20-%20Hugo%20Teso%20-%20Aircraft%20Hacking%20-%20Practical%20Aero%20Series.pdf, Apr. 2013. Last accessed: 16. November 2017.

[9] U.S.Code, "Information security." https://www.law.cornell.edu/uscode/pdf/uscode44/lii_usc_TI_44_CH_35_SC_III_SE_3542.pdf, Jan. 2012. Published in U.S. Code Title 44, Chapter 35, Subchapter III, § 3542(1)(A)/(B)/(C).

[10] R. Shirey, "Internet security glossary," 2000.

[11] C. Eckert, *IT-Sicherheit: Konzepte, Verfahren, Protokolle*. Oldenbourg Verlag, 7 ed., 2012.

[12] K. D. Morgan, "The rtos difference," *BYTE*, vol. 17, pp. 161–172, Aug. 1992.

[13] W. Stallings, *Operating Systems: Internals and Design Principles*. Upper Saddle River, NJ, USA: Prentice Hall Press, 6th ed., 2008.

[14] WindRiver, "Vxworks product." https://www.windriver.com/products/vxworks/, 2017. Last accessed 05. August 2017.

[15] WindRiver, "Wind river vxworks 653 platform," techreport, Wind River Systems, Inc., Aug. 2017. Last accessed 27. November 2017.

[16] I. O. for Standardizatioin, "Information technology – programming languages – ada," standard, International Organization for Standardizatioin, Geneva, CH, 12 2012. Last accessed 06. August 2017.

[17] AdaCore, "Gnatbench – ada development plug-in for eclipse and wind river systems workbench." https://www.adacore.com/gnatpro/toolsuite/gnatbench, 2017. Last accessed 27. November 2017.

[18] WindRiver, "Workbench 4 - getting started," tech. rep., Wind River Systems, Inc., Aug. 2017. Last accessed 27. November 2017.

[19] G. C. Rafael V. Aroca, "A real time operating systems (rtos) comparison," 2009.

[20] A. K. Sood, "Digging inside the vxworks os and firmware – the holistic security," 2011.

[21] Rapid7, "Shiny old vxworks vulnerabilities." `https://blog.rapid7.com/2010/08/02/shiny-old-vxworks-vulnerabilities/`, Aug. 2010. Last accessed: 05. October 2017.

[22] M. Rhodes-Ousley, *Information security the complete reference*, vol. 2. McGraw-Hill, 2013.

[23] J. Xie, H. R. Lipford, and B. Chu, "Why do programmers make security errors?," in *2011 IEEE Symposium on Visual Languages and Human-Centric Computing (VL/HCC)*, pp. 161–164, Sept 2011.

[24] LockheedMartinCorporation, "Joint strike fighter – air vehicle – c++ coding standards – for the system development and demonstration program," Dec. 2005.

[25] J. Wakely, "Gcj." `https://gcc.gnu.org/wiki/GCJ`, July 2017.

[26] C. Pohl and H. Hof, "Secure scrum: Development of secure software with scrum," *CoRR*, vol. abs/1507.02992, 2015.

[27] R. C. Seacord, *Secure Coding in C and C++*. Addison-Wesley Professional, 2nd ed., 2013.

[28] Aleph1, "Smashing the stack for fun and profit." `http://phrack.org/issues/49/14.html`, 11 1996. Last accessed 06. September 2017.

[29] *libc(7) Linux Programmer's Manual*, 12 2016.

[30] A. Follner, *On Generating Gadget Chains for Return-Oriented Programming*. phdthesis, Technical University Darmstadt - Faculty Computer Science, Darmstadt, Germany, Dec. 2017.

[31] J. Foster, *Sockets, Shellcode, Porting, and Coding: Reverse Engineering Exploits and Tool Coding for Security Professionals*. Elsevier Science, 2005. Last accessed 06. September 2017.

[32] M. Mol, "Rosetta code- integer overflow." `https://rosettacode.org/wiki/Integer_overflow`, 03 2017. Last accessed 25. August 2017.

[33] T. Granlund, "The gnu multiple precision arithmetic libary," techreport, Free Software Foundation, Inc., 12 2016.

[34] R. T. C. for Aeronautics (RTCA), "Software considerations in airborne systems and equipment certification," standard, RTCA, Inc., Washington, DC 20036-3816 USA, 12 2011.

[35] H. Afzali and H. Mokhtari, *A Quantitative Model of Operating System Security Evaluation*, pp. 345–353. Berlin, Heidelberg: Springer Berlin Heidelberg, 2012. Last accessed 06. September 2017.

[36] "CVE Details common vulnerabilities and exposures databaes." `http://www.cvedetails.com/`, 2017. Last accessed 30. Juli 2017.

[37] M. Corporation, *CWE - Common Weakness Enumeration - A Community-Developed Dictionary of Software Weakness Types*. MITRE Corporation, 2.11 ed., May 2017. CWE is a Software Assurance strategic initiative sponsored by the National Cyber Security Division of the U.S. Department of Homeland Security.

[38] FLEXnet, "Flexnet licensing end user guide." `http://www.media.3ds.com/support/simulia/public/flexlm108/EndUser/chap6.htm`, 2005. Last accessed 21. September 2017.

[39] Hawlett-Packard, Intel, Microsoft, Phoenix-Technologies, and Toshiba, "Advanced configuration and power interface specification," tech. rep., Hawlett-Packard Corporation, Intel Corporation, Microsoft Corporation, Phoenix Technologies Ltd. and Toshiba Corporation, Nov. 2013. Revision 5.0 Errata A.

[40] R. Jaworowski, "Flattend device trees for embedded freebsd," *The FreeBSD Project*, 2010.

[41] H. Zimmermann, "Osi reference model - the iso model of architecture for open systems interconnection," *IEEE Transactions on Communications*, vol. 28, pp. 425–432, April 1980.

[42] Freescale, "Qoriq multicore processor development – p4080 development system," tech. rep., NXP, 2013.

[43] P. O. Rino Micheloni, Giovanni Campardo, *Memories in Wireless Systems*. Springer Berlin Heidelberg, 2008.

[44] E. Harari, "Electrically erasable non-volatile semiconductor memory," 09 1978. United States Patent 4,115,914.

[45] SanDisk, "Sandisk sd card - product manual," tech. rep., SanDisk Corporation, 11 2004.

[46] G. Likely and J. Boyer, "A symphony of flavours: Using the device tree to describe embedded hardware," in *Proceedings of the Linux Symposium*, vol. 2, pp. 27–37, 2008.

[47] Freescale, "P4080 development system user's guide," techreport, NXP, 2010.

[48] J. Engblom, "Inside a synthetic simulated platform." `http://blogs.windriver.com/engblom/2012/07/inside-the-simics-qsp.html`, July 2012.

[49] M. Corporation, *Improper Authentication*. MITRE Corporation, 2.11 ed., May 2017. CWE is a Software Assurance strategic initiative sponsored by the National Cyber Security Division of the U.S. Department of Homeland Security.

[50] K. Zetter, "Weak password brings 'happiness' to twitter hacker." `https://www.wired.com/2009/01/professed-twitt/`, Jan. 2009. Last accessed: 18. October 2017.

[51] R. T. C. for Aeronautics (RTCA), "Airworthiness security process specification," standard, RTCA, Inc., 08 2014.

[52] R. T. C. for Aeronautics (RTCA), "Security do-355 information security guidance for continuing airworthiness," standard, RTCA, Inc., 06 2014.

[53] R. T. C. for Aeronautics (RTCA), "Airworthiness security methods and considerations," standard, RTCA, Inc., 09 2014.

[54] I. O. for Standardization, "Information technology – security techniques – evaluation criteria for it security – part 1: Introduction and general model," standard, International Organization for Standardization, Geneva, CH, 01 2014.

[55] I. O. for Standardization, "Information technology – security techniques – evaluation criteria for it security – part 2: Security functional components," standard, International Organization for Standardization, Geneva, CH, 05 2011.

[56] I. O. for Standardization, "Information technology – security techniques – evaluation criteria for it security – part 3: Security assurance components," standard, International Organization for Standardization, Geneva, CH, 05 2011.

[57] WindRiver, "Vxworks 7 – cryptography libraries programmer's guide," tech. rep., Wind River Systems, Inc., 2017.

[58] Cirt, "Default passwords." https://cirt.net/passwords, 2017. Last accessed: 16. October 2017.

[59] DefaultPasswordList, "Default password list – displaying 1812 passwords of total 1812 entrys.." http://www.defaultpassword.com, 2017. Last accessed: 16. October 2017.

[60] Phenoelit, "Default password list." http://www.phenoelit.org/dpl/dpl.html, 2007. Last accessed: 16. October 2017.

[61] B. Krebs, "Equifax breach: Setting the record straight." https://krebsonsecurity.com/2017/09/equifax-breach-setting-the-record-straight/, Sept. 2007. Last accessed: 16. October 2017.

[62] KrebsonSecurity, "Ayuda! (help!) equifax has my data!." https://krebsonsecurity.com/2017/09/ayuda-help-equifax-has-my-data/, Sept. Last accessed: 09. October 2017.

[63] M. Corporation, *Improper Privilege Managment*. MITRE Corporation, 2.11 ed., May 2017. CWE is a Software Assurance strategic initiative sponsored by the National Cyber Security Division of the U.S. Department of Homeland Security.

[64] J. H. Saltier and M. P. Schroeder, "Protection of information in computer systems," *IEEE CSIT Newsletter*, vol. 3, pp. 19–19, Dec 1975.

[65] J. Powers, R. Smith, Z. Korkmaz, and H. Ahmed, "Whistlist malware defense for embedded control system devices," in *2015 Saudi Arabia Smart Grid (SASG)*, pp. 1–6, Dec. 2015.

[66] McAfee, "Mcafee labs threats report june 2017," tech. rep., June 2017. Last accessed: 11. October 2017.

[67] L. H. Newman, "The ransomware meltdown experts warned about is here." https://www.wired.com/2017/05/ransomware-meltdown-experts-warned/, 2017. Last accessed: 19. October 2017.

[68] A. Greenberg, "Petya ransomware epidemic may be spilleover from cyberwar." https://www.wired.com/story/petya-ransomware-ukraine/, June 2017. Last accessed: 19. October 2017.

[69] D. Venugopal and G. Hu, "Efficient signature based malware detection on mobile devices," *Mobile Information Systems*, vol. 4, no. 1, pp. 33–49, 2008.

[70] S. Oh, W. Go, and T. Lee, "A study on the behavior-based malware detection signature," in *Advances on Broad-Band Wireless Computing, Communication and Applications*, pp. 663–670, Springer International Publishing, oct 2016.

[71] AV-Comparatives, "Malware protection test – file detection test with execution," tech. rep., Apr. 2017. Last accessed: 11. October 2017.

[72] lockheedmartin, "Software you wish you had: Inside the f-35 supercomputer." `http://lockheedmartin.com/us/news/features/2015/072015-f35-supercomputer.html`, July 2015. Last accessed: 19. October 2017.

[73] F. Daryabar, A. Dehghantanha, and N. I. Udzir, "Investigation of bypassing malware defences and malware detections," in *2011 7th International Conference on Information Assurance and Security (IAS)*, pp. 173–178, Dec 2011.

[74] M. Christiansen, "Bypassing malware defenses," *SANS Institute InfoSec Reading Room*, May 2010. Last accessed: 11. October 2017.

[75] M. I. Al-Saleh and B. Shebaro, "Enhancing malware detection: Clients deserve more protection," *Int. J. Electron. Secur. Digit. Forensic*, vol. 8, pp. 1–16, Dec. 2016.

[76] N. Aaraj, A. Raghunathan, and N. K. Jha, "A framework for defending embedded systems against software attacks," *ACM Trans. Embed. Comput. Syst.*, vol. 10, pp. 33:1–33:23, May 2011.

[77] "CVE Details linux kernel vulnerability statistics." `http://www.cvedetails.com/product/47/Linux-Linux-Kernel.html?vendor_id=33`, 2017. Last accessed 25. October 2017.

[78] "CVE Details microsoft windows 10 vulnerability statistics." `http://www.cvedetails.com/product/32238/Microsoft-Windows-10.html?vendor_id=26`, 2017. Last accessed 25. October 2017.

[79] "CVE Details qnx rtos vulnerability statistics." `http://www.cvedetails.com/product/2084/QNX-Rtos.html?vendor_id=436`, 2017. Last accessed 25. October 2017.

[80] "CVE Details wind river vxworks vulnerability statistics." `http://www.cvedetails.com/product/15063/Windriver-Vxworks.html?vendor_id=95`, 2017. Last accessed 25. October 2017.

[81] K. Lhee and S. J. Chapin, "Buffer overflow and format string overflow vulnerabilities," *Softw., Pract. Exper.*, vol. 33, no. 5, pp. 423–460, 2003.

[82] *PRINTF(3) – BSD Library Functions Manual*, 2009.

[83] C. Cowan, M. Barringer, S. Beattie, G. Kroah-Hartman, M. Frantzen, and J. Lokier, "Formatguard: Automatic protection from printf format string vulnerabilities," in *Proceedings of the 10th Conference on USENIX Security Symposium - Volume 10*, SSYM'01, (Berkeley, CA, USA), pp. 15–15, USENIX Association, 2001.

[84] R. L. Rivest, A. Shamir, and L. Adleman, "A method for obtaining digital signatures and public-key cryptosystems," *Commun. ACM*, vol. 21, pp. 120–126, Feb. 1978.

[85] D. Hoon Lee and X. Wang, *Advances in Cryptography - ASIACRYPT 2011 17th International Conference on the Theory and Application of Cryptography and Information Security, Seoul, Sout Korea, December 4-8, 2011,. P*, vol. 7073. Jan. 2011.

[86] J. Buchmann, "The digital signature algorithm (dsa)," *Neurosurgical Review*, 21 2001.

[87] V. Kapoor, V. S. Abraham, and R. Singh, "Elliptic curve cryptography," *Ubiquity*, vol. 2008, pp. 7:1–7:8, May 2008.

[88] Ayushi, "A symmetric key cryptographic algorithm," in *International Journal of Computer Applications - Volume 1 - No. 15*, 2010.

[89] I. O. for Standardization, "Information technology – trusted platform module library – part 1 architecture," standard, International Organization for Standardization, Geneva, CH, 08 2015. Last accessed 06. August 2017.

[90] M. Shapiro and E. Miller, "Managing databases with binary large objects," in *16th IEEE Symposium on Mass Storage Systems in cooperation with the 7th NASA Goddard Conference on Mass Storage Systems and Technologies (Cat. No.99CB37098)*, pp. 185–193, 1999.

[91] A. R. K. Moriarty, B. Kaliski, "Pkcs number 5: Password-based cryptographic specification version 2.1," Tech. Rep. 8018, Internet Engineering Task Force (IETF), Jan. 2017. Last accessed 07. November 2017.

[92] M. Corporation, *Improper Neutralization of Special Elements used in a Command (Command Injection)*. MITRE Corporation, 2.11 ed., May 2017. CWE is a Software Assurance strategic initiative sponsored by the National Cyber Security Division of the U.S. Department of Homeland Security.

[93] D. R. Patel, "Managing cybersecurity risk in weapon systems." Last accessed 07. November 2017, Mar. 2017.

[94] Microsoft, "Get a code signing certificate." `https://docs.microsoft.com/en-us/windows-hardware/drivers/dashboard/get-a-code-signing-certificate`, Apr. 2017. Last accessed: 08. November 2017.