A Systems Approach to Software Security in Aviation

by

Jonas Helfer

M. Sc. in Computer Science
Ecole Polytechnique Fédérale de Lausanne, 2011

Submitted to the department of Electrical Engineering and Computer Science in Partial Fulfillment of the Requirements for the Degree
of

Master of Science in Electrical Engineering and Computer Science

at the

Massachusetts Institute of Technology

February 2016

Signature of Author .... Signature redacted ....
Department of Electrical Engineering and Computer Science
January 27, 2016

Certified by .... Signature redacted ....
Nancy G. Leveson
Professor of Aeronautics and Astronautics
Thesis Supervisor

Accepted by .... Signature redacted ....
Leslie A. Kolodziejski
Professor of Electrical Engineering and Computer Science
Chair of the Committee on Graduate Students

Abstract

Field Loadable Software in commercial aviation is indispensable for vital avionics functions yet its security has never been studied in depth. Due to the recent introduction of wireless software loading capabilities and Internet-connected in-flight entertainment systems along with several high-profile information security breaches in other sectors, the security of Field Loadable software has come under closer scrutiny.

Conventional information systems security analysis approaches focus on finding and preventing vulnerabilities in the implementation of a system, but they are not designed to include the organizational "soft" components of a system.

The aim of this thesis is to provide a comprehensive security analysis of Field Loadable Software that includes organizational aspects in order to find existing vulnerabilities and propose security constraints that would fix the vulnerabilities or prevent them from being exploited.

A novel safety approach from safety engineering, called Systems Theoretic Process Analysis (STPA) was adapted and used to perform the security analysis of Field Loadable Software in commercial aviation.

The analysis produced a simple systems model for Field Loadable Software and found that current regulations and practices are not sufficient: there several significant vulnerabilities in the way Field Loadable Software is currently designed and distributed. However, the analysis also showed that the vulnerabilities could be removed with the addition of simple technical measures and security constraints.

Acknowledgements

First of all, I'd like to thank my advisor, Dr. Nancy Leveson, who introduced me to STAMP and STPA and taught me to see systems from a new perspective. Her support, advice and guidance was invaluable in the process of researching and writing this thesis.

Furthermore, I'd like to thank all my colleagues in the Systems Research Laboratory for their support and constructive feedback. Special thanks to John Thomas. His deep knowledge and keen insight is an inspiration to me and without his advice I would not be where I am today.

The research for this thesis was made possible by a grant from the Federal Aviation Administration. I'd like to give special thanks to Peter Skaves, Tong Vu and Raymond de Cerchio from the FAA for their feedback and support.

Table of Contents

List of Figures

List of Tables

Chapter 1 Introduction

Until recently, security played only a minor role in aircraft systems designs. Engineers were mindful of safety, however, and the aviation industry's accident numbers have dropped accordingly over the last 50 years [1]. After 9/11, significant efforts were made to improve physical security, but regulation pertaining to information systems security, such as DO-355, has yet to be adopted by the Federal Aviation Administration.

Avionics started to be used in commercial aircraft after WWII, but it wasn't until the early 80's that aircraft were built with components whose software could be updated without replacing the hardware—so-called field loadable software. The earliest versions of field loadable software systems in the 80's used floppy disks to apply updates. More recently, software suppliers and airlines have started distributing software over the Internet [2] and transmitting it to aircraft via wireless networks at airports (GateLink). Without security measures in place, critical avionics software in commercial aircraft could be compromised by a determined adversary. In addition to the aforementioned new technologies used for software distribution, other possible routes for an adversary to compromise the integrity of an aircraft are present in the form of in-flight entertainment systems and passenger WiFi.

Safety-critical software, such as avionics software used on commercial aircraft, is subject to much higher standards of correctness than other software. In particular, it must be possible to ascertain with an acceptable level of confidence that the software meets airworthiness requirements, meaning that it conforms to its type design and is safe for operation [3]. Software verification and validation was recognized as a very challenging problem a long time ago [4]. While static analysis can prove certain properties of a software's behavior, this alone is not enough to guarantee system safety [5, 6]. In fact, the most commonly used approaches in industry make little use of formal methods and rely instead on following a rigorous development process and performing extensive testing [7, 8, 3]. However, there is little to no research comparing the efficiency of different software methodologies with respect to preventing safety-critical flaws, and software testing is still largely an art and not a science [9].

The software currently used in commercial aircraft has to adhere to numerous standards intended to ensure safety, and the industry has managed to significantly reduce accident rates over the last 50 years [1]. However, the security of aviation software has only recently started to gain attention, and is thus not very well understood [10, 11]. Operators and suppliers in the aviation industry assume that they are safe from attacks because their systems do not run on commodity hardware. However, the operators of industrial control systems held similar beliefs, until the first real attacks proved them wrong [12,13].

While there are similarities between software safety approaches and software security approaches, software security requires that the software not only be resistant to inadvertent actions by benevolent actors, but also resistant to advertent actions by a determined opponent [14]. Security therefore requires different means than safety.

Commercial enterprises usually have very few incentives to invest in security until the occurrence of an incident, and it is possible that the aviation industry is no different in this regard. Security features are generally not part of the functional aspects of the system that customers are willing to pay for. The challenge that companies face is therefore to invest 'just enough' into security to convince their customer that their product is secure, but no more than what security incidents might cost the company [15].

This thesis summarizes the results of applying STPA (Systems Theoretic Process Analysis) to the safety and security of aviation information systems [14,16].

Security is not an intrinsic property of any individual system component; It is an emergent property of the whole system. Unsafe or insecure behavior at the system level can be predicted by modeling the system as a hierarchy of controllers that provide input and feedback to each other. A safe/secure system must ensure that the process it controls remains within a specified set of safe/secure states (and clear of unsafe/insecure states) by applying the proper control inputs to said process [5]. Rather than limiting the analysis to the software itself, this approach recognizes that insecure system states may arise from the interaction between people and software, and not just the software itself. It therefore includes not only the software but also organizations, individuals and hardware in the model of the system being analyzed. Such an approach has been successfully applied to other complex systems, and this thesis shows that it works for aviation software as well [17].

Chapter 2 Background

2.1 Field Loadable Software

In aviation, the term Field Loadable Software describes any piece of software that can be exchanged or updated without the need for replacing the hardware Line Replaceable Unit (LRU). Since their first introduction to commercial aviation in the 1970s, field loadable software units have become the de-facto standard. The main driving force behind this trend is the reduction of development and maintenance costs associated with avionics. Development is cheaper because the software runs on standardized hardware (even commercial-off-the-shelf hardware in some cases) and maintenance costs are lower because updating software takes less time and specialized equipment than replacing an LRU.

The earliest forms of Field Loadable Software used in commercial aviation were loaded with floppy disks. The different pieces of FLS used on the Boeing 777 for example were distributed in floppy disks. Over the years, floppy disks were replaced with portable media of higher capacity, such as CDs and portable hard drives. Currently, the state of the art is to send the software to airlines over the Internet. The airline in turn distributes it to the aircraft over wireless networks (GateLink). Once the software is on the aircraft, mechanics can send it to the LRU whose software needs to be updated.

It is likely that at some point in the near future, aircraft manufacturers and airlines will work together to reduce maintenance cost by altogether removing the need for manual interventions by ground personnel during the software updating process.

2.2 Aviation Systems Security

When avionics software was first introduced in the 70's, security was only a concern insofar as the computers running the software had to be protected from physical sabotage. This was mainly due to two reasons: (1) Most computers weren't networked and the Internet did not exist yet, and (2) avionics software and hardware used to be custom made for aviation. The systems were similar to more widely used commodity hardware and software in the PC mass-market, but they had some significant differences as well. Attacks that affected widespread commodity hardware and software were thus not a major concern for aviation.

In recent years, both of these "defenses" have become significantly weaker or have been removed altogether, explaining an increased focus on aviation information systems security: More and more avionics systems are networked; they communicate with each other and some of them communicate with airline or other ground services, which in turn are connected to the Internet. Passenger entertainment and in-flight Internet have become common, and they offer another vector of attack into avionics systems. In order to reduce costs, newer avionics systems are increasingly built using commercial off the shelf (COTS) hardware and software, which makes them cheaper to build and maintain, but also potentially vulnerable to common types of attacks carried out over the Internet. Using COTS hardware and software enables engineers to reuse code, tools and parts, but it also enables attackers to do the same, thus dramatically reducing the cost of developing exploits against aviation. It is even imaginable that aircraft could be affected by attacks not specifically directed against them, such as a virus that spreads itself through a vulnerability in a common operating system.

For all these reasons, the FAA, aircraft manufacturers and airlines have increased investments into aviation systems information security. They are trying to analyze avionics systems to determine whether they are vulnerable, and if so what measures can be taken to make them more secure.

2.2.1 Overview of current regulation

While there is current FAA guidance and regulation pertaining to software development (DO-178C [18], AC 20-115C [19] and Order 8110.49[20]), new guidance for cyber-security (DO-326A [21] and DO-355 [22]) has not yet been accepted or required as a means of compliance with airworthiness standards. It is unclear whether the current safety guidelines in DO-178C pertaining to field loadable software are sufficient to guarantee security as well, or whether DO-178C needs to be updated. ARINC Report 835 [23] describes the digital signature based process used by Boeing and Airbus to distribute software to their new aircraft.

DO-178C, DO-355 and DO-326 do not address physical security of aviation equipment as this is considered to fall under the responsibilities of the Transportation Security Administration (TSA) and the Department of Homeland Security (DHS).

The following sections of this document contain only a high-level overview of parts of DO-178C, DO-355, DO-326 and Order 8110.49. For a more detailed understanding, the reader is referred to the original documents.

2.2.2 RTCA DO-178C

RTCA DO-178C - Software Considerations in Airborne Systems and Equipment Certification outlines the process of developing software for aircraft. It also contains a section of guidance for the safety of field loadable software that reads as follows:

The safety-related requirements associated with the software data loading function are part of the system requirements. If the inadvertent enabling of the software data loading function could cause erroneous loading of software parts, then a safety-related requirement for the software data loading function should be specified in the system requirements.

System safety considerations relating to field-loadable software include:

* Detection of corrupted or partially loaded software

* Determination of the effects of loading the inappropriate software

* Hardware/software compatibility

* Software/software compatibility

* Aircraft/software compatibility

* Inadvertent enabling of the field loading function

* Loss or corruption of the software configuration identification display.

Unless otherwise justified by the system safety assessment process, the detection mechanism for partial or corrupted software loads should be assigned the same failure condition or software level as the most severe failure condition or software level associated with the function that uses the software load.

If a system has a default mode when inappropriate software is loaded, then each partitioned component of the system should have safety-related requirements specified for operation in this mode which address the potential failure condition.

The software loading function, including support systems and procedures, should include a means to detect incorrect software and hardware, and should provide protection appropriate for the function involved. If the software consists of multiple configuration items, their compatibility should be ensured.

If software is part of an airborne display mechanism that is the means for ensuring that the aircraft conforms to a certified configuration, then that software should either be developed to the highest software level of the software to be loaded, or the system safety assessment process should justify the integrity of an end-to-end check of the software configuration identification.

2.2.3 FAA Order 8110.49

FAA order 8110.49 is a guide for Aircraft Certification Service field offices and

Designated Engineering Representatives on how to apply RTCA DO-178B for approving software used in airborne computers. Chapter 5 of order 8110.49 concerns itself exclusively with the approval of field loadable software, and provides more detail than DO-178B. Below is an extract of chapter 5:

5.2 APPROVAL OF FLS.

The following procedures should be carried out by the certification authority as part of the authorization process for the approval of FLS.* Confirm that the software meets the objectives of RTCA/DO-178B or another acceptable means of compliance, as agreed to between the applicant and the certification authority.

* Confirm that the considerations outlined in RTCA/DO-178B, Section 2.5, have been addressed.

* Confirm that the software and hardware configurations were verified together during the verification process (that is, the software must be installed on the target computer in which the approval was granted).

* Confirm that the applicant has a configuration management process in place to assure that the installation configuration (that is, the software part number, the hardware part number, the aircraft or engine model, and the aircraft or engine serial number combinations, as applicable) is the same configuration that was approved during the authorization process.

* If redundant parts on the aircraft or engine are field-loadable, confirm that the applicant has defined the following: (1) the requirements for intermixing different software loads on the parts, (2) requirements for partially successful and partially unsuccessful loads, and (3) the aircraft or engine dispatchability effects of successful and unsuccessful loads on redundant parts.

* Confirm that there is a process in place to ensure that the software loaded is the

software approved and that the software has not been corrupted (for example, verification with an appropriate data transfer integrity check, such as a CRC).

[etc.]

5-3. FLS INSTALLATION CONSIDERATIONS.

The approved FLS may be installed on the aircraft via Service Bulletin, Engineering Change Request, or other FAA-approved means.

The approved means vary, depending on the method for granting approval. Whether the FLS approval is through TC, ATC, STC, ASTC, TSO authorization, or some other approval process, the document used to install the FLS should be approved by the certification authority and should specify the following elements:

* The aircraft and hardware applicability and intermixability allowances for redundant systems software loading.
* Verification procedures to assure that the software was correctly loaded into an approved and compatible target computer and memory devices.
* Any post-load verification and/or test procedures required to show compliance to the guidelines specified in this chapter.
* Actions to be taken in the event of an unsuccessful load (for example, prohibit dispatch of the aircraft).
* Approved loading procedure or reference to approved loading procedure.
* Maintenance record entry procedures required to maintain configuration control.
* Reference to Aircraft Flight Manual, Aircraft Flight Manual Supplement, or Operator's Manual, as appropriate.

2.2.4 RTCA DO-355 – Information Security Guidance For Continuing Airworthiness

RTCA DO-355, prepared by SC-216 describes its purpose as follows: "This document provides guidance for the operation and maintenance of aircraft and for organizations and personnel involved in these tasks. It shall support the responsibilities of the Design Approval Holder (DAH) to obtain a valid airworthiness certificate and aircraft operators to maintain their aircraft to demonstrate that the effects on the safety of the aircraft of information security threats are confined within acceptable levels."

DO-355 does not describe the process used for the distribution of software but refers to ARINC 667-1, ARINC 827 and ARINC 835 instead. It only provides guidance for the operational aspects of software distribution, such as the management of tools and media, the management of personnel and the handling of security incidents. It also describes means to ensure the security of network access points on the aircraft, including operational aspects of protecting the access points from unauthorized access.

DO-355 attempts to enumerate all possible vulnerabilities and describes ways to mitigate them. It is the opinion of the author of this paper that this sort of approach to security – finding all the vulnerabilities and plugging them faster than an adversary can exploit them – cannot possibly succeed in the long run. While this sort of tactical defense is necessary to protect legacy systems that are exposed to new threats, the aviation community should focus on developing systems that do not have that many vulnerabilities in the first place. For instance, instead of taking operational measures to protect the integrity of the software at every step between the supplier and the aircraft, the system should be designed such that end-to-end verification is possible, thus making it unnecessary to rely on all parties that handle the software in between.

2.2.5 RTCA DO-326A – Airworthiness Security Process Specification

RTCA DO-326A, prepared by RTCA SC-216 and EUROCAE WG-72 aims to specify an acceptable means to demonstrate security of an aircraft system for the purposes of certification: "This guidance material is for equipment manufacturers, aircraft manufacturers, and anyone else who is applying for an initial Type Certificate (TC), and afterwards (e.g. for Design Approval Holders (DAH)), Supplemental Type Certificate (STC), Amended Type Certificate (ATC) or changes to Type Certification for installation and continued airworthiness for aircraft systems."

The purpose of the Airworthiness Security Process (AWSP) is to establish that, when subjected to unauthorized interaction, the aircraft will remain in a condition for safe operation (using the regulatory airworthiness criteria). To accomplish this purpose, the Airworthiness Security Process:

- Establishes that the security risk to the aircraft and its systems are acceptable per the criteria established by the AWSP, and
- Establishes that the Airworthiness Security Risk Assessment is complete and correct.

As part of the security process, DO-326A requires defining a security scope, performing various security risk assessments, designing a security architecture and finally performing a security verification. Each step is described in sufficient detail so it could be implemented by a supplier, manufacturer or an airline.

Of all the documents presented here, DO-326A is the most extensive one and seems to be inspired by best practices from the government and other industries. While the described process will undoubtedly increase security, it is questionable whether the provided security will be sufficient given the information security track record of government and industry who have adopted similar practices. Rather than deriving security requirements from external threat scenarios, which are inherently hard to predict, the requirements should be derived from a thorough understanding of the system and its components.

2.2.6 FAA Advisory Circular AC 20-115

This FAA advisory circular recognizes RTCA DO-178C and RTCA DO-33X supplements to

2.2.7 ARINC Report 835

This report by Aeronautical Radio Inc. describes the standards used by Boeing and Airbus for applying digital signatures to field loadable software for their new aircraft (B-787, A-380). It is intended as guidance for other organizations who wish to implement digital signature schemes. The two main purposes of digital signatures for field loadable software parts that the report describes are integrity and authentication independent from the media used for transporting the software (network, portable media, etc.). Non-repudiation is mentioned as a secondary goal.

The schemes used by both Boeing and Airbus rely on a Public Key Infrastructure (PKI) and add a header file to the software being signed, but they differ in the type and number of signatures added to the software.

In the standard used by Airbus the software supplier will sign software before it leaves their facility. Before loading software on an aircraft, the airline is responsible for checking the validity of the supplier's signature. This requires a connection to an OCSP Server (on-line certificate status protocol).

For the Airbus A-380, the load is verified by equipment on the aircraft itself (how it does this is not specified in the report, but presumably it happens by checking the software's signature against the root CA's public key which is pre-loaded on the aircraft). This requires either that the aircraft is connected to an OCSP, or that the pre-placed root certificate is changed whenever a private key is lost (which is impractical).

In some cases, the signature file used in Airbus' scheme may include a timestamp, which enables keeping the validity of signatures that were created before a certificate was revoked. In order to add a timestamp to the signature, the supplier must connect to a trusted timestamp server when generating the signature.

In the standard used by Boeing, each party along the supply chain of the software will apply their own signature to the software. The original signature comes from the supplier, but the aircraft manufacturer (i.e. Boeing) will add its own signature after verifying the signature of the supplier. This signature is called an approval signature. When the airline receives a software part, it will verify Boeing's signature before applying its own. In this way, the aircraft need only be pre-placed with the certificate of the airline, and an on-line check of certificate validity becomes less important. However, the standard used by Boeing envisions that each organization (software supplier, aircraft manufacturer and airline) operate their own Certificate Authority.

2.3 Systems Theoretic Process Analysis

Systems Theoretic Process Analysis (STPA) [24] is a hazard analysis technique based on the STAMP model that has proven to be a powerful tool to study emergent properties of systems such as safety and security.

STPA is based on systems theory. Systems theory was developed in the 1950's, based on the realization that the reductionist approach common in science did not always provide the right tools to understand complex systems and that a more holistic approach was needed. It was first invented to deal with the complexity in biological systems, but since then it has been used in engineering as well. The complexity of man-made systems has increased greatly over the last 50 years, especially since the commercialization of microprocessors. While earlier mechanical systems had limited interactions and could be fully analyzed and understood, the systems built nowadays have thousands of components and so many potential interactions that they cannot possibly be fully analyzed.

Consequently, STAMP and STPA were motivated by several technological and societal trends in safety engineering:

- The fast pace of technological change means that technology is changing faster than the engineering methods invented to cope with it.

- The introduction of digital technology into almost all new systems has changed the nature of accidents that occur in those systems by introducing new failure modes. Many of the safety approaches that worked for electromechanical components – such as redundancy - are ineffective at controlling hazards that arise from failure of software components.

- Software has allowed the creation of vastly more complex systems than was possible before. The components of such systems are coupled in complex ways that defy understanding by all but a few experts, and even they may have difficulty predicting all its potential behaviors.

- Relationships between humans and automation are becoming more complex as automation takes over tasks that used to be carried out by human operators, thus moving the humans into higher-level decision making. These changes give rise to new types of human errors, such as mode confusion. However, many accidents blamed on human operators could be more accurately described as arising from flawed systems and interface design.

Traditional accident analyses are based on the “chain of events” model. Under this model, accidents are the result of a chain of events. Beyond the proximal physical events involved in or leading directly to the accident, the decision of which events to include in the analysis is subjective. Determining the “root cause” of the accident therefore amounts to nothing else than deciding on when to stop including even earlier events in the chain.

According to Leveson, accident investigations have two potential goals:

1) Assign blame for the accident

2) Understand why it happened so that similar accidents can be prevented in the future

While the first goal may be necessary from a legal perspective, only the second goal helps engineer safer systems for the future. Event-based models may be sufficient for assigning blame, but they will likely miss critical non-proximal factors that contributed to the accident.

Systemic causes for accidents, such as unsafe work procedures, are often present long before an accident occurs. In order to prevent similar accidents in the future, the systemic causes need to be uncovered by an accident analysis. The proximal event that is usually identified as the cause is simply the event that triggered the accident, but if the systemic causes are not addressed, a different event may trigger a similar accident in the future.

To address this challenge, STAMP is a new model for accidents that is based on control theory. Under this model, safety is considered a control problem: Accidents occur when the system does not adequately control the process it is built to control. This can occur for a number of reasons, such as: component failures, component interaction and events outside of the range the system was designed to handle. The goal of an STPA analysis is therefore to find scenarios in which the system is unable to keep the process from entering hazardous states in which accidents can occur.

2.3.1 STPA in practice

The initial step when performing an STPA analysis is to define the system losses and the hazards that can lead to these losses. Usually, these are defined at a very high level. As a rule of thumb, there are between 2 and 5 system-level losses and less than 10 high-level system hazards leading to the system losses.

It is very important that the hazards are actually under the system's control. If the hazards cannot be controlled by the system, then either the model of the system needs to be re-scoped and its boundary expanded, or the hazards need to be redefined. If on the other hand only a part of the system is necessary to control the hazards, the system boundary may have been drawn too wide.

Examples of system-level losses for a hypothetical rail transportation system:

1. Injury or death of passengers in a train
2. Extensive damage to train or static infrastructure
3. Injury or death of personnel or bystanders

Examples of system-level hazards:

1. Derailing

2. Train cannot be stopped in an emergency
3. Collision with another train or person

Once the losses and hazards are determined, a hierarchical control structure is drawn that includes all the controllers and control actions in the system that are necessary to keep the controlled process out of its hazardous states. Controllers are arranged in a hierarchy, such that the controlled process is at the bottom.

The hierarchical control structure serves to illustrate the system architecture. For every controller, each control action needs to be studied to determine whether it could potentially be an unsafe control action. There are four ways in which a control action can be unsafe:

1. A hazard occurs because a control action was provided when it should not have been
2. A hazard occurs because a control action was not provided when it should have been
3. A hazard occurs because a control action was stopped too soon or applied for too long
4. A hazard occurs because a control action was applied too soon or too late.

There is some redundancy in these four categories, but it has proven to be useful in eliciting unsafe control actions that would have otherwise not been thought of.

Once the table or list of unsafe control actions for each controller is established, the next step of STPA consists in finding scenarios that could lead to these unsafe control actions in order to propose safety constraints that will keep these scenarios from happening. Finding such safety constraints should be the final output of an STPA analysis, and no analysis is complete without them. Scenarios should be generated in a methodical way by people with expert knowledge in the system. A control loop diagram for each controller can be used to assist in the process.

2.3.2 STPA-Sec

STPA for security (STPA-Sec) is an extension to STPA proposed by Dr. William Young, a colonel in the US Air Force, during his time as a PhD student under Prof. Leveson. It adds the following elements to the classical STPA analysis:

1. The realization that STPA can be applied to security equally well as safety.

2. Producing a high-level system description becomes an explicit first step in the analysis. The high-level system description includes answering the following questions: **What** does

the system do? **How** does the system do it? **Why** does the system do what it does?

3. Before listing losses and hazards, a "mission" for the system is explicitly written down. The mission defines the initial state of the system, the end state of the system as well as all intermediate states necessary to attain the end state. The mission can then be divided up into stages, each of which is supposed to bring the system from one state to the next.

The third addition was clearly made with military applications in mind, but it can be applied in other areas as well. Instead of looking at the mission as having a fixed and limited duration, it can go on indefinitely, and all mission stages can be occurring simultaneously. If interpreted in that sense, the mission description is simply a functional description of the system or a more detailed answer to the question: **how** does the system do what it does?

STPA-Sec differs from most other security analysis methods in that it uses a strategic approach instead of a tactical approach. By forcing the analyst to think about the high-level system goals, the method makes it more likely that simpler and more radical solutions will be found where they are possible. Contrast this with more tactical approaches which ask the analyst to identify and fix vulnerabilities. Such approaches are more likely to lead to a patchwork of fixes, often with the ultimate result that security is only enhanced until the next vulnerability is found.

Step 2 of STPA-Sec is largely identical to STPA, but the focus is on potential exploits rather than failures or interactions. Figure 1 illustrates one way in which a control-loop can be split between the cyber and physical domains. In this case the actuators and sensors translate digital signals to physical signals and vice-versa. While figure 1 shows a control-loop that is split between the cyber domain and the physical domain, some control loops in the system may be entirely in the cyber or the physical domain.

The control-loop is simply an enumeration of several possible exploits and should not be considered an exhaustive list. In particular, it should not be used as a "checklist" for step 2 of STPA-Sec.

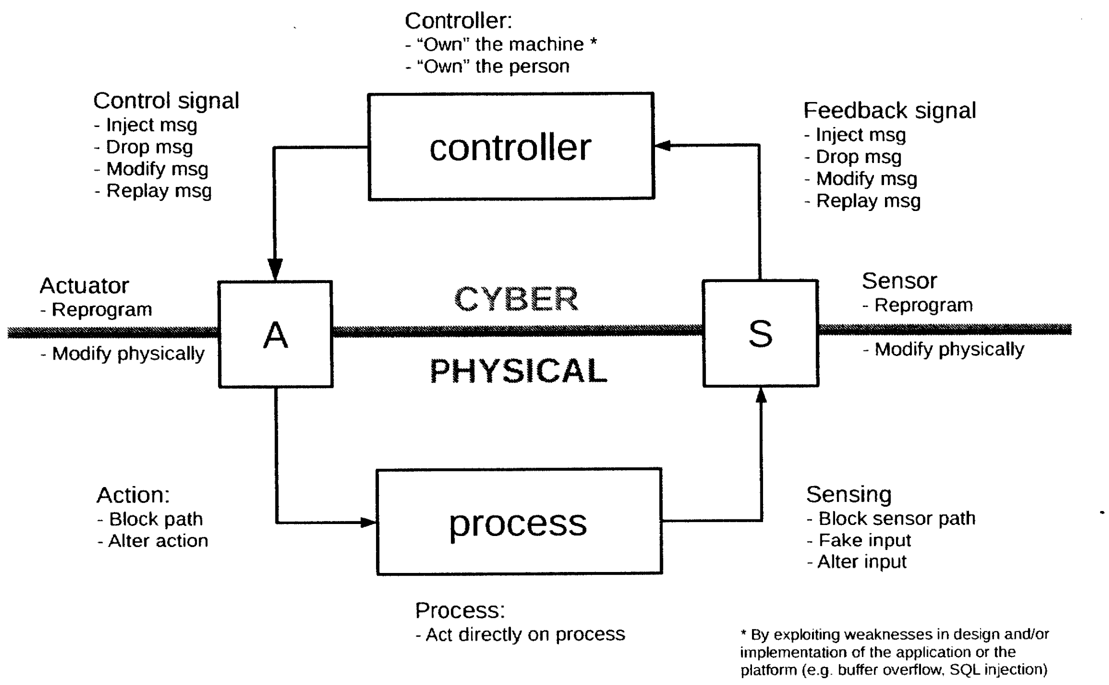

Figure 1: The cyber control-loop shows possible ways to exploit security weaknesses in the system design.

Chapter 3 Analysis of Field Loadable Software Distribution

This chapter presents the steps and results of the STPA analysis performed for the security of Field Loadable Software processes in commercial aviation. A similar analysis was performed for General Aviation. This chapter sometimes refers to it, but it does not present the complete results of that analysis (they can be found in the appendix). If not otherwise stated, the text in this chapter refers to commercial aviation. A comparison between the control structure and UCAs of commercial aviation and general aviation is made at the end of this chapter.

The analysis in this chapter follows the structure of STPA-Sec. Choices for how to abstract the system are justified where necessary, but for an explanation of the steps involved in an STPA analysis, the reader is referred to the sub-chapters on STPA earlier in this thesis.

3.1 System Purpose

The purpose of Field Loadable Software processes is to (*what* the system does)

keep Field Loadable Software safe and secure

by means of (*how* the system does what it does)

1. Developing safe and secure software and updates
2. Distributing and installing only safe and secure software on all relevant aircraft
3. Monitoring field loadable software in operation for flaws and attempted manipulation

in order to contribute to (*why* the system does what it does)

safe and efficient commercial aviation

3.2 Losses and Hazards

Losses:

L1: Death or injury to pilot, passengers or bystanders

L2: Damage to equipment

Hazards:

H1: Loss of aerodynamic/structural control → L1,L2

H2: Controlled flight into terrain → L1,L2

H3: Collision with other aircraft → L1, L2

Secondary hazards (concern Field Loadable software only):

H3: Unsafe or insecure software is installed on the aircraft → H1, H2, H3

H4: Software is installed improperly on the aircraft → H1, H2, H3

3.3 Control Structure

While the high-level control structure in the general aviation analysis contains only 5 controllers, commercial aviation has 9 controllers. See Figure 2. The *Foreign regulatory agency*, *Airline management* and *MRO management* controllers do not have a counterpart in general aviation. The *repair station* controller on the other hand is not found in commercial aviation, as most of its roles are now done by the *airline mechanics* and *MRO mechanics.*

FAA:

- Ensure that only safe and secure software and aircraft configurations get FAA type certificates
- Issue Airworthiness Directive for any safety-critical flaws in FLS that have a type certificate
- Ensure that airline operations follow FAA regulations
- Certify pilots

Foreign regulatory agency:

- Ensure that MROs are certified only in accordance with bilateral agreement/treaty

(Software) Supplier:

- Develop safe and secure field loadable software and accompanying manuals
- Distribute software and updates to airlines
- Deliver service bulletins to FAA and all airlines affected by an FLS flaw

Airline management:

- Provide manuals and training to pilots
- Ensure pilots are qualified and fit to fly
- Ensure that aircraft stays airworthy by sending aircraft to MRO for regular maintenance
- Ensure that pilot's work schedule is in accordance with local regulation
- Ensure that airline mechanics carry out work in accordance with local regulation

Airline Mechanic:

- Install and update Field Loadable Software on aircraft according to instructions provided by MRO

- Update EGPWS (FLS) terrain database

- Make sure aircraft is airworthy before takeoff

MRO management:

- Ensure that maintenance work is carried out as specified in contract with airline management

- Supervise MRO mechanics

MRO Mechanic:

- Install and update Field Loadable Software on aircraft according to instructions provided by MRO

- Update EGPWS (FLS) terrain database

Pilots:

- Operate aircraft safely (including checking configuration before takeoff)

- Follow EGPWS warnings

EGPWS (included as a generic example of an FLS component):

- Issue warning when aircraft is in danger of flying into terrain or obstacles

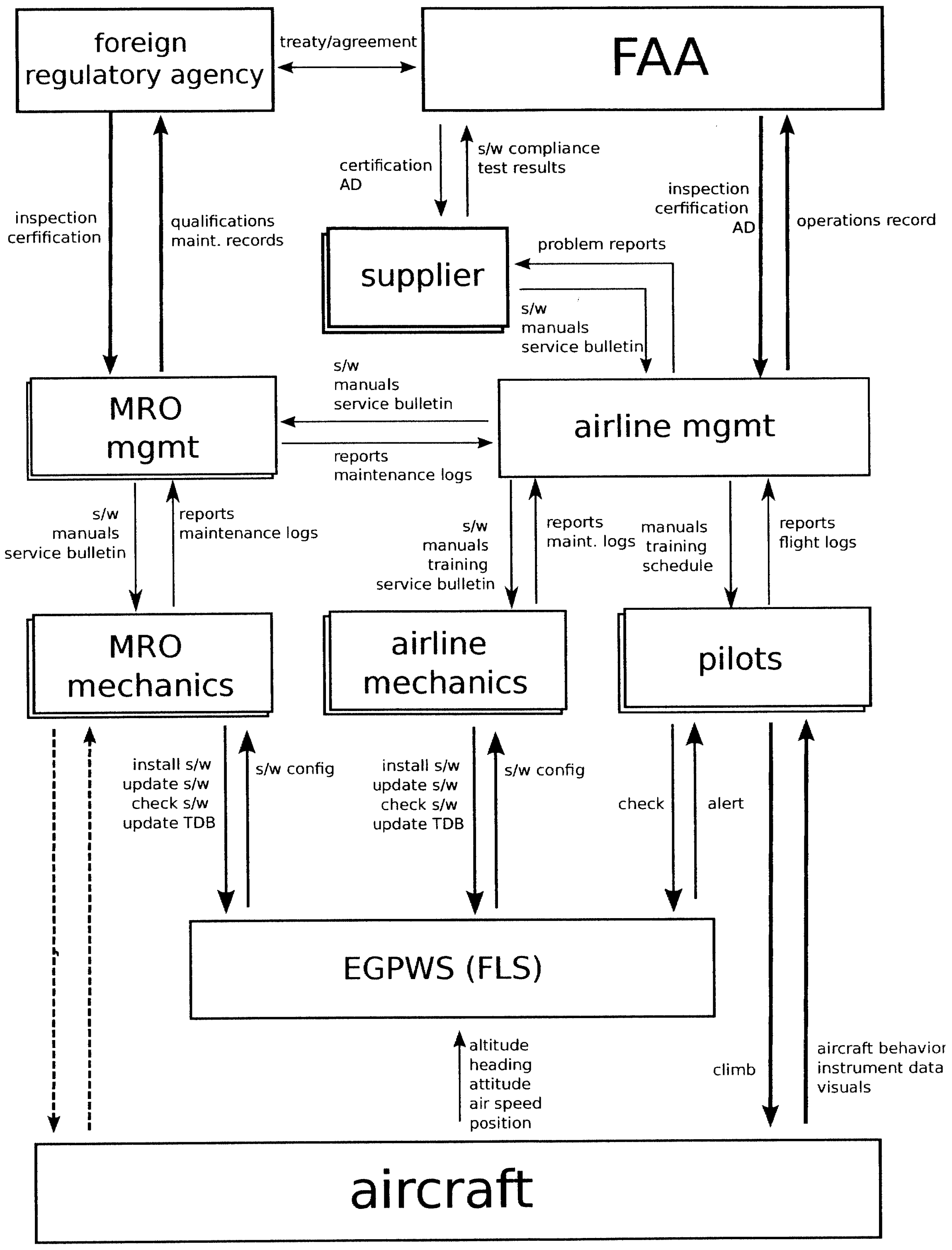
foreign regulatory agency
treaty/agreement
FAA
inspection cerfification
qualifications maint. records
certification AD
s/w compliance test results
inspection cerfification AD
operations record
supplier
problem reports
s/w manuals service bulletin
s/w manuals service bulletin
MRO mgmt
airline mgmt
reports maintenance logs
s/w manuals service bulletin
reports maintenance logs
s/w manuals training service bulletin
reports maint. logs
manuals training schedule
reports flight logs
MRO mechanics
airline mechanics
pilots
install s/w update s/w check s/w update TDB
s/w config
install s/w update s/w check s/w update TDB
s/w config
check
alert
EGPWS (FLS)
altitude heading attitude air speed position
climb
aircraft behavior instrument data visuals
aircraft

Figure 2: Control structure for FLS in commercial aviation

The control structure in Figure 2 reflects a setup that is quite common in commercial aviation: some maintenance is carried out by the airline itself, while other maintenance is carried out by so-called Maintenance Repair Organizations (MROs), which are often located overseas. These MROs are not under direct oversight of the FAA. Instead, they are certified and regulated by another country's or region's regulatory agency (e.g. EASA) with whom the FAA has a bilateral agreement. The FAA can inspect these facilities, but only after coordinating with the corresponding regulatory agency and the foreign government, this is the reason why there is no direct control arrow between FAA and MRO in the control structure. Technically the agreements are between governments and not between the FAA and another regulatory agency, but for the purposes of this figure it was simpler to draw a direct link without compromising the analysis.

3.4 Unsafe Control Actions

This section lists the unsafe control actions (UCAs) for each controller, and provides additional explanations where necessary.

3.4.1 MRO mechanics / Airline mechanics UCAs

Control Action	Providing causes hazard	Not providing causes hazard	Too soon/too late causes hazard
Initial FLS install	(1.1) Install FLS not as specified by type certificate → H3, H4	(1.2) Software install not performed when component is new → H4	
Update FLS	(2.1) Update FLS not as specified by AD or service bulletin → H3, H4	(2.2) Update FLS not provided in accordance with AD or service bulletin → H3	
Update terrain DB	(3.1) Load terrain DB other than official one → H2	(3.2) Not loading terrain DB with current obstacles → H2	

Table 1: Unsafe control actions for MRO Mechanics / Airline Mechanics

The mechanic's responsibility is to keep the aircraft in an airworthy condition. With respect to field loadable software this requires making sure that the software is installed according to the type certificate and that it is up to date with service bulletins.

In addition, the mechanics are also responsible for updating the terrain database of the enhanced ground proximity warning system, and other databases such as navigation databases. The various databases are usually each updated at regular, but different intervals.

Matters are complicated slightly by the fact that some maintenance work is performed by airline mechanics while other maintenance work is outsourced to MROs. Especially for large overhauls, airlines often use MROs in countries where the cost of labor is significantly lower than in the US in order to save money. Dividing maintenance work between different organizations requires that coordination or control is particularly well organized. If this is not the case, misunderstandings can lead to hazardous situations.

3.4.2 Airline management UCAs

Control Action	Providing causes hazard	Not providing causes hazard	Too soon/too late causes hazard
Instruct MRO to perform maintenance		(4.1) Instruction to update FLS not provided when AD or service bulletin requires it, and airline mechanics are not tasked with performing update. → H3Instruction to update FLS not provided when AD or service bulletin requires it, and MRO mechanics are not tasked with performing update. → H3	
Instruct airline mechanics to perform FLS update		(5.1) Instruction to update FLS not provided when AD or service bulletin requires it, and MRO mechanics are not tasked with performing update. → H3	

Table 2: Unsafe control actions for Airline management

The unsafe control actions of the *airline management* controller are related to its role as coordinator between the airline mechanics and the MRO. Specifically, the airline management has to ensure that its aircraft are maintained in an airworthy state by assigning maintenance tasks to either the airline's mechanics or the MRO, but not to both. Furthermore, it has to ensure that the tasks assigned to each are carried out as required.

3.4.3 Pilot UCAs

Control Action	Providing causes hazard	Not providing causes hazard	Too soon/too late causes hazard
Perform pre-filght check of EGPWS		(6.1) Pre-flight check of EGPWS not performed when EGPWS does not conform to type certificate → H3, H4	
Pull up + TOGA	(7.1) Pull up + TOGA performed when low on fuel or there is conflicting traffic → H1	(7.2) Pull up + TOGA not performed when EGPWS correctly issued terrain alert → H2	(7.3) Pull up + TOGA performed too late when EGPS correctly issued terrain alert → H2

Table 3: Unsafe control actions for pilot

Pilots have many responsibilities, but only a few of those are relevant to our analysis: verifying that the EGPWS is in working order before taking off and avoiding collisions with terrain, obstacles and other aircraft. The potentially unsafe control actions are: not performing the pre-flight check of EGWPS and not properly reacting to a terrain alert. There are many more control actions involved in manipulating the aircraft, but the two control actions above capture the essence of what the pilots do with respect to the EGPWS, which we are focusing our analysis on. Adding more details would only make the analysis more complicated without generating further insights.

3.4.5 EGPWS UCAs

Control Action	**Providing causes hazard**	**Not providing causes hazard**	**Too soon/too late causes hazard**
Issue terrain alert		(8.1) Does not provide terrain alert when collision with terrain or ground structure is imminent→ H2	(8.2) Warn too late of imminent collision with terrain or ground structure → H2

Table 4: Unsafe control actions for EGPWS

For the purposes of this analysis, we only consider one control action for the EGPWS: issuing a terrain alert. Strictly speaking, issuing a terrain alert is not even a control action per se because the pilot makes the ultimate decision and thus remains in control. The EGPWS only notifies the pilot of terrain proximity. However, since rules require pilots to react to EGPWS terrain alerts under almost all circumstances, it is acceptable to characterize it as a control action for the purposes of this analysis.

3.4.6 Supplier UCAs

Control Action	Providing causes hazard	Not providing causes hazard	Too soon/too late causes hazard
Deliver software	(9.1) Software is delivered that is unsafe or insecure → H3, H4		
Deliver update	(10.1) Update is delivered that is unsafe or insecure → H3	(10.2) Update is not delivered when current version of FLS is unsafe or insecure → H3	(10.3) Update is delivered too late when current version of FLS is unsafe or insecure → H3
Issue service bulletin		(11.1) Service bulletin is not issued when current FLS is unsafe or insecure → H3, H4	(11.2) Service bulletin is issued too late when current FLS is unsafe or insecure → H3, H4

Table 5: Unsafe control actions for supplier

The supplier is responsible for developing safe and secure Field Loadable Software and distributing it to airlines. It should make sure that any flaws in the software are fixed quickly and airlines are notified by means of service bulletins. Potentially unsafe control actions of the supplier include: delivering unsafe/insecure software or software updates, delivering updates too late or not at all and issuing service bulletins too later or not at all.

3.4.7 FAA UCAs

Control Action	Providing causes hazard	Not providing causes hazard	Too soon/too late causes hazard
Provide initial type certificate	(12.1) Provide type certificate when FLS software is unsafe or insecure → H3, H4		
Provide type certificate for update	(13.1) Provide type certificate when FLS update is unsafe or insecure → H3, H4	(13.2) Type certificate for update is not provided when current FLS is hazardous and unsafe or insecure → H3, H4	(13.3) Certificate for update is provided too late when current FLS is unsafe or insecure → H3
Issue airworthiness directive		(14.1) Airworthiness directive is not issued when current FLS is unsafe or insecure → H3	(14.2) Airworthiness directive is issued too late when current FLS is unsafe or insecure → H3
Certify airline	(15.1) Certify airline which does not handle FLS install/update properly→ H3, H4		

Table 6: Unsafe control actions for FAA

The FAA is responsible for overseeing airline, supplier and MRO operations, if the organization falls under its jurisdiction. It should coordinate with foreign regulatory agencies for oversight of MROs that don't fall under it's jurisdiction (this is not a control action, however).

The FAA must ensure that it does not issue type certificates for FLS configurations that are unsafe or insecure. If an update to FLS is necessary, the FAA must provide any necessary type certificates quickly and in some cases issues an airworthiness directive. Furthermore, the FAA is responsible for licensing airlines. Airlines that do not follow safe maintenance practices with respect to FLS should not be licensed.

The FAA obviously has many more responsibilities, but for the purpose of this analysis, we consider just the above.

3.4.8 Foreign Regulatory Agency UCAs

Control Action	Providing causes hazard	Not providing causes hazard	Too soon/too late causes hazard
Certify MRO	(16.1) Certify MRO which does not handle FLS install/update properly $\rightarrow$ H3, H4		

Table 7: Unsafe control actions for Foreign regulatory agency

The foreign regulatory agency technically has responsibilities similar to that of the FAA. In an ideal world, the FAA and the foreign regulatory agency would complement each other perfectly and each would fulfill their role of overseeing the aviation industry in their respective countries. However, we live in an imperfect world, and not every foreign regulatory agency is as well equipped to fulfill that task as the FAA is. It therefore makes more sense to consider the FAA and the foreign regulatory agency to be asymmetrical in their responsibilities: From the point of view of the US aviation industry (with respect to FLS), the foreign regulatory agency's only responsibility is to ensure that only capable and competent MROs are certified to perform maintenance on aircraft.

A careful reader will have noticed that the MRO management does not have any UCAs. This is because of the way the roles are assigned: The MRO is not in fact responsible for making sure that maintenance was performed properly, this is still the airline's job. Someone in the airline management has to make sure that the commissioned maintenance work was actually done (i.e. written into the maintenance log). As explained later in this analysis, there are scenarios in which this division of tasks can lead to hazardous situations.

3.5 Scenarios for Unsafe Control Actions

This section presents a selection of scenarios in which the analysis suggests that an attacker could cause one of the system losses through the manipulation of a field loadable software component. The attacker could be anyone with the necessary expertise: a dishonest employee, a spy, a terrorist, a criminal, a security researcher, etc.

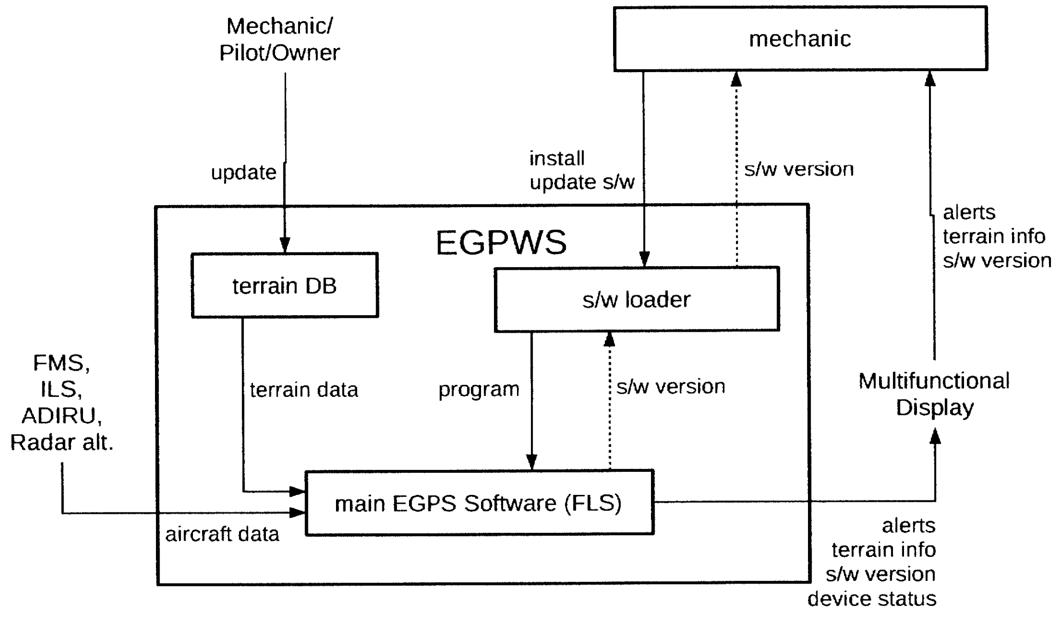

Figure 3: A more detailed view of the EGPWS and its interactions with the mechanic, illustrating that there is currently no way of determining the exact FLS version installed on the device without relying on the installed software itself – a clear security vulnerability.

Scenario 1:

An attacker could send modified field loadable software to the airline by posing as the supplier, or send modified field loadable software to the MRO by posing as the airline. While all MROs and airlines have procedures in place for handling FLS, these measures depend on the discipline of every single person involved. The prevalence of phishing and other forms of social engineering shows that such defenses are generally not effective enough. For parts that do not require digital signatures, there are currently no technical measures in place to prevent a mechanic from installing such software on an aircraft. In fact, once the software is installed on the aircraft, the airline currently has no simple way of finding out whether the software has been tampered with (see Figure 2).

Scenario 2:

An attacker could gain physical access to an aircraft and install modified software on a FLS component. Once the attack is complete, there is no way to positively determine that it happened. An airline mechanic can query the device for the installed software version, but the device in turn obtains that information by asking the installed software, which may claim to be the most recent version published by the manufacturer, even though its code has been tampered with. If harmful software is installed on the aircraft, it cannot be trusted with identifying itself correctly.

There are currently no controls in place to make sure that software that has been tampered with cannot be installed on a field loadable component. The process guards against accidental modification by using checksums, but an attacker could simply generate a valid checksum (even the exact same checksum) after modifying the software.

Scenario 3:

An MRO could be forced by the government of the country it is operating in to install a modified FLS version on the aircraft. Even when digital signatures are used, it is likely that the government could either produce a valid signature or find ways to defeat the signature verification on the component. This could be done without physically modifying the FLS part. It is currently not possible to determine exactly what software is installed on most components, thus the sabotage would be unlikely to be discovered.

Scenario 4:

An attacker could send a fake terrain database update to the airline or MRO and cause them to install it on the component. The database could have added obstacles that cause spurious terrain alerts and possibly dangerous evasive maneuvers, or it could have an altered terrain profile, which may cause the pilot to fly the aircraft into terrain in bad weather without ever getting a terrain alert. Currently, there are no technical measures in place to ensure that only official terrain databases can be loaded. Neither is there a way to tell a legitimate version of the terrain database from an illegitimate one after it has been installed.

These scenarios show that in terms of security current field loadable software does not meet the requirements of Order 8110.49. Specifically, the following points are not currently adequately addressed:

In chapter 5 -2 of Order 8110.49:

- Confirm that there is a process in place to ensure that the software loaded is the software approved and that the software has not been corrupted.

Issue: Some current FLS use only checksums to verify the data load during the loading process. Even for FLS that contain digital signatures, software can only be identified by its version number once installed, which means that if an adversary succeeds in installing modified software, it cannot be easily discovered.

- Confirm that the applicant has a configuration management process in place to assure that the installation configuration (that is, the software part number, the hardware part number, the aircraft or engine model, and the aircraft or engine serial number combinations, as applicable) is the same configuration that was approved during the authorization process.

Issue: Currently, field loadable software is generally identified by a version number that could easily be falsified by an adversary.

In chapter 5 -4 of Order 8110.49:

- The applicants Aircraft Maintenance Manual or Instructions for Continued Airworthiness should include a procedure that requires maintenance personnel to verify the software part number configuration before and after maintenance is performed on the airborne equipment.

Issue: Current FLS components do not provide a way to determine the exact version of installed software without relying on the installed software itself to provide the correct answer. An adversary could take advantage of this and modify the software while still making it identify itself as an official version.

Note: The scenarios presented in this section are just a small fraction of the hazardous scenarios that were found in the analysis. For a full list of scenarios, please refer to the appendix.

Chapter 4 Results and Recommendations

The STPA security analysis of Field Loadable Software showed that there are currently two major issues (for more issues, refer to the appendix).

1. There is no secure way of identifying software once it is installed on the aircraft.
2. Not all FLS components prevent installation of software from non-official sources.

Based on the analysis, the following security controls could be used to ensure the safety of the aircraft in the presence of adversaries who attempt to interfere with the proper operation of the software:

1. A secure hash function could be used to uniquely identify software. This hash should be explicitly mentioned in the type certificate and the manufacturer's manuals and service bulletins. Mechanics should be required to verify this hash before and after installing software.

Field loadable hardware should provide a secure way of computing this hash for the currently installed software to make it possible to easily and quickly determine its exact version. This would make it significantly more difficult for an attacker to surreptitiously install modified software without physically tampering with the FLS component. This is especially important for components that are loaded remotely via LoadStar or similar technologies.

The advantage of simple cryptographic hashes over digital signatures is that hashes do not depend on the secrecy of any keys, and they can be easily computed anywhere. That means that the software and the valid hash for it can be independently distributed, therefore eliminating single points of attack. Furthermore, there are ways to present hashes such that they can easily be verified by humans if necessary, thus allowing humans to be in the loop for updates.

2. To ensure that any modification of software installed on an aircraft is discovered, a system could be used that securely combines the trusted hashes of all software components installed on the aircraft. This list of hashes could in turn be hashed to provide a unique fingerprint for all software installed on the aircraft. Since every combination of hardware and software configurations must be covered by a type certificate, the airline could use a list of type certified fingerprints provided by the FAA to quickly check an aircraft's airworthiness.

3. Secure digital signatures could be used to certify the origin of software and data loads. Field loadable components could be pre-loaded with a root certificate such that software without a valid signature could not be installed without physical tampering. Such schemes are especially important for software that is loaded remotely via LoadStar where installation could be automated.

Boeing and Airbus have both developed digital signature schemes based on a public key infrastructure, but their schemes differ in significant ways: In the Airbus scheme, the software contains only one signature, the signature by Airbus. In Boeing's scheme, there are three (or possibly more signatures): The software supplier's signature, Boeing's signature and the airline's signature. On the aircraft component, only the airline's signature is verified. Both of these schemes have some advantages and some disadvantages:

Advantages of Boeing's scheme:

- The airline has the final say on what software can be installed on the aircraft.

- Not all Boeing aircraft in the world automatically accept software signed by a single key

Disadvantages of Boeing's scheme:

- Because there are three points at which signatures are generated and verified, an attacker has more weak spots to attack. At most one of three keys has to be compromised to insert malicious software into the chain.

- If airlines do not verify Boeing's signature at the same time as signing software with their own, an attacker could modify the software between the time Boeing's signature was verified and the Airline's signature is applied.

Advantages of Airbus' scheme:

- As long as Airbus' private keys are secure, software cannot be modified after it leaves Airbus. This means that security is ensured regardless of how the airline handles the software.

Disadvantages of Airbus' scheme:

- All eggs in one basket: Every Airbus aircraft will accept software signed by a single private key. If an attacker can get their hands on the secret key, they can potentially attack all airbus aircraft around the world (it would be possible to use separate keys to mitigate this though).

Both schemes share the disadvantage that their security depends on the secure distribution of public keys. While the public keys are not secret, it must be ensured that everyone holds the correct public keys. If a public key infrastructure is used for that, all security will depend on the secrecy of the root CA's private key.

Should a private key ever be compromised, all components would need to have the corresponding certificates revoked and a new public key loaded. This must by design be expensive and difficult (e.g. physically removing the component and opening it) because if such updates were easy, attackers could exploit this fact to install their own public keys. Should a public key ever need to be revoked, the financial consequences for airlines or aircraft manufacturers could be catastrophic.

Overall the digital signature schemes used by Airbus and Boeing are a reasonable way to increase security from attackers that do not possess sophisticated skills. They do however come at the costs of significantly increased risk from sophisticated attackers such as nation states as soon as software distribution is completely automated.

In addition to the measures above, which are necessary for compliance with Order 8110.49, regulation and guidance should be updated to require a security assessment such as described in DO-326A for any future certification. The security assessment should clearly document the assumptions under which the system is considered secure. STPA could be used to derive a systems model and find the security constraints that need to be applied to avoid losses. The FAA should provide guidance on what kind of security threats should be considered. Because the security landscape is constantly changing, such guidance should be updated whenever new and relevant information becomes available

Chapter 5 Future Work

STPA is very useful and powerful because it provides a means for abstracting systems problems in a way that seems to be easy to grasp for people, i.e. it presents the relevant information about a system in a very intuitive way.

However, because it is highly structured, it cannot easily be applied to all systems: The assumption underlying STPA is that a system can be modeled as a set of hierarchically organized controllers, each of which has a limited set of control actions. The space of system states can be discretized and hazardous states occur when one or more control actions are not applied properly in a given state.

A model of a system is necessarily a simplification of reality and does not perfectly correspond to it. Some systems are a more natural fit for STPA than others. Industrial control systems and cyber-physical control systems, for which STPA was initially invented, generally fit the the model very well, but even then it may not be straightforward to find the right mapping from reality to the abstract system. In reality even relatively simple systems can have hundreds of controllers and thousands of control actions, and finding a good abstraction be a challenge. This is especially true of computer systems that are not organized in a clear hierarchical fashion.

Physical systems – systems that physically manipulate a part of the world, like many industrial control systems do – are relatively tightly constrained by the laws of physics. These physical constraints require the system to control parts of the environment it is in and keep variables within safe bounds. In the case of industrial control systems, this has led to a relatively standard hierarchical organization which tends to fit the STPA model well.

In computer systems however, there are few physical constraints. The physical constraints apply to the lower levels of abstraction (i.e. the hardware), but the software itself is several steps removed from the hardware and has many more degrees of freedom, all of which add complexity. While there are many patterns and paradigms in software engineering, there is so far no one established way to structure software. Instead, engineers can choose from many languages, architectures and patters, and they may even freely combine them or make modifications to them. This freedom enables engineers to implement very complex and sometimes hard-to-understand solutions. If given a very powerful tool that makes certain solutions easier than others, software engineers will be inclined to try to fit the problem to the tool instead of fitting the tool to the problem, leading to solutions that are more difficult to reason about than necessary.

An additional problem posed by computer systems is that they are relatively opaque: mechanical systems, such as a clockwork or a bicycle can be observed in action and it is often very clear when they exhibit behavior that is not according to specification. Computer systems however are harder to inspect. In order to understand their inner working, they need to be instrumented. In ideal cases, a debugger can be used, but in many systems under real-time constraints, even that is not possible and the only thing engineers can do is to run simulations or to study traces of the program's execution after the fact.

Due to all these factors, many computer systems do not fit STPA very well. Instead of having clearly separated controllers all of them may be running on the same machine and in the same process, instead of using hierarchical control the different parts may simply communicate with each other through messages, instead of organizing the code by the function it is supposed to achieve it may be organized by which physical components it affects, etc.

It is not hard to imagine that for some systems this freedom of structure does more harm than good, because it does not provide engineers with clear rules and guidelines to follow. Indeed, many common programming frameworks currently in use actually force programmers to do things in one specific way, thus taking away some power but adding value through simplicity.

For all the reasons stated above, I think it would be interesting to undertake research to see whether designing software systems according to the principles of STPA could help make software more safe and secure. I believe that for applications in many (if not all) cyber-physical systems it would be very beneficial, because hierarchy and control are natural ways for people to think about systems. If the software controlling a cyber-physical system is structured as a hierarchy of controllers, not only could STPA be used to analyze that structure, but it would provide a direct and simple way for engineers to visually inspect their program at a level of abstraction above the source code.

I imagine that an STPA-based programming framework would (among other things) be based on the following:

1. Controllers would be represented in the form of actors at a lower level. Each controller would run in its own process or thread. If necessary, processes could have different privileges and even run on separate physical or virtual machines. Where a certain process runs would be a simple matter of configuration and no code changes would be necessary.

Each controller defines interfaces for receiving feedback messages and interfaces for receiving control messages. Similar to standard object-oriented programming, controllers have their internal state and internal methods which can be accessed by the code defined in the interfaces. Each interface accepts only one specific message type. Message types have to inherit from control message or feedback message and would be defined separately. They could possibly be implemented with protocol buffers or a similar language that enforces type safety.

2. A tool with a graphical user interface that allows software engineers to define the architecture of the system at the level of controllers. In that user interface, connections between controllers would be explicitly defined, and the flow of the program would have to follow these connections. That means for example that controller A could not directly send

control message M to controller B unless they were explicitly connected. Controller B could not send feedback message F to controller A unless explicitly specified. The graphical user interface would have the benefit that the higher-level structure of the program could always be immediately inspected in a graphical form, making it very easy for anyone with the requisite knowledge to see and understand the higher-level structure of the program without needing to inspect many lines of source code or trust the documentation. In a way, the program structure would become self-documenting.

Rather than simply providing another separate perspective on the code, the graphical user interface should provide a way to explore the code. Thus it should be possible for instance to inspect the source code of an interface or a controller simply by clicking on it, not unlike "zooming in". When debugging, it should be possible to step through the execution of the code at the architecture level or zoom into one controller if desired. If an event in the controlled process triggers the sending of feedback or control messages, it should be possible to visually follow them as they "bubble up" to higher level controllers or "trickle down" to lower level controllers, where they are converted into outputs that act on the physical process being controlled.

3. A runtime monitor, not unlike the reference monitor in the Android OS, would ensure that the rules and connections defined in the graphical user interface are enforced. Additional rules, such as the maximum frequency of feedback from controller B to controller A could be defined and enforced as well with limited additional overhead.

I believe that such a framework could dramatically increase an engineer's intuitive understanding of the system and thus speed up development times, reduce the number of requirement errors and bugs as well as making debugging much easier. The visual representation of the program structure and the ability to zoom between different levels may enable engineers who are not familiar with a system to find their way around the code much faster.

In a second step, the framework could integrate profiling information, testing results, error traces and any textual documentation by directly visualizing them in the graphical user interface. It should not be necessary for the engineers to use many different programs and views just to understand what their program does. Source code, architecture, configuration, unit tests, profiling etc. should all be accessible through one integrated interface because they really belong together. Good software engineers don't think of their program's source code, architecture, configuration, execution profiles, error traces, documentation etc. as separate from one another. Instead, they integrate all of these information sources to a complete mental representation of the program in their mind. The tool(s) they use for working on their programs should present information in a way that facilitates the construction of these mental representation and not in a way that hinders them.

The framework outlined above is tailored to cyber-physical systems, but with minor modifications, it should be usable in other kinds of software as well, such as web applications or large distributed systems.

It seems clear that building such a system would be a major undertaking, but I believe that it is worth exploring. Software engineers have been historically constrained by a representation of their programs – text files in folders – and it is time that they are freed from those constraints. When the first programs were written, getting computers to understand (i.e. interpret) and execute the programs was the constraining factor, and so it was logical to choose a format that computers already understood: text files organized in folders. In today's large and complex software engineering projects, the challenge is no longer to get computers to run the programs, but for humans to understand them. It is only logical then that we should choose to represent programs in a form that is closer to the mental representations that we have of them.

Chapter 6 Conclusions

This project set out to assess the safety and security of field loadable software processes in commercial aviation using STPA and to make recommendations for improvements. To the author's knowledge this is the first comprehensive analysis of field loadable software practices and regulations, and it is the first complete published example of STPA applied to security.

Overall, this thesis successfully demonstrated that STPA can be applied to security in the context of aviation software. It studied field loadable software and the hardware it runs on, as well as the processes and procedures surrounding it. A high-level description of the FLS life-cycle and an insightful hierarchical control structure were produced and used to determine which control actions are potentially unsafe. Based on the list of unsafe control actions scenarios were found in which security vulnerabilities in field loadable software could lead to hazardous situations in which human lives could be put in danger. It demonstrated among other things that there is currently no way to conclusively determine whether the correct software was installed on a component or not, a weakness that could be exploited by a technically competent adversary. The industry's response when being confronted with this weakness was that their *security by obscurity* made attacks unlikely. That approach has not been very effective in any industry.

The research showed that simple changes, such as the addition of a hardware-supported secure hash for software components and minor modifications to the loading process could be used to control the biggest hazards and eliminate most loss scenarios found in this thesis.

Even though no information could be obtained directly from airlines and only one component manufacturer informally provided information for this research, it is evident that current field loadable software processes are not sufficiently secured against attacks by well-funded adversaries. Because the security landscape is constantly changing, attacks that only a well-funded adversary can carry out today could become possible for individual hackers before long.

This research furthermore showed that the field loadable software processes do not always conform to current FAA regulation. Even where processes follow FAA regulation, safety and security are not guaranteed because current FAA regulation does not take security into account sufficiently. For instance, the STPA analysis showed that the important feedback loop from pilots and airlines to software suppliers is very important, but currently it is not covered by any effective regulation. Guidelines for software distribution in aviation are available, but no secure standard has been adopted by the FAA. For software development, there are currently no official standards at all in aviation. Clearly, there remains quite some work to be done to make field loadable software secure.

The STPA analysis presented in this thesis was relatively high level; it concerned itself with the recommended practices and regulations and did not study any particular supplier or airline's practices in detail. Such work is necessary in the future to determine how serious the weaknesses really are that this analysis found.

The analysis presented in this thesis also did not study the software implementation itself. It would be an interesting topic for further research to see if and how STPA can be applied to software directly. As I argued in this thesis, it may not be straightforward to apply STPA to software in general because not all software has a hierarchical control structure, but it may be a very interesting and rewarding research project to see whether software used in cyber-physical systems can be designed according to the principles of STAMP, i.e. with a hierarchical control structure and clearly defined controllers, control messages and feedback.

Bibliography

[1] Boeing Commercial Airplanes, Statistical Summary of Jet Airplane Accidents, 1959 – 2014 (2015)
[2] Beck, Walter R., et al. "Method and Apparatus for Loadable Aircraft Software Parts Distribution." U.S. Patent Application 12/277,174.
[3] Johnson, Leslie A. "DO-178B, Software considerations in airborne systems and equipment certification." *Crosstalk, October* (1998).
[4] Hoare, Charles AR. "How did software get so reliable without proof?." *FME'96: Industrial Benefit and Advances in Formal Methods.* Springer Berlin Heidelberg, 1996.
[5] Leveson, Nancy. *Engineering a safer world: Systems thinking applied to safety.* MIT Press, 2011.
[6] Silva, Vijay D., Daniel Kroening, and Georg Weissenbacher. "A survey of automated techniques for formal software verification." *Computer-Aided Design of Integrated Circuits and Systems, IEEE Transactions on* 27.7 (2008): 1165-1178.
[7] Woodcock, Jim, et al. "Formal methods: Practice and experience."*ACM Computing Surveys (CSUR)* 41.4 (2009): 19.
[8] Parnas, David L., A. John van Schouwen, and Shu Po Kwan. "Evaluation of safety-critical software." *Communications of the ACM* 33.6 (1990): 636-648.
[9] Bertolino, Antonia. "Software testing research: Achievements, challenges, dreams." *2007 Future of Software Engineering.* IEEE Computer Society, 2007.
[10] Robinson, Richard, et al. "Electronic distribution of airplane software and the impact of information security on airplane safety."*Computer Safety, Reliability, and Security.* Springer Berlin Heidelberg, 2007. 28-39.
[11] Kornecki, Andrew J., and Mingye Liu. "Fault tree analysis for safety/security verification in aviation software." *Electronics* 2.1 (2013): 41-56.
[12] Byres, Eric, and Justin Lowe. "The myths and facts behind cyber security risks for industrial control systems."*Proceedings of the VDE Kongress.* Vol. 116. 2004.
[13] Ravi, Srivaths, et al. "Security in embedded systems: Design challenges." *ACM Transactions on Embedded Computing Systems (TECS)* 3.3 (2004): 461-491.
[14] Young, William, and Nancy G. Leveson. "An integrated approach to safety and security based on systems theory." *Communications of the ACM* 57.2 (2014): 31-35.
[15] Anderson, Ross. "Why information security is hard-an economic perspective." *Computer Security Applications Conference, 2001. ACSAC 2001. Proceedings 17th Annual.* IEEE, 2001.
[16] Checkland, Peter. "Systems thinking, systems practice." (1981).
[17] Antoine, Blandine. *Systems Theoretic Hazard Analysis (STPA) applied to the risk review of complex systems: an example from the medical device industry.* Diss. Massachusetts Institute of Technology, 2013.
[18] RTCA / EUROCAE. “Software Considerations in Airborne Systems and Equipment Certification”, DO- 178C/ED-12C (2011)
[19] Federal Aviation Administration, “Airborne Software Assurance, Advisory Circular AC 20.115C (2013)
[20] FAA, “Software Approval Guidelines”, Order 8110.49 (2003)
[21] RTCA/ EUROCAE. “Airworthiness Security Process Specification”, DO-326A (2010)
[22] RTCA/ EUROCAE. “Information Security Guidance for Continuing Airworthiness”, DO-355 (2014)
[23] SAE, “Guidance For Security of Loadable Software Parts Using Digital Signatures”, ARINC Report 835-1 (2014)

[24] Leveson, Nancy. "A new accident model for engineering safer systems." *Safety science* 42.4 (2004): 237-270

Appendix

Complete List of Scenarios for Commercial Aviation

Note: We consider software to be unsafe or insecure if it can lead to one of the system hazards defined earlier. STPA can help to determine whether the requirements prevent known hazards from occurring, but verifying whether or not a particular implementation of the software corresponds to these requirements is not in the scope of this analysis.

Field loadable software can be unsafe in different ways:
- The software may cause the component on which it is installed to stop working properly
- The software may cause other components to which it is connected to stop working

The failure of a field loadable component may affect other components physically by
- causing an electrical fault (e.g short-circuit, over-current, etc.)
- overheating, catching fire or exploding.
- cause water, hydraulic fluid or fuel leaks
- etc.

The individual scenarios could naturally be refined to a higher level of detail if desired. We decided to stop at the point where enough detail is available to devise reasonable security controls.

MRO mechanic / airline mechanic UCAs

UCA 1.1, 2.1: Mechanic installs software not as specified by type certificate or does not perform update as specified by service bulletin.

Due to incorrect feedback or incorrect instructions from another controller:

- Mechanic installs software onto field loadable system not as specified by the type certificate because the mechanic was provided the wrong software to install. This may occur due to several reasons:
    - the attacker hacked the airline's computer and installed his/her own DNS and certificates
    - the attacker carried out a man-in-the middle attack on the connection to the supplier's server
    - the attacker obtained the supplier's web domain and redirected it to his server.
    - the attacker sent a phishing e-mail to the airline with instructions to update the FLS and a link pointing to his server

- Mechanic installs software onto field loadable system not as specified by the type certificate because the mechanic accidentally installed the wrong software for one of the following reasons:
    - the airline or MRO's management of software parts makes it easy to confuse different versions of the same software
    - the supplier's naming convention makes it easy to confuse software versions

- Mechanic installs software onto field loadable system not as specified by the type certificate because the service bulletin was modified by someone to contain instructions to install different software. This person could be anywhere at or between the supplier and the mechanic.

- MRO or airline Mechanic installs software onto field loadable system not as specified by the type certificate because he believes that the correct version is already installed on the aircraft for one of the following reasons:
    - The other mechanics usually perform these installs (either MRO or airline respectively).
    - The maintenance log was falsified by someone
    - There was a mistake in the maintenance log
    - The maintenance log is hard to read
    - The software installed on the component displayed the correct version, even though it was not the correct software

Due to process model flaw or algorithm flaw:
- Mechanic installs software onto field loadable system not as specified by the type certificate because he confused two software versions that he received earlier.

- Mechanic does not update software according to service bulletin because the mechanic forgot about the service bulletin (and the process depends on his memory)

- Mechanic installs software onto field loadable system not as specified by the type certificate because the software was modified on the MRO or airline mechanic's computer in one of the following ways:
    - An insider with access modified it
    - An attacker hacked the computer to modify the software

- Mechanic *intentionally* installs software onto field loadable system not as specified by the type certificate because he was pressured into doing so or because he holds a grudge against the airline or the MRO.

- The MRO mechanic is instructed to install a modified FLS by the government of the country in which the MRO is based.

Control action is issued but not effective:
- The mechanic does not properly carry out the install because he forgets one of multiple steps in the installation process, thus leaving the component in an inconsistent state.

- The mechanic does not properly complete the install because the component provides no clear feedback about whether the install succeeded / failed

- The mechanic installed the wrong software because the loading device contained multiple software versions and the mechanic selected the wrong one.

- The mechanic carried out the installation according to the supplier's instructions, but the installation produces an incorrect result because either the loading device was broken or tampered with, or the aircraft component was broken or tampered with.

UCA 2.1, 2.2: Initial software install not performed when component is new

Due to incorrect feedback or incorrect instructions from another controller:
- The mechanic assumed that no install was necessary because he did not get the correct instructions from the airline management.

Due to process model flaw or algorithm flaw:
- The mechanic forgot to install the software because some parts come with software pre-installed while others don't, and the mechanic confused the component in question with another one. This may occur because:
  - the mechanic often takes shortcuts to save time instead of following the manual step by step.
  - the mechanic does not have the necessary experience and is not properly supervised.
  - the mechanic was interrupted in the middle of the task and did not complete it

- The mechanic did not install the software because he thought another mechanic would do it. The supervisor assumed the work was done and noted it in the maintenance log.

Control action is issued but not effective:
- The mechanic tried to install the software, but did not complete it because he thought the install was successful even when it was not, either because the component did not give clear enough feedback or because the instructions in the installation manual were unclear.

- The mechanic could not carry out the installation because he did not have the right tools for the job (e.g. the proper portable loading device)

UCA-2.2, 2.3: Update FLS not provided in accordance with service bulletin or AD

Due to incorrect feedback or incorrect instructions from another controller:
- The mechanic did not update the FLS because the service bulletin was not forwarded by airline management.

- The mechanic received the service bulletin but did not get the software update or the correct instructions from the airline because the airline forgot to provide them.

- The mechanic does not install the update because he believes that the software was already updated. This could happen if it is not straightforward to determine what version of software is installed on the component when the maintenance log is not clear about it.

Due to process model flaw or algorithm flaw:
- The mechanic decides not to apply the update because he is in a rush and thinks the

update is not critical because it's just a service bulletin and not an airworthiness directive.

- The mechanic intends to do the update but installs the wrong software because it is difficult to tell different versions apart (they're all just files with names) and the MRO or airline is not organized enough to keep track of the different versions.

Control action is issued but not effective:
- N/A

UCA 3.1: MRO or airline mechanic loads terrain database other than supplier's official one

Due to incorrect or missing feedback or incorrect instructions from another controller:
- The mechanic installs an unsafe database from a non-official source because the airline obtained it from a non-official source. This could happen if the attacker knows airline internals and sends the modified terrain database directly to the right person.

- The mechanic installs an unsafe database because an attacker swapped it with the real database either at the supplier or somewhere between the supplier and the mechanic.

- The MRO mechanic installs an unsafe terrain database upon request of the government of the country that the MRO is based in.

Due to process model flaw or algorithm flaw:
- N/A

Control action is issued but not effective:
- The mechanic installs the official database, but the install does not succeed, leaving the database in the old state or in a non-functioning state.

Airline Management UCAs

UCA 4.1, 5.1: Airline management does not oder either airline mechanics or MRO mechanics to update software when a service bulletin or AD requires it

Due to incorrect or missing feedback or incorrect instructions from another controller:
- The airline misreads a line in the maintenance log and believes that the update was performed even though it wasn't.

Due to process model flaw or algorithm flaw:
- The airline does not order the mechanics or the MRO to update the FLS because only a service bulletin (and not an AD) was received and the airline management does not want to pay the cost of updating.

- The airline management believes that it ordered the MRO to perform the FLS update when it didn't. It believes this because the contract with the MRO usually covers this kind of maintenance but not in this case.

- The airline management does not order the airline's mechanics to perform the FLS update because it erroneously believes that the MRO mechanics already updated the FLS. It believes this because the service bulletin or AD was received at the time the aircraft was sent into maintenance, and other aircraft that were sent in later and came back from the MRO earlier had the update applied.

Control action is issued but not effective:
- The airline orders the MRO to perform the update, but the update is not carried out because the airline did not provide the software or the manual to perform the update.

- The airline management orders the airline mechanics to perform the update, but the update is not completed because there is not enough personnel available and software updates are considered lower priority.

Pilot UCAs

UCA 6.1: Pilot does not perform pre-flight check of EGPWS when installed software does is not conform with type certificate

Due to incorrect or missing feedback or incorrect instructions from another controller:
- The pilot does not perform the pre-flight check of EGPWS because the airline's procedures do not require it.

Due to process model flaw or algorithm flaw:
- The pilot thinks the pre-flight check for the EGPWS was already performed because he was distracted during the process and didn't perform each check step-by-step.

- The pilot does not perform the pre-flight check of EGPWS because he/she thinks that this only needs to be done for the first flight of the day.

Control action is issued but not effective:
- The pilot performs the pre-flight check of the EGPWS, but does not notice that malicious software is installed because the software announces the same version number as the supplier's official version and the pilot has no other way of checking.

UCA 7.1, 7.2Pilot does not initiate TO/GA or initiates it too late when aircraft is about to hit terrain

Due to incorrect feedback or incorrect instructions from another controller:
- The pilot does not pull up because the EGPWS system is switched off and no alert sounds
- The pilot does not pull up because a TCAS alert is issued at the same time.

Due to process model flaw or algorithm flaw:
- The pilot does not pull up because he believes that the alert is just a nuisance alert because he is used to landing at airports that are not in the terrain database.
- The pilot does not pull up because he thinks he knows where he is and doesn't think the terrain is close.

Control action is issued but not effective:
- The pilot initiates TO/GA, but it is not effective because another system prevents the control action (due to high angle of attack etc.)
- The pilot initiates TO/GA, but the aircraft is too slow and cannot produce enough lift. Rather than risking a stall, the pilot continues descent.
- The pilot pulls up too late because he is distracted (maybe due to a malfunction) when the terrain alert sounds.

EGPWS UCAs

UCA 7.2, 7.3: EGPWS does not provide timely terrain alert when aircraft is about to hit terrain or ground structure

Due to incorrect feedback or incorrect instructions from another controller:
- The EGPWS does not sound an alert because the radar altimeter data is incorrect or missing due to malfunction or manipulation.

- The EGPWS does not sound an alert because the airspeed data is incorrect or the position is incorrect which could occur due to GPS sensor failure or due to spoofing or jamming of the GPS signal.

- The EGPWS does not sound an alert because someone (maybe even the pilot or co-pilot) turned EGWPS off. This could happen if the aircraft frequently lands at an airport that is not in the database.

Due to process model flaw or algorithm flaw:
- The EGPWS does not sound an alert because the terrain database is incorrect (mountain/hill/building missing or at wrong location) or outdated (new building)

- The EGPWS does not sound an alert because an adversary intentionally loaded the wrong map

- The EGPWS does not sound an alert because an adversary modified the FLS software to suppress alerts in certain conditions.

Control action is issued but not effective:

- The EGPWS device issues an alert, but the pilot does not hear or see it because the alert doesn't sound due to the pilot's headset failing.

- The EGPWS device issues an alert, but the pilot does not hear or see it because there is too much noise.

- The EGPWS device issues an alert, but the pilot does not hear or see it because he is focused on dealing with other warnings.

- The EGPWS device issues an alert, but the pilot does not hear or see it because the cable to the speakers and the warning light is disconnected.

- The EGPWS device issues an alert and the pilot notices it but takes no action because he intends to crash the plane.

- The EGPWS device issues an alert but the pilot cannot take action because he is incapacitated.

Supplier UCAs

UCA 10.1: Supplier delivers unsafe or insecure software

Due to incorrect feedback or incorrect instructions from another controller:

- The supplier delivers unsafe or insecure software because the FAA certified it, which could happen if tests are not extensive enough or certification criteria are not stringent enough. It could also happen if the software tested is not exactly the same version as the one that was shipped.

Due to process model flaw or algorithm flaw:

- The supplier delivers unsafe or insecure software because the wrong version of the software was published by accident. This could occur if the publication process depends on only one employee or if there is no proper way of identifying software versions.

- The supplier delivers unsafe or insecure software because the wrong version of the software was published intentionally by an attacker/malicious insider who modified the software before publication.

- The supplier delivers unsafe or insecure software because severity of *known* hazard was downplayed and considered non-critical. This could happen because fixing issue and re-certifying software would be costly & time consuming.

- The supplier delivers unsafe or insecure software because an insider/attacker inserted malicious code into the code base.

- The supplier delivers unsafe or insecure software because some tests were skipped due to lack of time or money or because the testing setup did not work as intended.

Control action is issued but not effective:

- The supplier publishes safe and secure software, but the software is modified after being published by the supplier. This could happen on the server or on the way to the airline or MRO, or on the airline or MRO's computers themselves.

- The supplier publishes safe and secure software, but the airline or MRO mechanic installed the wrong version/software by accident

UCA 11.1: Supplier delivers update when update is unsafe or insecure

Software update is different from initial version insofar that the process leading to it is different, thus there can be additional scenarios. All the scenarios for initial software version should also be considered here.

Due to incorrect feedback or incorrect instructions from another controller:
- N/A

Due to process model flaw or algorithm flaw:
- The supplier ships an unsafe or insecure update because the bug fix/ update introduced new hazards for one of the following reasons:
    - the code change was not properly reviewed before publishing
    - the effect of change on interacting components was not considered
    - none or insufficient tests were added for the new code
    - the fix violated a system design assumption which was not properly documented

- The supplier ships an unsafe or insecure update because the update did not fix the underlying problem but only patched one instance of the bug.

Control action is issued but not effective:
- The supplier issued a safe and secure update, but an attacker tricked the airline into downloading a modified version the software from their own website. They could do this by sending a forged letter or e-mail to the airline which instructs them to download the software from the attacker's website.

UCA 11.2, 11.3: Supplier ships update too late or not at all when update is critical

Due to incorrect feedback or incorrect instructions from another controller:
- The supplier ships an update too late or not at all because the supplier is not aware of hazards in original software. This could happen if the software is not identified as causal factor in accidents or because the supplier does not get necessary feedback/reports from pilots and mechanics in the first place. The software may not be detected as the cause because the hazard arises only in rare conditions / for rare combinations of variables.

- The supplier decides not to ship an update for a known issue because they are not legally required to do so as long as the FAA does not issue airworthiness directive. The FAA may not issue an airworthiness directive if the hazard is not recognized as serious because the exact effects are hard to understand.

Due to process model flaw or algorithm flaw:
- The supplier ships an update too late because the update depends on an engineer that is unreachable (left company, sick, on vacation etc.). This could occur if the software is not documented sufficiently well for other engineers to understand it quickly enough.

- The supplier ships an update too late because of a lack of personnel

- The supplier does not ship an update because it is too costly and the company is in financial trouble.

Control action is issued but not effective:
- The supplier issues an update, but the update is not installed by the airline or the MRO because a corresponding service bulletin is not issued, or the service bulletin is issued but does not reach all the airlines.

- The supplier issues an update, but the airline or MRO mechanic mistakes unfixed version with fixed version and do not notice their mistake soon enough.

UCA 8.1, 8.2: Supplier issues service bulletin too late or not at all when FLS is unsafe or insecure

Due to incorrect feedback or incorrect instructions from another controller:
- N/A

Due to process model flaw or algorithm flaw:
- The supplier issues a service bulletin too late or not at all because the internal processes do not ensure that a service bulletin is issued whenever an update or new instructions are available.

Control action is issued but not effective:

- The service bulletin is issued, but it is not received by all airlines because letters are lost in the mail, or an e-mail server did not deliver the e-mail.

- The service bulletin is issued and received by the airline, but the airline does not act on it because the service bulletin does not set a deadline.

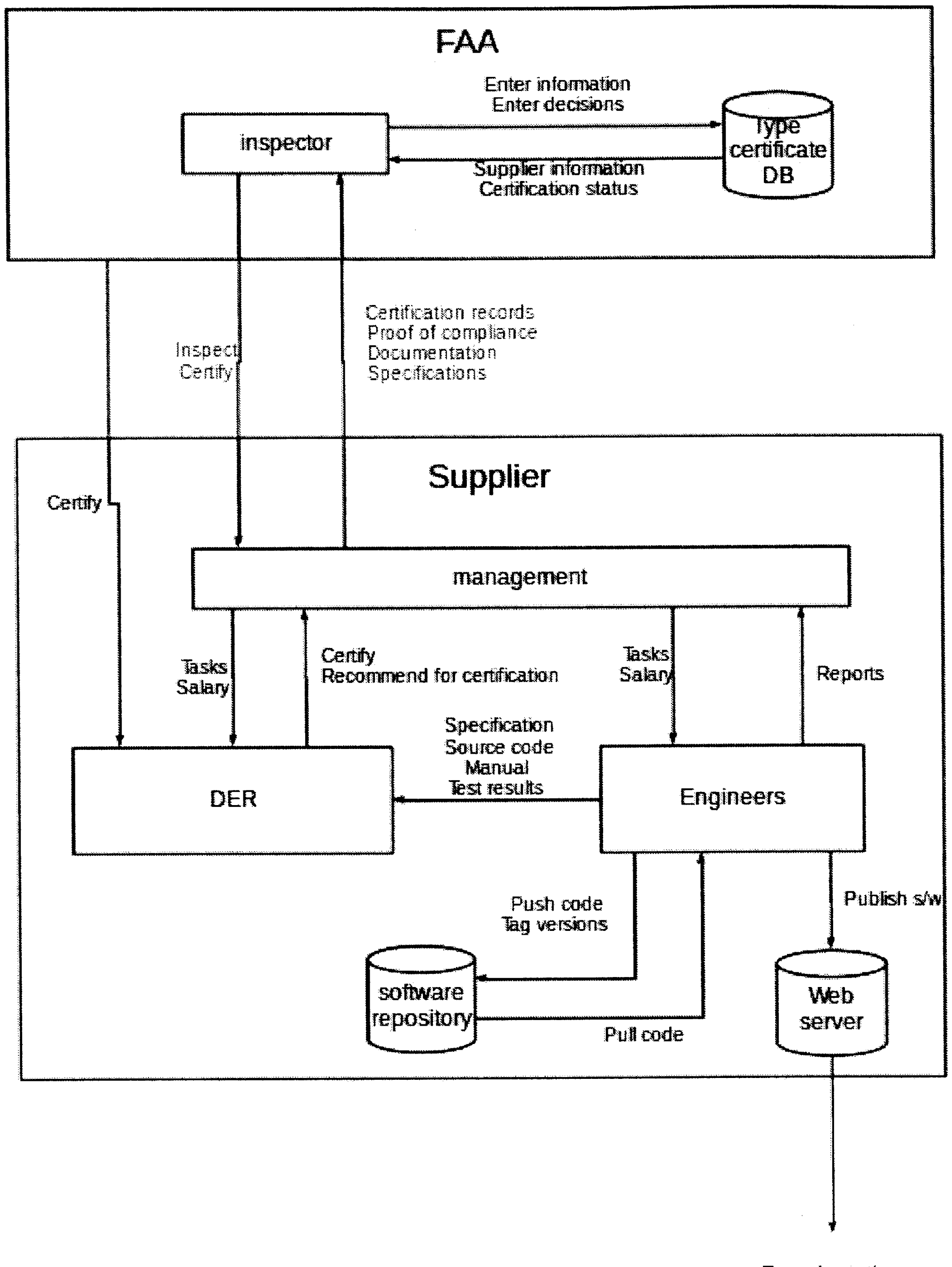

Figure 4: A more detailed view of the FAA and supplier controllers and their interactions

FAA UCAs

UCA 14.1, 15.1: FAA provides a type certificate for unsafe FLS or FLS software or update

Due to incorrect feedback or incorrect instructions from another controller:
- A type certificate is provided for unsafe software because the software tested on aircraft is not the exact software certified/delivered. This could happen if the software is not tracked appropriately with versioning.

Due to process model flaw or algorithm flaw:
- A type certificate is provided for unsafe software because someone hid a flaw on purpose (could be an employee of the supplier or an FAA official)

- A type certificate is provided for unsafe software because FAA employees are under pressure not to deny too many type certificates

- A type certificate is provided for unsafe software because the designated representative feels pressure to sign off on certificate because he/she is afraid of losing the job.

- A type certificate is provided for unsafe software because tests/checks are not thorough enough because the supplier has good track record with the FAA.

- A type certificate is provided for unsafe software because tests and inspections are insufficient to discover a hazardous flaw if the flaw only shows up in rare situations (like the one in BA flight 38)

- A type certificate is provided for unsafe software because the certificate lists a software version different from the one that was intended be certified due to a mix-up.

- A type certificate is provided for unsafe software because configurations that were not tested were certified based on their similarity with tested configurations. In this case, a subtle difference could be overlooked.

Control action is issued but not effective:
- NA

UCA 15.2, 15.3: FAA provides certification for a critical update too late or not at all

Due to incorrect feedback or incorrect instructions from another controller:
- FAA certifies a critical update too late or not at all because the request for certification gets lost in the mail (or someone's inbox).

- FAA certifies a critical update too late or not at all because the supplier did not submit all required documentation in the right format

Due to process model flaw or algorithm flaw:
- FAA certifies a critical update too late or not at all because the FAA does not have enough inspectors to process all the certification requests in time

- FAA certifies a critical update too late or not at all because the application gets stuck on someone's desk at the FAA and nobody else takes over the task.

- FAA certifies a critical update too late or not at all because the application is not treated with the necessary urgency. This could happen by accident if there is no formal way of prioritizing important applications

Control action is issued but not effective:
- The certification is issued but it gets lost in the mail (or someone's inbox) and nobody notices for several days or weeks.

UCA 16.1, 16.2: FAA issues airworthiness directive too late or not at all for hazardous FLS

Due to incorrect feedback or incorrect instructions from another controller:
- The airworthiness directive is issued too late or not at all because no incidents are reported to the FAA despite hazard. This could occur for one of the following reasons:
    - because hazard only occurs in rare circumstances
    - because no evidence of software issues can be found after the fact
    - because reporting is too onerous for mechanics or pilots
    - because the airline or MRO prefer to report directly to the supplier
    - because supplier doesn't report hazard to FAA
    - because mechanics or pilots do not consider the issues serious enough to warrant FAA report

- The airworthiness directive is issued too late or not at all because reported incidents never make it to the FAA due to a flaw in the system. Nobody notices the flaw, because there is no follow-up for reported incidents.

- The airworthiness directive is issued too late or not at all because reported incidents do not contain all necessary information to pinpoint the problem source.

Due to process model flaw or algorithm flaw:
- An airworthiness directive is issued too late or not at all because reported incidents are considered pilot errors
- An airworthiness directive is issued too late or not at all because reported incidents do not get dealt with in efficient manner, for example if the FAA cannot deal with the large number of reports due to bureaucratic overhead.
- An airworthiness directive is issued too late or not at all because discovered hazards are not considered serious enough to warrant an AD. This could occur if the supplier assures the FAA that they will deal with the problem themselves or if the FAA trusts the supplier's assurances that the hazard is not serious.

Control action is issued but not effective:
- FAA issues an airworthiness directive, but it is not effective because it was not noticed by all owners/pilots of the concerned aircraft.

Complete List of Scenarios for General Aviation

Repair Station UCAs

UCA 1.1: Repair station installs software not as specified by type certificate or does not perform update as required by service bulletin.

Due to incorrect feedback or incorrect instructions from another controller:
- Mechanic installs software onto field loadable system not as specified by the type certificate because the mechanic was tricked into downloading the software from an attacker's website. This may occur due to several reasons:
    - the attacker hacked the mechanic's computer and poisoned the DNS and certificates
    - the attacker carried out a man-in-the middle attack
    - the attacker obtained the supplier's web domain and redirected it to his server.
    - the attacker sent a phishing e-mail to the mechanic with a link pointing to his server

- Mechanic installs software onto field loadable system not as specified by the type certificate because the mechanic accidentally downloaded the wrong software for one of the following reasons:
    - the supplier's website UI made it easy to confuse software versions
    - the supplier's naming convention makes it easy to confuse software versions

- Mechanic installs software onto field loadable system not as specified by the type certificate because the service bulletin sent to the mechanic was modified on the way with instructions to install different software.

- Mechanic installs software onto field loadable system not as specified by the type certificate because he believes that the correct version is already installed on the aircraft for one of the following reasons:
    - The maintenance log was falsified
    - There was a mistake in the maintenance log
    - The mechanic recalled erroneously from memory and didn't check the maintenance log
    - The software installed on the component displayed the correct version, even though it was not the correct software

Due to process model flaw or algorithm flaw:
- Mechanic installs software onto field loadable system not as specified by the type certificate because he confused two software versions that he downloaded earlier.

- Mechanic does not update software according to service bulletin because the mechanic forgot about the service bulletin (and the process depends on memory)

- Mechanic installs software onto field loadable system not as specified by the type certificate because the software was modified on the repair station's computer in one of

the following ways:
- An insider with access modified it
- An attacker hacked the computer to modify the software

- Mechanic *intentionally* installs software onto field loadable system not as specified by the type certificate because he was pressured into doing so or because he holds a grudge against the pilot, aircraft owner or the repair station owner.

Control action is issued but not effective:
- The mechanic does not properly carry out the install because he forgets one of multiple steps in the installation process, thus leaving the component in an inconsistent state.

- The mechanic does not properly complete the install because the component provides no clear feedback about whether the install succeeded / failed

- The mechanic installed the wrong software because the loading device contained multiple software versions and the mechanic selected the wrong one.

- The mechanic installed the right software, but he installed it onto the wrong component because the components are in similar locations and share the same screen

- The mechanic carried out the installation according to the supplier's instructions, but the installation produces an incorrect result because either the loading device was broken or tampered with, or the aircraft component was broken or tampered with.

UCA 1.2: Initial software install not performed when component is new

Due to incorrect feedback or incorrect instructions from another controller:
- The mechanic assumed that no install was necessary because he did not get the correct instructions from the supplier and the component seemed to be working.

Due to process model flaw or algorithm flaw:
- The mechanic forgot to install the software because some parts come with software pre-installed while others don't, and the mechanic confused the component in question with another one. This may occur because:
  - the mechanic often takes shortcuts to save time instead of following the manual step by step.
  - the mechanic does not have the necessary experience
  - the mechanic was interrupted in the middle of the task and did not complete it

- The mechanic decided to skip installation/update because he was in a rush

- The mechanic did not install the software because he thought another mechanic would do it.

Control action is issued but not effective:

- The mechanic tried to install the software, but did not complete it because he thought the install was successful even when it was not, either because the component did not give clear enough feedback or because the instructions in the installation manual were unclear.

- The mechanic could not carry out the installation because he did not have the right tools for the job (e.g. the proper portable loading device)

UCA-2.1, 2.2: Update FLS not provided in accordance with service bulletin or AD

Due to incorrect feedback or incorrect instructions from another controller:
- The mechanic did not update the FLS because the service bulletin got lost in the mail and he never received it.

- The mechanic received the service bulletin but can not get the software update or the correct instructions because his computer is not working or the Internet connection is not working or the supplier's server is off-line.

- The mechanic does not install the update because he believes that the software was already updated. This could happen if it is not straightforward to determine what version of software is installed on the component when the maintenance log is not clear about it.

Due to process model flaw or algorithm flaw:
- The mechanic decides not apply the update because he is in a rush and thinks the update is not critical because it's just a service bulletin and not an airworthiness directive.

- The mechanic decides not apply the update because he is in a rush and assumes that the pilot/owner will not find out anyway since it's difficult to determine the installed version.

- The mechanic intends to do the update but installs the wrong software because it is difficult to tell different versions apart (they're all just files with names) and the repair station is not organized enough to keep track of the different versions.

Control action is issued but not effective:
- The mechanic cannot update the software because the pilot/owner never brought the aircraft in for maintenance or brought it in too late. This could happen if the pilot/owner didn't know about the service bulletin in the first place because they did not get the information from the supplier or the repair station could not reach them (maybe their contact details were no longer valid)

Pilot/owner UCAs

UCA 3.1: Pilot/owner loads terrain database other than supplier's official one

Due to incorrect or missing feedback or incorrect instructions from another controller:
- The pilot/owner downloads a modified database from a non-official source because it

looks like the real database and it is the first search result that comes up.

- The pilot/owner downloads a modified database from a non-official source because a phishing e-mail that looked like it came from the supplier had instructions to do so.

Due to process model flaw or algorithm flaw:
- N/A

Control action is issued but not effective:
- The pilot/owner installs the official database, but the install does not succeed, leaving the device in a non-functioning state.

UCA 3.2: Pilot/owner loads terrain database update too late or not at all

Due to incorrect or missing feedback or incorrect instructions from another controller:
- The pilot/owner does not load the update because he/she does not know about.

Due to process model flaw or algorithm flaw:
- The pilot/owner does not update the database because he/she thinks that the mechanic (or someone else) already did it.

- The pilot/owner does not update the database because he/she does not consider it important.

- The pilot/owner does not update the database because he/she forgets about it and is not reminded.

Control action is issued but not effective:
- The pilot/owner attempted to update the database but the update did not succeed. The pilot/owner did not check to make sure that the update succeeded.

UCA 4.2, 4.3: Pilot does not initiate TO/GA or initiates it too late when aircraft is about to hit terrain

Due to incorrect feedback or incorrect instructions from another controller:
- The pilot does not pull up because the EGPWS system is switched off and no alert sounds
- The pilot does not pull up because a TCAS alert is issued at the same time.

Due to process model flaw or algorithm flaw:
- The pilot does not pull up because he believes that the alert is just a nuisance alert because he is used to landing at airports that are not in the terrain database.
- The pilot does not pull up because he thinks he knows where he is and doesn't think the terrain is close.

Control action is issued but not effective:

- The pilot initiates TO/GA, but it is not effective because another system prevents the control action (due to high angle of attack etc.)
- The pilot initiates TO/GA, but the aircraft is too slow and cannot produce enough lift. Rather than risking a stall, the pilot continues descent.
- The pilot pulls up too late because he is distracted (maybe due to a malfunction) when the terrain alert sounds.

EGPWS UCAs

UCA 5.1, 5.2: EGPWS does not provide timely terrain alert when aircraft is about to hit terrain or ground structure

Due to incorrect feedback or incorrect instructions from another controller:
- The EGPWS does not sound an alert because the radar altimeter data is incorrect or missing due to malfunction or manipulation.

- The EGPWS does not sound an alert because the airspeed data is incorrect or the position is incorrect which could occur due to GPS sensor failure or due to spoofing or jamming of the GPS signal.

- The EGPWS does not sound an alert because someone (maybe even the pilot or co-pilot) turned EGWPS off. This could happen if the aircraft frequently lands at an airport that is not in the database.

Due to process model flaw or algorithm flaw:
- The EGPWS does not sound an alert because the terrain database is incorrect (mountain/hill/building missing or at wrong location) or outdated (new building)

- The EGPWS does not sound an alert because an adversary intentionally loaded the wrong map

- The EGPWS does not sound an alert because an adversary modified the FLS software to suppress alerts in certain conditions.

Control action is issued but not effective:

- The EGPWS device issues an alert, but the pilot does not hear or see it because the alert doesn't sound due to the pilot's headset failing.

- The EGPWS device issues an alert, but the pilot does not hear or see it because there is too much noise.

- The EGPWS device issues an alert, but the pilot does not hear or see it because he is focused on dealing with other warnings.

- The EGPWS device issues an alert, but the pilot does not hear or see it because the cable to the speakers and the warning light is disconnected.

- The EGPWS device issues an alert and the pilot notices it but takes no action because he intends to crash the plane.

Supplier UCAs

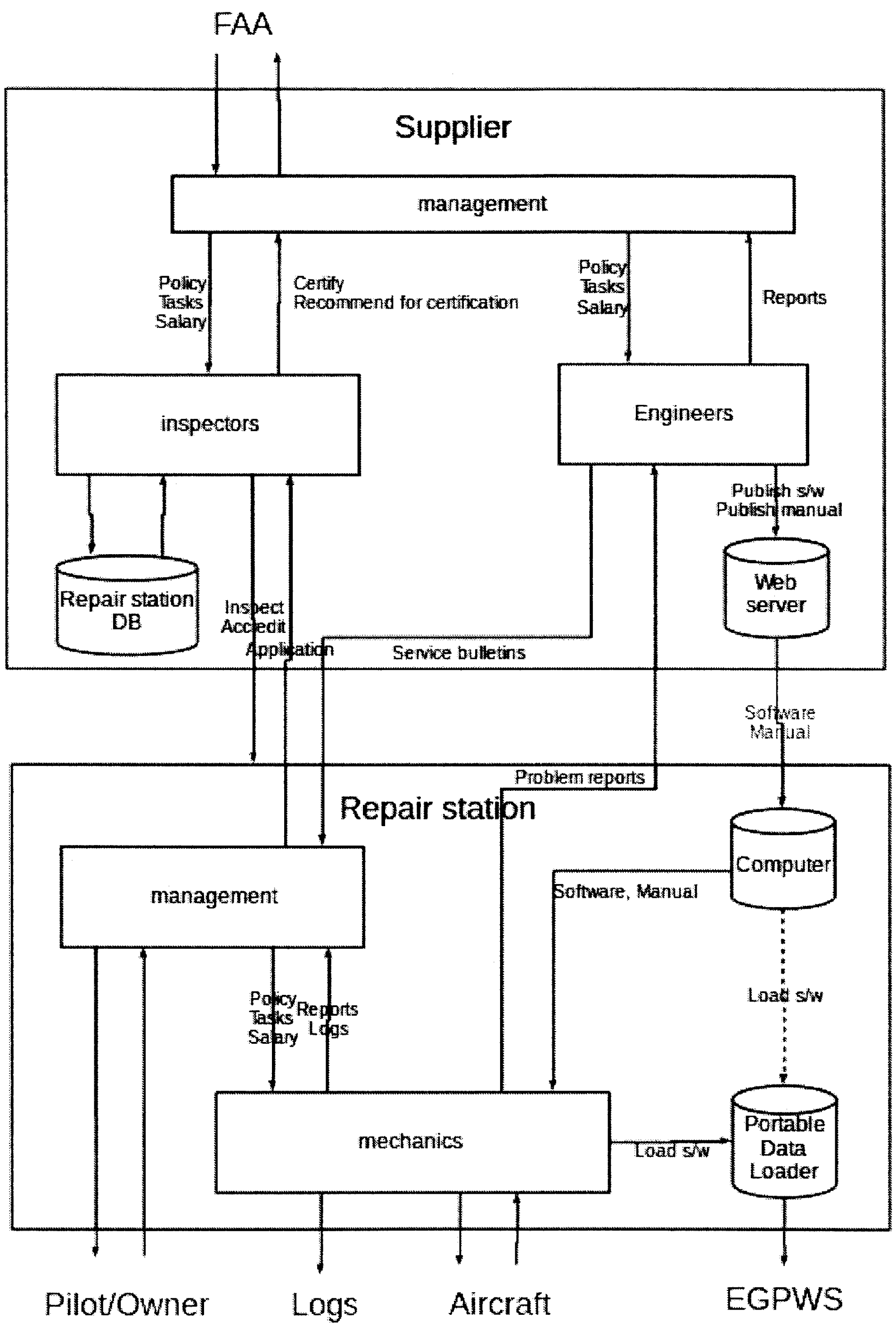

Figure 5: A more detailed view of the control loop between the supplier and the repair station

UCA 6.1: Supplier delivers unsafe or insecure software

Due to incorrect feedback or incorrect instructions from another controller:

- The supplier delivers unsafe or insecure software because the FAA certified it, which could happen if tests are not extensive enough or certification criteria are not stringent enough. It could also happen if the software tested is not exactly the same version as the one that was shipped.

Due to process model flaw or algorithm flaw:

- The supplier delivers unsafe or insecure software because the wrong version of the software was published by accident. This could occur if the publication process depends on only one employee or if there is no proper way of identifying software versions.

- The supplier delivers unsafe or insecure software because the wrong version of the software was published intentionally by an attacker/malicious insider who modified the software before publication.

- The supplier delivers unsafe or insecure software because severity of *known* hazard was downplayed and considered non-critical. This could happen because fixing issue and re-certifying software would be costly & time consuming.

- The supplier delivers unsafe or insecure software because an insider/attacker inserted malicious code into the code base.

- The supplier delivers unsafe or insecure software because some tests were skipped due to lack of time or money or because the testing setup did not work as intended.

Control action is issued but not effective:

- The supplier publishes safe and secure software, but the software is modified after being published by the supplier. This could happen on the server or on the way to the repair station, or at the repair station itself.

- The supplier publishes safe and secure software, but the repair station installed the wrong version/software by accident

UCA 11.1: Supplier delivers update when update is unsafe or insecure

Software update is different from initial version insofar that the process leading to it is different, thus there can be additional scenarios. All the scenarios for initial software version should also be considered here.

Due to incorrect feedback or incorrect instructions from another controller:

- N/A

Due to process model flaw or algorithm flaw:

- The supplier ships an unsafe or insecure update because the bug fix/ update introduced new hazards for one of the following reasons:
    - the code change was not properly reviewed before publishing
    - the effect of change on interacting components was not considered
    - none or insufficient tests were added for the new code
    - the fix violated a system design assumption which was not properly documented

- The supplier ships an unsafe or insecure update because the update did not fix the underlying problem but only patched one instance of the bug.

Control action is issued but not effective:
- The supplier issued a safe and secure update, but an attacker tricked the repair station into downloading a modified version the software from their own website. They could do this by sending a forged letter or e-mail to the repair station which instructs them to download the software from the attacker's website.

UCA 7.1, 7.2: Supplier ships update too late or not at all when update is critical

Due to incorrect feedback or incorrect instructions from another controller:
- The supplier ships an update too late or not at all because the supplier is not aware of hazards in original software. This could happen if the software is not identified as causal factor in accidents or because the supplier does not get necessary feedback/reports from pilots and mechanics in the first place. The software may not be detected as the cause because the hazard arises only in rare conditions / for rare combinations of variables.

- The supplier decides not to ship an update for a known issue because they are not legally required to do so as long as the FAA does not issue airworthiness directive. The FAA may not issue an airworthiness directive if the hazard is not recognized as serious because the exact effects are hard to understand.

Due to process model flaw or algorithm flaw:
- The supplier ships an update too late because the update depends on an engineer that is unreachable (left company, sick, on vacation etc.). This could occur if the software is not documented sufficiently well for other engineers to understand it quickly enough.

- The supplier ships an update too late because of a lack of personnel

- The supplier ships an update too late because he has no incentive (financial, legal, etc.) to provide update in timely manner

- The supplier does not ship an update because it is too costly and the company is in financial trouble.

Control action is issued but not effective:
- The supplier issues an update, but the update is not installed by repair station because a corresponding service bulletin is not issued, or the service bulletin is issued but does not reach the clients.

- The supplier issues an update, but the repair stations mistake unfixed version with fixed version and do not notice their mistake soon enough.

UCA 8.1, 8.2: Supplier issues service bulletin too late or not at all when FLS is unsafe or insecure

Due to incorrect feedback or incorrect instructions from another controller:
- N/A

Due to process model flaw or algorithm flaw:
- The supplier issues a service bulletin too late or not at all because the internal processes do not ensure that a service bulletin is issued whenever an update or new instructions are available.

Control action is issued but not effective:

- The service bulletin is issued, but it is not received by all repair stations because letters are lost in the mail, or an e-mail server did not deliver the e-mail.

- The service bulletin is issued and received by the repair station, but the repair station does not act on it because the service bulletin does not set a deadline.

FAA UCAs

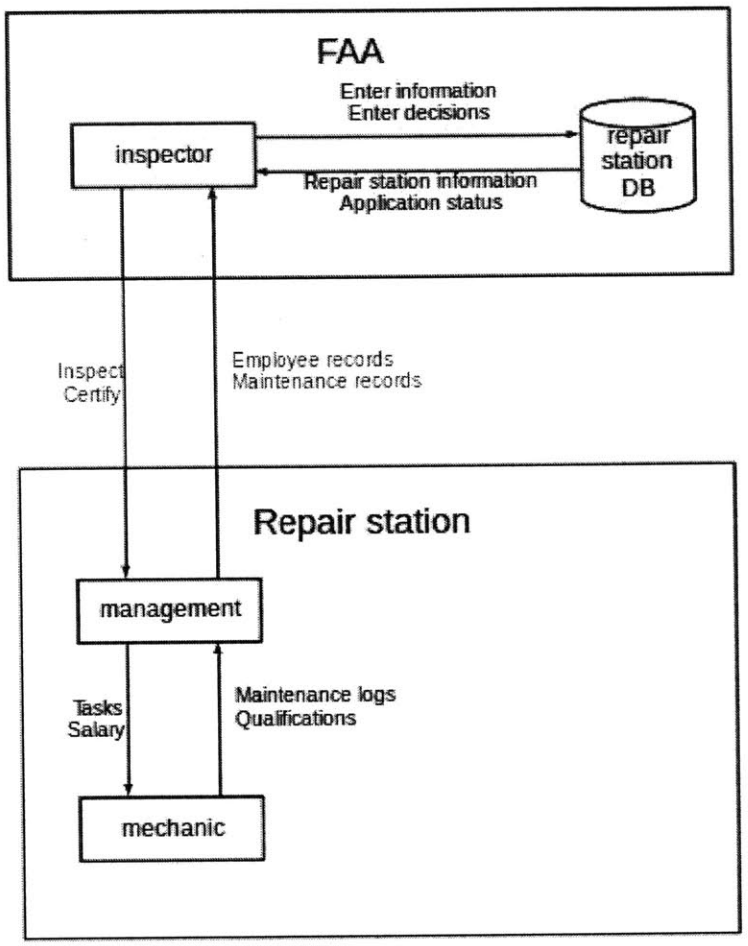

Figure 6: A more detailed view of the control loop between FAA and repair station

UCA 9.1, 10.1: FAA provides a type certificate for unsafe FLS or FLS software or update

Due to incorrect feedback or incorrect instructions from another controller:
- A type certificate is provided for unsafe software because the software tested on aircraft is not the exact software certified/delivered. This could happen if the software is not tracked appropriately with versioning.

Due to process model flaw or algorithm flaw:
- A type certificate is provided for unsafe software because someone hid a flaw on purpose

(could be an employee of the supplier or an FAA official)

- A type certificate is provided for unsafe software because FAA employees are under pressure not to deny too many type certificates

- A type certificate is provided for unsafe software because the designated representative feels pressure to sign off on certificate because he/she is afraid of losing the job.

- A type certificate is provided for unsafe software because tests/checks are not thorough enough because the supplier has good track record with the FAA.

- A type certificate is provided for unsafe software because tests and inspections are insufficient to discover a hazardous flaw if the flaw only shows up in rare situations (like the one in BA flight 38)

- A type certificate is provided for unsafe software because the certificate lists a software version different from the one that was intended be certified due to a mix-up.

- A type certificate is provided for unsafe software because configurations that were not tested were certified based on their similarity with tested configurations. In this case, a subtle difference could be overlooked.

Control action is issued but not effective:
- NA

UCA 10.2, 10.3: FAA provides certification for a critical update too late or not at all

Due to incorrect feedback or incorrect instructions from another controller:
- FAA certifies a critical update too late or not at all because the request for certification gets lost in the mail (or someone's inbox).

- FAA certifies a critical update too late or not at all because the supplier did not submit all required documentation in the right format

Due to process model flaw or algorithm flaw:
- FAA certifies a critical update too late or not at all because the FAA does not have enough inspectors to process all the certification requests in time

- FAA certifies a critical update too late or not at all because the application gets stuck on someone's desk at the FAA and nobody else takes over the task.

- FAA certifies a critical update too late or not at all because the application is not treated with the necessary urgency. This could happen by accident if there is no formal way of prioritizing important applications

Control action is issued but not effective:
- The certification is issued but it gets lost in the mail (or someone's inbox) and nobody notices for several days or weeks.

UCA 11.1, 11.2: FAA issues airworthiness directive too late or not at all for hazardous FLS

Due to incorrect feedback or incorrect instructions from another controller:
- The airworthiness directive is issued too late or not at all because no incidents are reported to the FAA despite hazard. This could occur for one of the following reasons:
  - because hazard only occurs in rare circumstances
  - because no evidence of software issues can be found after the fact
  - because reporting is too onerous for repair stations or owners and pilots
  - because repair stations prefer to report directly to the supplier
  - because supplier doesn't report hazard to FAA
  - because mechanics or pilots do not consider the issues serious enough to warrant FAA report

- The airworthiness directive is issued too late or not at all because reported incidents never make it to the FAA due to a flaw in the system. Nobody notices the flaw, because there is no follow-up for reported incidents.

- The airworthiness directive is issued too late or not at all because reported incidents do not contain all necessary information to pinpoint the problem source.

Due to process model flaw or algorithm flaw:
- An airworthiness directive is issued too late or not at all because reported incidents are considered pilot errors
- An airworthiness directive is issued too late or not at all because reported incidents do not get dealt with in efficient manner, for example if the FAA cannot deal with the large number of reports due to bureaucratic overhead.
- An airworthiness directive is issued too late or not at all because discovered hazards are not considered serious enough to warrant an AD. This could occur if the supplier assures the FAA that they will deal with the problem themselves or if the FAA trusts the supplier's assurances that the hazard is not serious.

Control action is issued but not effective:
- FAA issues an airworthiness directive, but it is not effective because it was not noticed by all owners/pilots of the concerned aircraft.

List of unsafe control actions for General Aviation FLS

Controller	Control action	Providing causes hazard	Not providing causes hazard	Too soon/ too late causes hazard
Repair station	initial install of FLS	(1.1) Install software not as specified by type certificate → H3, H4	(1.2) Software initial install not performed when component is new → H4	
	Update FLS	(2.1) Update FLS not as specified by AD / service bulletin → H3, H4	(2.2) Update FLS not provided in accordance with AD or service bulletin → H3	
Pilot	Update terrain DB	(3.1) Load terrain DB other than official one → H2	(3.2) Not loading terrain database with current obstacles → H2	
	pull up + TOGA	(4.1) low fuel or conflicting traffic → H1	(4.2) EGWPS real terrain alert → H2	(4.3) EGWPS real terrain alert → H2
EGPWS	sound warning		(5.1) Aircraft is about to hit terrain or ground structure → H2	(5.2) Warn too late of terrain → H2
Supplier	Deliver software	(6.1) Software is delivered that is unsafe or insecure → H3, H4		
	Deliver update	(7.1) Update is delivered that is unsafe or insecure → H3	(7.2) Update is not delivered when current version of FLS is unsafe or insecure → H3	
	Issue service bulletin		(8.1) Service bulletin is not issued when current FLS is unsafe or insecure → H3, H4	(8.2) Service bulletin is issued too late when current FLS is unsafe or insecure → H3, H4
FAA	Provide initial type certificate	(9.1) Provide type certificate when FLS software is unsafe or insecure → H3, H4		
	Provide type certificate for update	(10.1) Provide type certificate when FLS update is unsafe or insecure → H3, H4	(10.2) Type certificate for update is not provided when current FLS is hazardous and unsafe or insecure → H3, H4	(10.3) Certificate for update is provided too late when current FLS is unsafe or insecure → H3
	Issue airworthiness directive		(11.1) Airworthiness directive is not issued when current FLS is unsafe or insecure → H3	(11.2) Airworthiness directive is issued too late when current FLS is unsafe or insecure → H3

Table 8: Unsafe control actions for General Aviation