Mestrado em Engenharia Informática
Dissertação
Relatório Final

Risk-driven Security Assessment of Quadcopter's Flight Controller

Daniel Filipe da Cunha Martins Mendes
dfmendes@student.dei.uc.pt

Orientador:
Prof. Dr. Henrique Santos do Carmo Madeira

Co-Orientadores:
Dr. Naghmeh Ramezani Ivaki

Data: July 2, 2018

FCTUC FACULDADE DE CIÊNCIAS E TECNOLOGIA
UNIVERSIDADE DE COIMBRA

Acknowledgements

Being this final project a very important mark on my academic life, there are a couple people whom I need to thank for all the effort they put onto my success.

I would like to start by thanking Dr. Naghmeh Ivaki, who welcomed me to the *Software and Systems Engineering* research group, which led to this project. Dr. Naghmeh is a great professional and a great person, who I admire. Her attention to detail and motivation with my work always made me want to do better and improve myself.

I would also like to thank Professor Henrique Madeira, who vast experience and knowledge always brought a new perspective to my work and critical thinking. Being a very busy person, always had time to give feedback and counsel, which was very important to the success of this project.

Lastly, but not least I would like to thank my parents. Although they had not a direct influence of this project, their support and encouragement always made me work harder and improve myself. I am very grateful for being born with such remarkable parents.

Abstract

Unmanned Aerial Vehicles (UAVs) are no longer exclusively military and scientific solutions. These vehicles have been growing in popularity among hobbyist and also as industrial solutions for specific activities. The flying characteristics and the absence of a crew on board of these devices allow them to perform a wide variety of activities, which can be unaccessible to humans or may threat their life. Despite the advantages, they also bring up major concerns regarding security breaches in the flight controller software, which may lead to security (e.g., vehicle hijacking by attackers), safety (e.g., crashing the vehicle into a planned area or building), or privacy (e.g., eavesdropping or stealing video footage) problems. Since building a flawless software systems is a complicated task, if not impossible, a comprehensive security test is required to effectively detect and remove security vulnerabilities from the flight controller. However, executing a complete and exhaustive security test is very expensive and time consuming. For this reason, we use a risk-driven security testing approach, in order to identify and focus on the most risky components or states of the flight controller system. The main objective of this internship is to disclose the security vulnerabilities of a commonly used UAVs' flight controller, namely ArduCopter. The vulnerabilities will be exploited by execution of the tests, which will emulate an attack, that will be defined based on the results of a risk analysis process. In this report, we present the architecture of the System Under Assessment (SUA), the threat-modeling and risk analysis performed to identify the most risky components, leading to an effective security assessment. The outcome of the security assessment led to the testing of the system under GPS Spoofing attacks, where the GPS messages received by the quadcopter are tampered or delayed. It is presented the experimental setup used for testing, the result's analysis of the outcomes from the tests, and propose some defense mechanisms.

Keywords

Security, Safety, Privacy, Vulnerabilities, Threat model, Risk Analysis , Unmanned Aerial Vehicle (UAV), ArduPilot Flight Controller, GPS Spoofing

This page is intentionally left blank.

Contents

This page is intentionally left blank.

Acronyms

EKF Extended Kalman Filter. 18, 30–33

GCS Ground Control Station. 30

GPS Global Positioning System. 10, 26

HAL Hardware Abstraction Layer. 31, 32

IMU Inertial Measuring Units. 17, 26

OS Operating System. 17

SITL Software in the Loop. 30, 32

SUA System Under Assessment. 12, 13, 24, 30, 32, 33

UAVs Unmanned Aerial Vehicles. ii, 1, 10, 14

This page is intentionally left blank.

List of Figures

This page is intentionally left blank.

List of Tables

This page is intentionally left blank.

Chapter 1

Introduction

A variety of models for Unmanned Aerial Vehicles (UAVs) have been appearing over the last few years, with different flying abilities and flight controllers. The most popular models are multi-rotor helicopters, known as quadcopters, because of their range of applications, simple mechanical design, flight abilities, and variety of solutions and prices. This type of vehicles have a large number of commercial, industrial and academical uses, due to their movement flexibility [20, 25], having 6 degrees of freedom (i.e., able to move and rotate over 3-axis). This agility allows them to have a full set of motions (i.e., horizontal and vertical flight, vertical landing and take-off, hovering), making it a suitable candidate for autonomous flight. Having all the features necessary for autonomous flight also makes these systems target for actions with malicious intents towards humans or facilities, with intention to hijack and control the vehicle, track people, take them down, or eavesdrop them.

This kind of vehicles are usually very easy and cheap to build, with a few sensors like Inertial Measuring Units (e.g., gyroscope, accelerometer), Barometers and GPS (i.e., mandatory for autonomous flights), having a lot of literature on how to build one, but it also makes them easy targets for hijacking, hacking, or deviation of their original mission. When building a flight controller, these issues are not usually taken into account, since it is already complex to deal with the unsteady environment and faulty components or sensors on their own [17, 27], which raises the complexity of the system and therefore the security risks increase. The bigger and more complex a system is, the higher the odds of existing vulnerabilities in it, because there are more states and vulnerabilities to take into account.

In the case of quadcopters, security may bring attached other issues like safety and privacy. A safety problem is defined as the absence of catastrophic consequences on the user(s) and the environment [15]. When there is a malicious attack to take the quadcopter down or to lock it out preventing to be controlled by legitimate user, it may cause harm to everyone on its surroundings or even damage facilities and vehicles. Privacy concerns with the collection, storage or use of sensitive information (e.g., information that identifies a person or entity) [38]. Attacks against privacy can be harder to detect, since an attacker can be eavesdropping or stealing video footage without interfering in the behavior of vehicle. As explained in [36], an attacker can easily run malicious software on the vehicle's system, and intercept the video streaming. Considering the possibility of hijacking the vehicle, it is very easy to track and follow a person, which is a major privacy issue. Security should be then a major concern when working on a quadcopter system. Although there are already some mechanisms used in UAVs to tolerate erroneous input from the sensors (e.g., redundancy, sensor fusion and data filtering), they just cover a small part of the problem and are not reliable, since the time span of effectiveness for this mechanisms is very limited. [27].

To deal with the above issues (security, safety, privacy) in a quadcopter system, and in order to detect and remove security vulnerabilities, a comprehensive security testing is required, which can be very expensive and time consuming, specially with a complex system like a flight controller, which includes many components and states. For this reason, we aim to use a risk-driven security testing to perform an effective testing over components that are more risky, identified as a result of risk analysis. Risk analysis [34] is the quantitative or qualitative analysis of risks existing in a system. This process involves identification of threats, which is done by applying the STRIDE threat modeling scheme, and measurement of their risk level, which is done by applying the DREAD risk analysis scheme. The results obtained by this analysis are used to identify the most vulnerable and risky states or components, which should be tested more intensely. Risk analysis improves the testing process by identifying the components and states with hight risk and providing a means to prioritize them.

In this internship, we aim to assess the security of a quadcopter's flight controlling software system by using a risk-driven approach. Using this approach, components and states that present high risk to the system will be identified and prioritized, in order to effectively plan the security tests. Our main objective is to identify security threats to quadcopters and detect security vulnerabilities that exist in a quadcopter, which uses a commonly used flight controller, namely ArduPilot [3], and its' communications with other devices (e.g. ground stations, radio controllers). In this report, we present state of the art of security testing and risk-driven approach, a profound description of the System Under Assessment (SUA), threat modeling and risk analysis performed over SUA, description of the experimental setup used for testing the system in presence of emulated attacks to the GPS component, the analysis of the test outcomes, and propose some defense mechanisms.

This report is organized as follows. Chapter 2 is dedicated to the state of the art and related work, in which the main concepts related to security testing, threat models, and risk analysis are presented. We also review several works that have been done in the scope of risk-driven security testing of UAVs. In Chapter 3, the main objectives of this internship, the approach used to reach these objectives, and work plan are presented. In Chapter 4, system under assessment (SUA) and its architecture are described. Chapter 5 presents the risk analysis and corresponding results. Chapter 6 is described the experimental environment used, the fault model, the type of attacks and the results classification. Chapter 7 presents the analysis of the test results and proposes some defense mechanisms. Finally, Chapter 8 concludes this report.

This page is intentionally left blank.

Chapter 2

State of the art

In this Chapter, we first review the main concepts related to software security and its attributes, security testing, threat models, and risk analysis. We then review several works that have been done in the literature within the scopes of risk-driven security testing and UAVs security testing.

2.1 Software Security

Software is just about everywhere, and we are relying on it for almost everything. It is produced by humans, meaning that it may have faults and vulnerabilities, which can be exploited (for instance by an attack), causing security issues. The increasing use of software in critical systems, raises the importance of security as quality requirement. Security is a property of an entire system, which means that it does not refer just to a security mechanism or features built on a system (e.g., encryption, authentication, firewalls) [12]. Usually, when the objective is security assurance of an information system, three attributes are taken into consideration: Confidentiality, Integrity and Availability [4]; but with a critical controlling system, like UAVs' s flight controller, security assurance covers a wider range of attributes, like the following ones [4, 8]:

- **Confidentiality**, refers to the absence of unauthorized disclosure of information;
- **Integrity**, refers to the absence of improper system alteration;
- **Availability**, refers to the readiness of the system to provide correct service;
- **Non-repudiation**, refers to the ability of a system to assure that a given action cannot be denied by his author;
- **Authorization**, refers to the ability of a system to allow access to information or a service, for users with the demanded privileges;
- **Safety**, refers to the fact that the system should not endanger or jeopardize the health of individuals, environment, or associated assets.

These properties should exist in order to make software behave correctly and secure in presence of a malicious attack or even in a spontaneous action, without malicious intentions, changing the systems state. However, the complexity and interconnectivity of software systems is increasing, making it harder to have flawless code. Bigger projects mean bigger development teams, composed by people with different trainings and backgrounds. It only takes a small vulnerability to compromise an entire system, thus, covering all possibilities becomes the biggest obstacle to software security assurance. Poorly written code can lead to defects with security ramifications, so software security assurance should start earlier at the software development phase.

2.2 Security of UAVs

There are growing concerns over UAVs, in particular quadcopters, regarding security, privacy, and safety. Due to the lack of authentication mechanisms in off-the-shelf quadcopters, they can easily be hijacked to, for instance, track and monitor people [16].

Moreover, these vehicles can easily be target of GPS spoofing attacks. Although customized solutions like military solutions, as shown in [32], use authenticated GPS, most of quadcopters use civilian GPS, which is unauthenticated.

Another security concern over UAVs is regarding the lack of encrypted connections. Either using ground stations over Wi-Fi or using communications protocols, like MAVLink [1], these connections can be easily hacked by someone within the connection range of the vehicle, as shown in [9, 41].

Furthermore, the off-the-shelf quadcopters also been used to exploit safety issues. They have been used to smuggle contraband for prisoners [5] or altered to be used as attack drones [41]. Some countries already took actions against UAVs with video footage, like Sweden, which banned them unless a plausible reason for filming was presented, preventing illegal vigilance [26, 42].

2.3 Software Security Testing

Security testing [34] is the process of discovering security vulnerabilities that exist on a certain system and that threat some of the system properties like confidentiality, availability, integrity, authorization, availability, or non-repudiation. There are several techniques to detect or disclose security vulnerabilities of a software system. In general, such techniques can be divided in three categories:

- Penetration Testing
- Code Review
- Security Monitoring using Anomaly detection tools

Penetration Testing [30] is done by emulating an attack on a computer system, where it is attempted to gain access to resources without having normal means of access. The difference between this test and a real attack, is the authorization granted to the tester to perform this actions, with the goal to assess the security of the system and identify vulnerabilities.

Code reviews [29] is a process where two or more agents, that visually analyze the code, attempting to identify defects, bad practices, vulnerabilities or potential malicious code.

This type of testing, involve a set of standard forms, in order to save information about the problems found in the code, who found it, where it was found, in what category it is in and what was the disclosure on the Review meeting. Code review can are used in formal inspections or in more informal processes (e.g. pair programming, walkthroughs).

In Security Monitoring using Anomaly detection tools [28], a mechanism checks for pre-defined conditions during runtime of the system, detecting threats or strange behavior in real time. This method allows to detect and try to correct anomalies at runtime, although only for already known and defined threats.

Although there are different mechanisms for security testing, the testing process usually follows a common procedure. According to [19], the first step is **Test Planning**, where the systems requirements, security goals and objectives, and test mechanisms are defined. Then the second phase of security testing, **Test Design and Implementation**, starts. In this phase, tests cases and the workload and expected output for each input are designed and implemented. At this stage, it is also necessary to collect or implements tools that are necessary to execute the tests in the next stage, **Test Execution**. At this point everything should be ready to execute the security tests. The security tests are executed and the output is logged for further analysis. The last stage belongs to **Results Analysis**, where the output of the tests is compared with the expected output from the test cases, in order to detect erroneous behavior by the system, caused by some vulnerabilities and bugs existing in the system.

Even after an exhaustively testing of all the paths of a program we cannot guarantee that it is free of vulnerabilities, as Dijkstra said *"Testing can show the presence of bugs, not their absence."* [11]. Testing has its limitations and they grow bigger as the software complexity increases. It is not possible to guarantee that all hypothesis and states were tested, either because testing is expensive or because humans cannot cover all possibilities, and new technologies, attacks and methodologies appear every day. Also, software is not static, and each change invalidates the previous testing phase, since it creates new states and alter the previous ones, which may lead to undiscovered vulnerabilities [10]. Even testing tools and testers aren't immune to bugs. A misconception or bad implemented tester may present different results, with less vulnerabilities or vulnerabilities that don't exist, which leads to misleading conclusions and insecure software [11].

2.4 Threat Modeling

Threat modeling [34] is a procedure where threats are identified and assessed, organizing under a certain model and analyzed. A threat to the system is any potential or actual undesirable event with or without malicious intent, that may cause harm to a system or it's environment. There are many different threat models, suitable for different contexts [31]. In the following sections, we present the ones that are commonly used in the literature.

2.4.1 STRIDE

STRIDE [24] is a threat classification model for known threats, developed by Microsoft. This acronym is used to classify threats according to the kind of exploit used and impact or motivation of the threat. Based on STRIDE, the threats are categorized as follows:

- **Spoofing Identity**, refers to when a user is able to became or use the attributes of another user;

- **Tampering with data**, refers to malicious modifications of data;
- **Repudiation**, is associated to users denying performing an action without any way to prove otherwise;
- **Information Disclosure**, involves the disclosure of information to other users that should not have access to it;
- **Denial of service**, refers to attacks that deny service of valid users;
- **Elevation of privilege**, refers to when an unprivileged user gains permission to access privileged areas an information, thus has sufficient power to compromise or destroy the system.

2.4.2 VAST

This acronym stands for *Visual, Agile and Simple Threat* modeling [2]. This modeling scheme is designed to be used within an Agile Methodology. It is specifically designed to overcome the scalability problems of other methodologies and provide actionable outputs for the various stakeholders within the project scope, without requiring specific security experts.

2.5 Risk Assessment

Risk Assessment [34] aims at identifying the risks associated with a system and prioritizing them. To measure the risks, several parameters including complexity, severity, and impact are taken into consideration, leading to privatize them. There are several schemes to classify risks [31]. In the following sections, we present the ones that are commonly used in the literature.

2.5.1 DREAD

DREAD is a risk assessment model for known risks or threats, that is used to qualify, compare, and prioritize the level of risk for each threat [34]. DREAD uses five categories, including Damage, Reproducibility, Exploitability, Affected users, and Discoverability, to rate the security threats usually between 0 and 10, The final rate of each threat is an average of the scores of these categories. Table 2.1 presents these categories, their scales and values dedicated to each scale.

Threat	Description	Value
Damage Potential	Level of damage that will occur	0 - no damage 5 - data is compromised or affected 10 - complete system or data destruction
Reproducibility	The difficulty of reproducing the exploit	0 - very hard or impossible 5 - one or two steps for authorized users 10 - just a web browser and the address bar is sufficient, without authentication.
Exploitability	The level of resources needed to exploit the threat	0 - custom or advanced tools and advanced programming and network knowledge 5 - available tools and malware 10 - just a web browser
Affected Users	Value for the extension of users affected	0 - none 5 - some but not all 10 - all users
Discoverability	The difficulty of discovering the threat	0 - very hard or impossible 5 - possible by monitoring network traces or guessing 9 - known at public domain and easily discovered 10 - visible in an address bar or form

Table 2.1: Threats Definition and Values [31]

2.5.2 P.A.S.T.A.

This risk-centric methodology stands for *The Process for Attack Simulation and Threat Analysis* [39] and consists of seven steps, which allow a dynamic threat identification and scoring process, based on parameters like probability of attack, threat likelihood, inherent risk and impact of compromise.

Stage	**Description**
1. Define Objectives	- Identify Business Objective - Identify Security & Compliance Requirements - Business Impact Analysis
2. Define Technical Scope	- Capture the boundaries of the technical environment - Capture Infrastructure, Application & Software - Dependencies
3. Application Decomposition	- Identify Use Cases, Define App Entry Points & Trust Levels - Identify Actors, Assets, Services, Roles, Data Sources - Data Flow Diagramming, Trust Boundaries
4. Threat Analysis	- Probabilistic Attack Scenarios Analysis - Regression Analysis on Security Events - Threat intelligence Correlation & Analytics
5. Vulnerability & Weakness Analysis	- Queries of Existing Vulnerabilities Reports - Design Flaw Analysis - Scorings, Enumerations
6. Attack Modeling	- Attack Surface Analysis - Attack Tree Development - Attack to Vulnerability & Exploit Analysis
7. Risk & Impact Analysis	- Qualify & Quantify Business Impact - Countermeasure Identification - Risk Mitigation Strategy

Table 2.2: P.A.S.T.A. Process Stages [39]

2.5.3 CVSS

CVSS stands for *Common Vulnerability Scoring System* [18] and, using the principal characteristics of the vulnerabilities, scores them numerically , based on the severity, which can also be translated into qualitative representation. It has three types of metrics, Base, Temporal and Environmental. The Base metric represents the vulnerabilities' characteristics, while Temporal reflects the characteristics' that change over time and Environmental show the aspects unique to a particular environment. This last two groups refine the Basic group metrics' values.

Exploitability Metrics	**Impact Metrics**
Attack Vector	Confidentiality Impact
Attack Complexity	Integrity Impact
Privileges Required	Availability Impact
Scope	Scope
User Interaction	

Table 2.3: Base Metric Group

Temporal Metrics
Exploit Code Maturity
Remediation Level
Report Confidence

Table 2.4: Temporal Metric Group

Environmental Metrics
Modified Base Metrics
Confidentiality Requirement
Integrity Requirement
Availability Requirement

Table 2.5: Environmental Metric Group

2.6 Risk-driven security testing

Risk-driven Security Testing is a security testing methodology where Risk Analysis is used in order to achieve the most important security test cases [14], diminishing the number of tests needed and focusing on the most critical ones. This type of testing is very useful, specially on complex system with many different states. In [43] is proposed a model-based methodology for risk-driven security testing of centric systems. In order to improve testing efficiency, [33] used a threat modeling approach, STRIDE, to identify highly risky states, resulting in a reduced and more efficient test suite. Medical devices are critical systems, being safety assurance one of the main concerns for developers of this type of software. In [6] risk-driven approach of security testing is used to improve test design, in order to detect more safety risks.

We can find several works in the literature addressing the security challenges and threats of UAVs' systems and the privacy and safety issues it raises. In [32], the authors perform several attacks on one of the main flight components, namely Global Positioning System (GPS), to demonstrate how it can be used to hijack UAVs. Due to the emergence of Unmanned Aerial Vehicles (UAVs) as emergency tools or as weapons, in [23], a threat modeling and analysis approach is used to identify high priority threats and mitigate them. Also in [22], a risk assessment approach was used to evaluate storage, sensorial information, communication system and fault handling mechanism of some vehicles. In [7], a monitoring system, capable of collect flight data and analyze it in real-time, was developed to search for abnormal behavior.

The authors in [41] studied exploits from an attacker within Wi-Fi range of the vehicle, in order to hijack, steal user data or take-down commercial solutions of best-selling brands for drones. Security threat analysis of AR.Drone was performed in [36], and exploited the most obvious security vulnerabilities, using attacks of high-jacking, eavesdropping video streaming and people tracking.

This page is intentionally left blank.

Chapter 3

Research Objectives and General Approach

In this Chapter, the main objectives of this internship, the approach chosen to achieve these goals and, the work plane are presented.

3.1 Objectives

The main objectives of this work are focused on:

- Performing a threat (threats to security, privacy and safety) and risk analysis and rating over a commonly used flight controller of quadcopters, namely ArduCopter.
- Performing a security test on the System Under Assessment (SUA) based on the results obtained from the risk analysis to reveal security vulnerabilities, remove them, or develop defense mechanism against them.

From the threat and risk analysis, we aim to obtain a list of recommendations helping to improve the security of quadcopters. The security testing is accomplished based on the results obtained from the previous phase, to disclose and deal with the existing vulnerabilities in the risky components and states of ArduCopter in terms of security, privacy and safety.

3.2 Approach

To assess the security of SUA, a risk-driven approach is used. Figure 3.1 depicts the process to be followed. The first step belong to the **Study SUA Architecture and System States**. The risk analysis requires knowledge of the architecture and stats of the system in order to identify potential threats.

The next step is **Threat Identification** found on the system. To identify and organize these threats, a thread-modeling approach, called STRIDE [24], will be used. STRIDE will allow to identify technical and non-technical threats in each state or component of the system.

Listing the threats is followed by the **Risk Analysis**. In this stage, risk level associated with the threats is calculated using risk parameters. To do so,the DREAD risk assessment model [34] will be used, to assess the risk rate of each threat. An average of the risk parameters (or categories) of DREAD, presented on Subsection 2.5.1, will be used as the risk value for each threat:

$$Risk\_Value = \frac{Damage\_Potential+Reproducibility+Exploitability+Affected\_Users+Discoverability}{5}$$

Usually the risk value is comprised between zero and ten. The most risky threats are the ones with a risk value closer to ten.

With this results, we will be able to give some recommendations to improve the security of quadcopters in general. It will also be possible to **Identify Risky Components and States**. These are the components that will be under assessment on the testing stages.

The next step is **Test Planning**, which includes 1) Identify the components and states to be tested; 2) Identify what kind of tests should be done; and 3) Identify how the tests should be executed.

In the **Test Execution**, test cases are implemented and testing tools are gathered or implemented if necessary. Afterwards, all the test will be executed by the testing mechanisms and the results collected.

In the **Result analysis**, the exploited vulnerabilities need to analyze and categorize. From the analysis of the resulting list, it will be proposed solutions to deal with the vulnerabilities and remove them from the SUA.

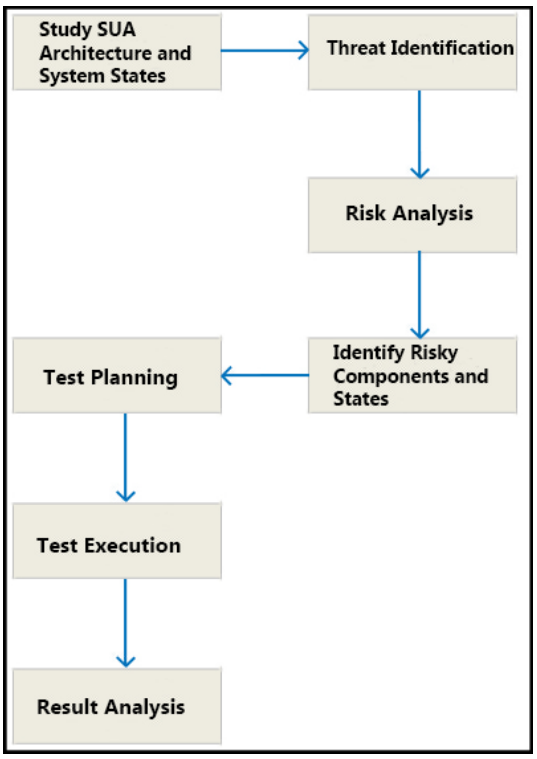
Study SUA Architecture and System States
Threat Identification
Risk Analysis
Identify Risky Components and States
Test Planning
Test Execution
Result Analysis

Figure 3.1: Risk-driven security testing approach

3.3 Work Plan

For the first semester, we have planned to accomplish the following tasks:

1. Collect information on software security, how it applies to UAVs and on security testing;
2. Study the system under assessment (ArduCopter) and detail the architecture and its' modules;
3. Preliminary Threat Identification and Modeling;
4. Preliminary Risk Analysis, over the Threat Modeling results;
5. Write the report.

Figure 3.2 presents the Gantt chart for our work plan in the first semester.

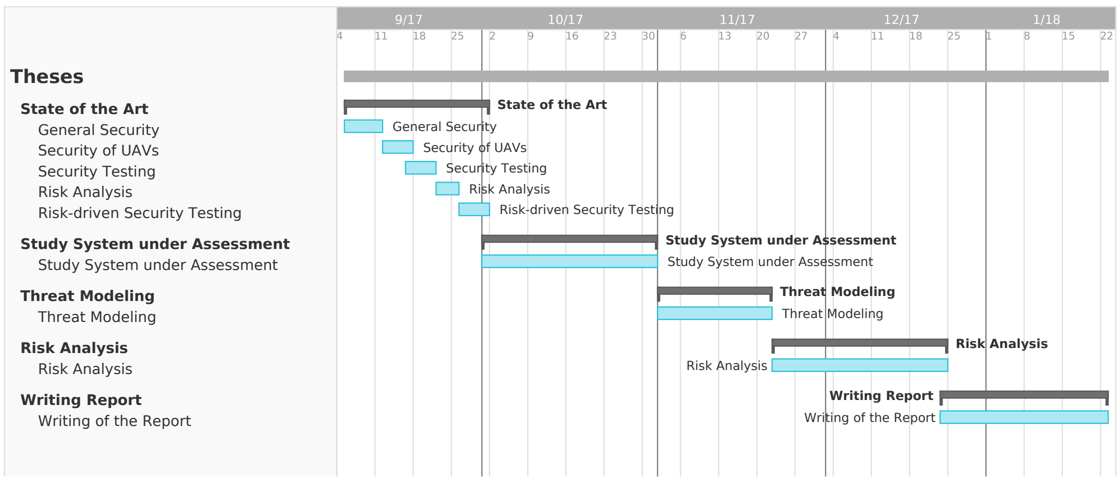

Figure 3.2: First semester schedule

For the second semester, the following tasks are planned to be accomplished:

- Complete threat modeling and risk analysis;
- Test planning;
- Test implementation and execution;
- Analysis of the results;
- Write the report;
- Write a conference paper.

Figure 3.3 presents the schedule plan for the second semester.

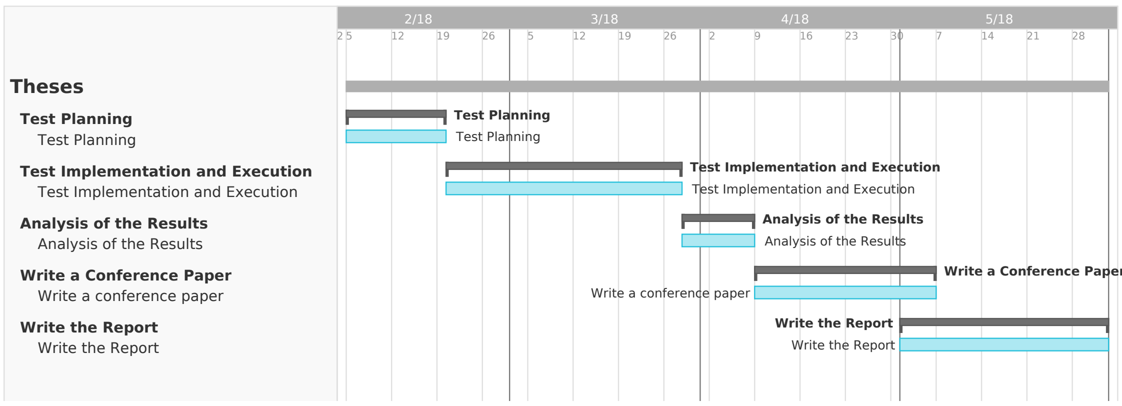

Figure 3.3: Second semester schedule

This page is intentionally left blank.

Chapter 4

System Under Assessment (SUA)

The system under assessment (SUA) is a quadcopter (helicopter-type drone with four rotors) that uses a Navio2 control board developed by *Emlid* [13] and Raspberry Pi [40], running on a *Raspbian* Operating System (OS) [21]. A high-level view of the SUA is presented in Figure 4.1:

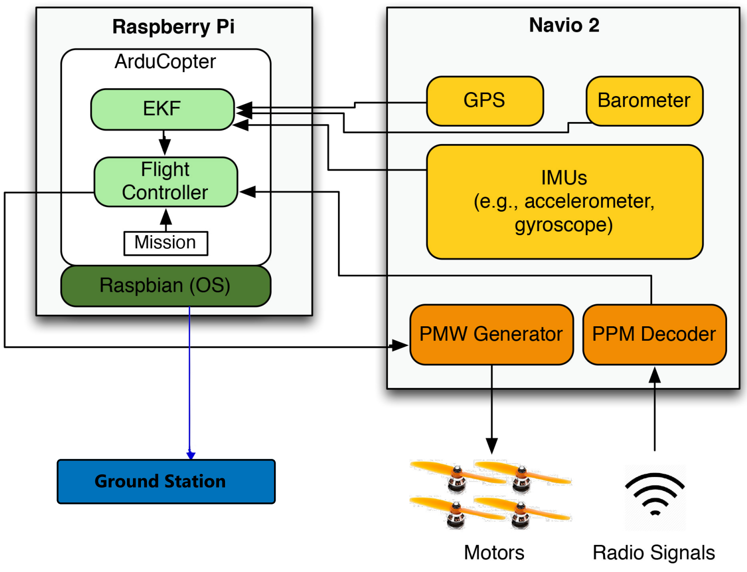

Figure 4.1: High-level view of SUA architecture

The Navio 2 Control Board includes the sensors (such as two Inertial Measuring Units (IMU), precision barometer and GPS module) and the components responsible for transforming the flight controller's messages into signals to the motors (*PWM Generator*) and the other way around (*PPM Decoder*). This board is responsible for bridging the information between the flight controller and outside components (e.g. receiving radio communication input, receiving sensorial input, send output signals to the motors).

The Raspberry is using the Raspbian OS, based on Debian LINUX computer OS, which has on top the flight controller ArduCopter being executed. ArduCopter has two main functions:

- **Sensor Fusion**, where it is implemented the Extended Kalman Filter (EKF) algorithm, that receives the information from the Navio 2 board relative to the inertial sensors, barometer and GPS readings, plus the information from pilot commands (some flight modes do not require pilot input) and outputs corrections needed to the flight control. Since the sensors send input within different time intervals, there is a complementary EKF to resolve sampling rates discrepancy.

- **Flight Control**, which calculate the position, altitude and orientation of the quadcopter for real-time navigation.

ArduCopter also makes the connection between the flight controller and the user interface (known as Ground Station), uses the Micro Air Vehicle Communication Protocol, commonly known as MAVLink. This is a very lightweight, header-only message marshalling library for micro air vehicles [1], which was first released in 2009 by Lorenz Meier.

The user makes uses of software applications, usually called *Ground Stations*, that are used to see real-time values of the system (e.g. orientation, altitude, battery), make changes to the vehicle current state (e.g. change flight mode) or create and send missions for autonomous flight based on GPS coordinates.

The Flight Controller is the center of all the system, responsible for handling all the data and calculations. It also has to deal with two different types of flight modes, *Manual* and *Autonomous*. Autonomous flight has a different flow of information and is more dependent of flight controller action, in order to correct is positioning and attitude, while Manual modes add input from the pilot to the flight control algorithm, but its less dependent on the positioning system of the flight controller, as seen in the Subsections 4.2.1 and 4.2.2.

4.1 Flight Controller

The main component of the system under test is the flight controller, ArduCopter. A more detailed architecture view can be seen in the Figure 4.2. This modules represent how the information travels from the sensors and pilot commands, to the EKF and are used by a flight mode algorithm, to calculate the position and attitude of the vehicle, to finally send the required signal to the motors to maintain or change throttle or orientation.

Hardware Abstraction Layer, that handles all the types of different hardware input, from the different brands, and gives a standard response to the flight controller, removing this overhead from the main modules

The **Main Loop** is a very important module of the flight controller, being responsible for scheduling tasks, like managing the several sample rated inputs from the different sensors, handling the sensor fusing, among others. This module is part of the specific code for each vehicle, since it needs to handle different information and different modules for each vehicle (e.g ground vehicle, helicopters, fixed-wing vehicles).

Between the scheduler and flight control stage, there is the sensor fusion, handled by the EKF implementation. The sensor fusion and the background threat that receives the sensorial information belongs to the shared libraries. In the **Extended Kalman Filter** there are two main phases: State Prediction and Measurement Fusion. The State

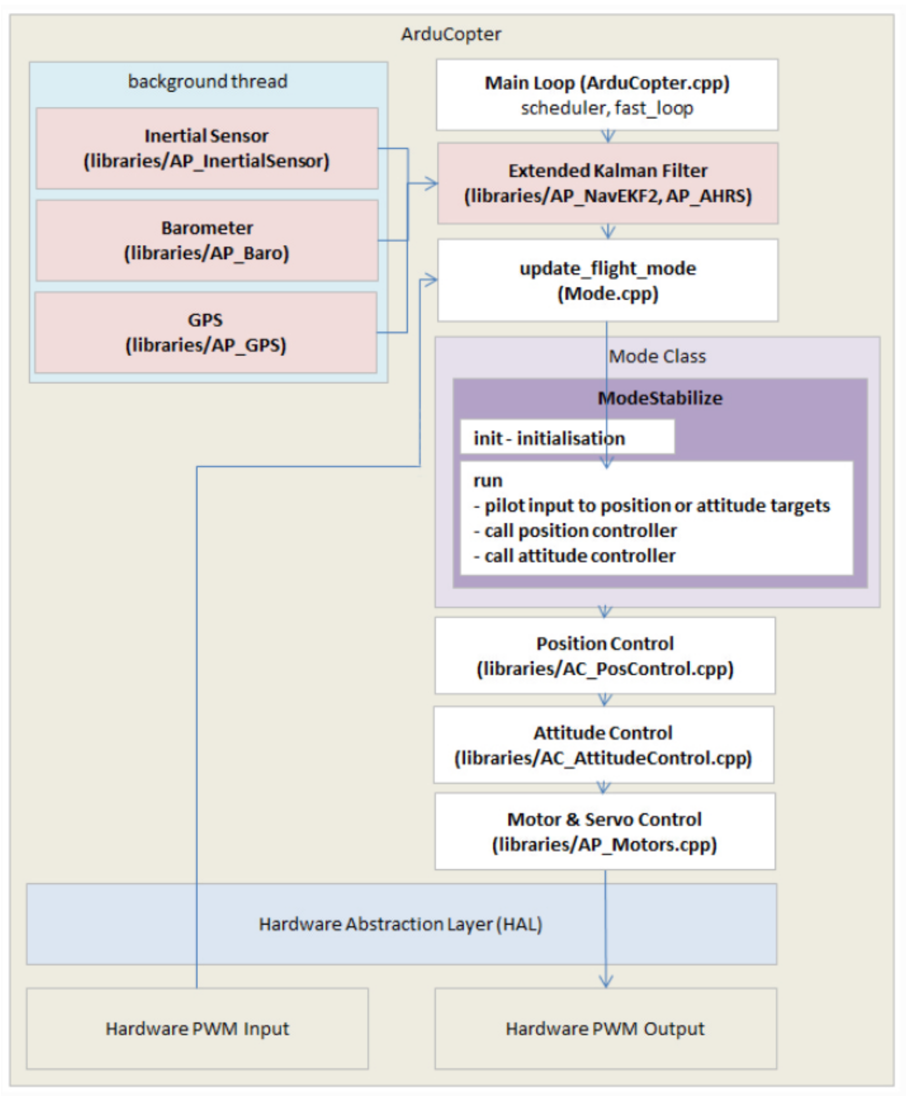
ArduCopter
background thread
Inertial Sensor (libraries/AP_InertialSensor)
Barometer (libraries/AP_Baro)
GPS (libraries/AP_GPS)
Main Loop (ArduCopter.cpp) scheduler, fast_loop
Extended Kalman Filter (libraries/AP_NavEKF2, AP_AHRS)
update_flight_mode (Mode.cpp)
Mode Class
ModeStabilize
init - initialisation
run
- pilot input to position or attitude targets
- call position controller
- call attitude controller
Position Control (libraries/AC_PosControl.cpp)
Attitude Control (libraries/AC_AttitudeControl.cpp)
Motor & Servo Control (libraries/AP_Motors.cpp)
Hardware Abstraction Layer (HAL)
Hardware PWM Input
Hardware PWM Output

Figure 4.2: ArduCopter Flight Controller architecture [3]

Prediction uses inertial navigation equations to predict changes in the position, velocity and orientation since the last readings, creating an estimated state were the vehicle should be. The Measurement Fusion filters the predicted state using the sensorial information, in order to calculate corrections needed to be applied to the predicted states, in order to achieve a precise current state of the vehicle.

Reaching the flight control phase, the information from the EKF is joined with the pilots command(if there are some) as inputs for a specific flight mode. The corrections calculated by the sensor fusion algorithm and the pilot input is used for Position Control and Attitude Control, which will send the information to the Motors Control, in order to change throttle, orientation, in order to match the correct state of the flight. From the Motors control, a *Pulse-width modulation* generator, will encode the messages from the flight controller into pulsing signals, allowing it to control the motors.

4.2 Flight Modes

There are many types of flight modes in the ArduCopter flight controller, but they can be split into two major groups: **Manual flight modes**, that require pilot commands to fly, and **AutoPilot flight modes**, that are fully or semi-autonomous flight modes.

4.2.1 Manual Flight Modes

Manual Flights need pilot input in order to fly. They can be assisted by some algorithms of the flight controller (e.g. Stabilize, Drift) or only use user input (e.g. Acro). In Figure 4.3 it is possible to see the information flow.

Module	**Description**
Flight_Mode	Checks flight mode variable and calls flight mode specific function.
Control_Stabilize	Interprets pilot inputs and sets target values for roll, pitch and yaw angles.
AttitudeControl	Calculates attitude error and converts them to high level motor requests.
MotorsMatrix	Converts high level motor requests into individual motor outputs.
RCOutput	Sendes PWM messages to each ESCs.

Table 4.1: Manual Flight Description

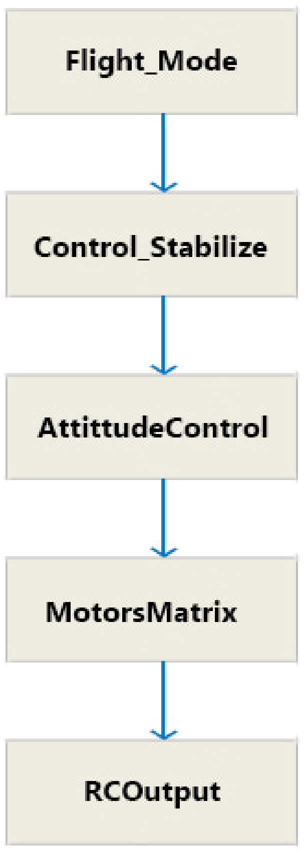

Figure 4.3: Manual Flight [3]

4.2.2 AutoPilot Flight Modes

AutoPilot Flight modes are heavily dependent of the flight controller and sensorial readings. Some of the flight modes require waypoints or previously planned missions (e.g. Auto, RTL). The information flow is presented in Figure 4.4.

Module	**Description**
Flight_Mode	Checks flight mode variable and calls flight mode specific function.
Control_RTL	Uses rtl_state variable to decide which sub function to call and calls waypoint navigation controller to get desired roll, pitch, throttle
WPNav	Calculates position and velocity error and updates PosControl targets.
PosControl	Calculates desired lean angles and throttle.
AttitudeControl	Calculates attitude error and converts them to high level motor requests.
MotorsMatrix	Converts high level motor requests into individual motor outputs.
RCOutput	Sendes PWM messages to each ESCs.

Table 4.2: Auto Flight Description

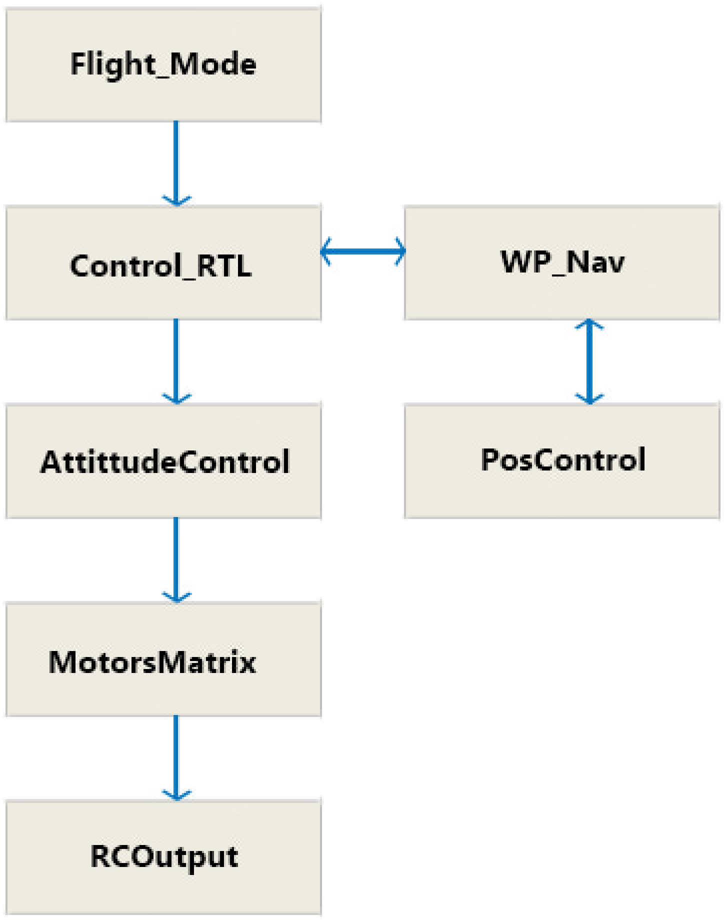

Figure 4.4: Auto Flight [3]

This page is intentionally left blank.

Chapter 5

Risk Analysis

In this Chapter, we present the work has been accomplished so far with regard to the threat identification and risk analysis of the SUA.

5.1 Threat identification

Malicious users can use their own quadcopters to perform actions that violate privacy or safety (e.g., track people, spy private facilities, explode remote weapons). This kind of issues fall out of the focus of our work and should be dealt by political parties and regulators, by establishing laws and rules to prevent this kind of activities. In this internship, the focus is on cases where an attacker try to perform an attack over a drone of other users.

The first step to identify the threats is to find entry points of the SUA where attackers can interact with the system and identify components and states that the attackers could be interested in. The threats detected on the flight controlling systemare then grouped based on the STRIDE Threat Model, which depending on how the threat affects a component, they are set in one of the six categories shown on Subsection 2.4.1.

Using the architecture of the SUA and the functionality of its components, presented in Chapter 4, we identifies the main inputs to the system and outputs from the system, resulted in drawing the data flow of the SUA. It allows to identify entry points and states of the system. Figure 5.1 presents the data flow of the SUA.

As shown in the Figure, there are two main entry points in the system, the sensor readings, coming from the sensors and the flight control commands coming from ground station through communication protocols. There is also vulnerable information on system, like flight logs or the flight mission, that can be the target of attacks.

From the study of the SUA, we observed that there are major threats to the system through the communication protocols. It is possible to connect to a given IP address and Port of the system without any type of authentication mechanism, making it an easy target for hijacking.

Moreover, attackers are able to fly away with a quadcopter or prevent the legitimate user to interact with him, either by impersonating a legitimate ground station which is

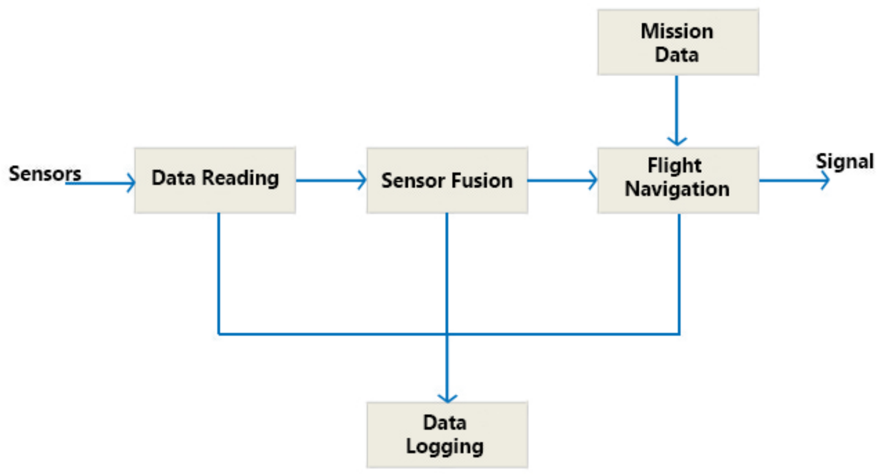

Figure 5.1: Data Flow of the System

sending flight commands or by blocking the connection of the real user, preventing him from recovering the control of the vehicle.

Furthermore, attackers can spoof the sensors and GPS and send erroneous data to the sensor fusion algorithm, which will affect all the system. Each sensor needs to be studied differently, since they do not affect the system on the same way [27].

An attacker can also tamper mission data information by changing the flight route on the mission file or by sending commands, impersonating a user using a ground station.

The flight logs are also vulnerable to malicious actions, being saved on a plain text file. Attackers can easily steal them or tamper them, without the user knowledge.

There is also the threat of an attacker access the video streaming. This allows malicious users to spy infrastructures or other people.

Table 5.1 lists the above threats and their category based on the STRIDE model. The security properties, presented below that is affected by the threat is also presented in the table:

- **Safety**, in case the threat causes danger towards humans or the environment;
- **Privacy**, when the threat allows access to sensitive information;
- **Integrity**, if the threat causes changes on the system state;
- **Repudiation**, in case the threat is able to change information about the drone usage;
- **Authorization**, when the attacker can access and send commands to the vehicle.

Name	**Description**	**Property**	**STRIDE Category**
IMU Spoofing	An attacker tampers the sensorial readings from the IMU.	Safety	Data Tampering Denial of Service
Barometer Spoofing	An attacker tampers the sensorial readings from the barometer.	Safety	Data Tampering Denial of Service
GPS Spoofing	An attacker tampers the sensorial readings from the GPS.	Safety	Data Tampering Denial of Service
Ground Station Spoofing	An attacker is able to send commands to the system.	Integrity	Elevation of Privilege Denial of Service
Access to Video Data	A malicious user views or saves the video stream / footage.	Privacy	Information Disclosure
Access to Log Data	A malicious user views or tampers the flight logs.	Repudiation	Data Tampering Denial of Service Repudiation
Access to Mission Data	An attacker is able to view and modify mission waypoints.	Integrity	Data Tampering Denial of Service Information Disclosure
Locking out the drone	An attacker prevents the legitimate user from connecting to it.	Authorization	Denial of Service
Flying the drone way	An attacker can change the course of the drone to his intents.	Authorization	Denial of Service

Table 5.1: Threat Modeling

5.2 Risk Rating

The attacker may exploit the vulnerabilities with two distinct scenarios in mind:

- Hijacking the quadcopter, stealing it from the legitimate user or to use in some malicious action.
- Denial of service, preventing the quadcopter to finish it's task, chosen by it's legitimate user.
- Data leakage, being flight mission details or video stream, compromises confidential information or can be used for spying.

So the damage of an attack is measured by how well it performs, based on the attack goal. If the goal is hijacking, an attack without any damage, is when the quadcopter is not deviated from his original path, and the highest damage of this attack is the malicious user running away or using the quadcopter to his own purpose. While in a denial of service attack, the highest damage result is to prevent the correct use of the quadcopter, while if the attack is unsuccessful, the quadcopter finishes it's task without any deviations in time or distance.

Although many quadcopter's software is open source, to discover a vulnerability it is necessary more than programming knowledge. And even if the vulnerability is discovered, it does not mean it can be exploited to create a certain behavior. So besides the existence of a vulnerability, the attacker must know the system logic in order to create a successful attack. Information about the attacks is relatively scarce, since attackers do not want their approach to be blocked by the flight controllers software developers, although there are some exceptions, like one of the most discussed attacks, GPS Spoofing [37].

When analyzing threats is also important to evaluate the ratio between the cost of an attack and the outcome of the attack. Even if a vulnerability is discovered it does not mean it is worth it to exploit it. Although an attack has a high damage potential, but the amount of resources or time needed to exploit the vulnerability make it impracticable, the threat is not as dangerous as it looks.

In order to identify the risky components, the threats identified previously are rated using the DREAD scheme. The risks' value will be obtained by doing an average of the five categories of this scheme, as shown in Subsection 2.5.1. The rating of the threats is presented in Table 5.2 from the highest to the lowest risk.

Name	**DREAD**	**Risk Value**
GPS Spoofing	Damage Potential - 10 Reproducibility - 10 Exploitability - 5 Affected Users - 10 Discoverability - 10	9
Ground Station Spoofing	Damage Potential - 10 Reproducibility - 5 Exploitability - 5 Affected Users - 10 Discoverability - 9	7.8
Access to Mission Data	Damage Potential - 10 Reproducibility - 5 Exploitability - 5 Affected Users - 10 Discoverability - 9	7.8
Sensor Spoofing	Damage Potential - 10 Reproducibility - 5 Exploitability - 5 Affected Users - 10 Discoverability - 5	7
Access to Video Data	Damage Potential - 5 Reproducibility - 5 Exploitability - 5 Affected Users - 10 Discoverability - 5	6
Access to Log Data	Damage Potential - 5 Reproducibility - 5 Exploitability - 5 Affected Users - 5 Discoverability - 5	5

Table 5.2: Risk Rating

This page is intentionally left blank.

Chapter 6

Experimental Setup

From the previously identified, *GPS Spoofing* is the most common and documented threat. In order to assess the tolerance mechanism of the SUA, several GPS Spoofing attacks were emulated. In this chapter it is detailed how the experiments were performed, the faults injected and how the results were analyzed.

6.1 Experimental Setup

The experiments were performed in the simulator, Software in the Loop (SITL), available on the ArduPilot repository. This software is responsible for simulating the physics of the quadcopter and of the environment, feeding the same flight controller software used on the real system (ArduCopter), through a a flight dynamics model. A high level view of the SITL and how it is included in the experimental setup is shown in Figure 6.1.

The SITL's flight dynamics model, simulates all the necessary hardware (e.g. Inertial sensors, motors, battery) and several environmental conditions (e.g wind speed and direction). All of this parameters can be tuned to approach the test environment, as much as possible, to the real system's flight. In our experiments, the noise-free setting was used, in order to guarantee that the results were not affected by noisy values or environmental disturbances.

As in the real quadcopter, the flight controller reads the mission file (which is stored locally) and sends the required commands to complete that mission. The output of each mission is logged by the ArduCopter software and is also stored locally. Since the quadcopter is turned on, the logs save detailed information of each aspect of the flight, such as sensorial data readings, GPS messages, vehicle's position, output of the EKF and, in the SITL's case, the quadcopter's position in the simulator. In order to order to download the logs from the SUA at the end of each flight, the simulator is connected to Ground Control Station (GCS), called *MAVProxy*, through a TCP/IP connection. To automate the experimental process, it was implemented a Python commander, using the DroneKit API [35], connected to the GCS through a UDP/IP connection.

The source code of the ArduCopter used in the experiments was instrumented with a fault injection module, which contains the methods responsible for reading, the previously created, files with the information of the faults to be injected in each flight. This module is also responsible for injecting the the faults in the proper time span (defined in the fault injection file). The faults are injected right after the GPS messages are processed by the

Hardware Abstraction Layer (HAL), before being fed to EKF.

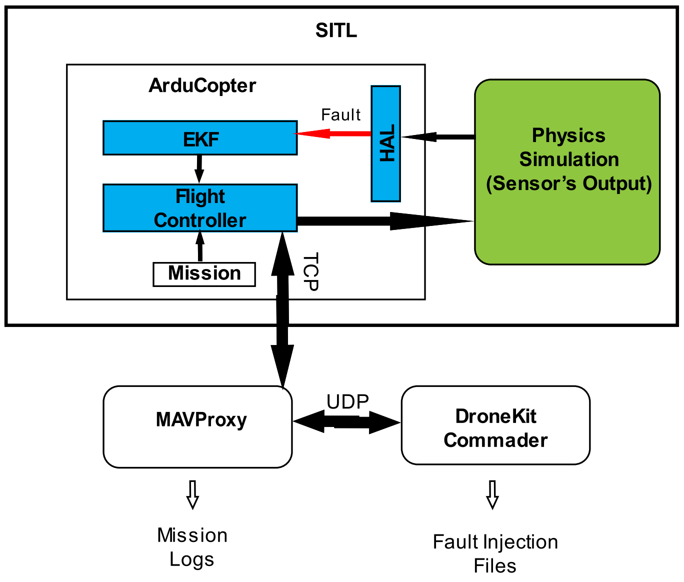

Figure 6.1: Diagram of the experimental setup

6.2 Flight Mission

The flight mission contains a basic trajectory, that is very common in most flights, and can be split into three distinct sectors:

1. **Takeoff**: From the quadcopter's initial position (Home coordinates), the device moves vertically to the first waypoint, at 1.5m of altitude.
2. **Horizontal Line**: After the takeoff, always at the same height, the quadcopter moves 30m forward in a straight line, and goes back to the first waypoints position. This section is traveled two times.
3. **Landing**: When the Horizontal Line's second lap ends, the quadcopter lands on the same position where it took off (Home coordinates).

The mission takes around 150 seconds to execute, since the moment the quadcopter is turned on until it lands and disarms the motors. In case the injected fault affects the system or the quadcopter as any other problem, the simulator keeps running until after 180 seconds of the beginning of the mission (an extra 30 seconds of the original time), before considering the mission concluded and shutting down.

Figure 6.2 shows the result of a simulated mission. The red line represent the reference trajectory, the blue line shows the simulated trajectory and the red asterisks are the waypoints (read from the mission file).

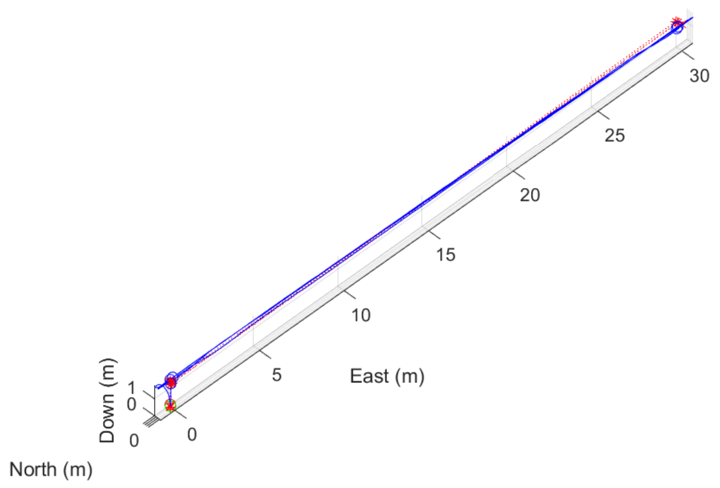

Figure 6.2: Simulated gold run mission example

6.3 Fault Model

In order to assess the effect of a *GPS Spoofing* attack on the SUA, it were injected faults in the GPS readings, between the parsing of the GPS message by the HAL, and the information being fed to the EKF. Since the GPS readings are altered, they cannot be relied on to analyze the flight output, so, instead of the GPS information, the position of the quadcopter on the simulator will be used in the analysis. Before a new fault-injection experiment, the SITL and the glsgcs are reseted, to prevent accumulation of fault effects.

The fault model is defined by three dimensions: trigger, duration and type. The **fault trigger** is time based and it is the same for all the experiments, which means that for each flight, after 45 seconds of the beginning of the mission (since the quadcopter is turned on and not the moment it takes off), the fault is injected. The duration goes from 1 second to 20 second, with a step of 1 second. This will allow to study the evolution of the impact throughout the time.

The type defines the how the fault will influence the sensor, as following:

- **Random Longitude**, tampers the longitude reading with a random value of a valid range of values (-180º to 180º).
- **Random Latitude**, tampers the latitude reading with a random value of a valid range of values (-90º to 90º).
- **Random Position**, tampers the position readings with random valid values on the three axis (latitude, longitude and altitude).
- **Landing**, tampers the altitude values, with higher values than the real one, trying to force an unplanned landing.
- **Message Delay**, does not tamper the message information, but delivers them with a certain delay.

- **Hijack with a second drone**, tampers the position readings with values from another drone trajectory, on the three axis (latitude, longitude and altitude).
- **Hijack with attacker position**, tampers the position readings with values from static position given by the attacker, on the three axis (latitude, longitude and altitude).

6.4 Result Analysis

In the experimental setup, it is given a mission file, with the commands and waypoints necessary to the flight, and a fault injection file, with all the information of the fault model, to each flight. The output of each experiment is a file, with all the information concerning the flight (e.g. sensorial readings, quadcopter position, EKF output). In order to analyze the impact of *GPS Spoofing* attacks, it was studied the the behavior of the SUA, by observing the three-dimensional coordinates of the position of the quadcopter. Since the GPS coordinates are being tampered by the fault injection mechanism, it will be used the SITL position information of the quadcopter.

Before injecting faults, it were made fault free flights, to determine the correct behavior of the quadcopter. Due to some indeterminism, default sensorial noise and the control algorithm characteristics, the quadcopter as some deviation from the mission file's predefined path, without external influences. If there is a deviation in the correct behavior of the quadcopter, it is expected that the experiments with injected faults have the same (in case the fault has no effect on the behavior of the quadcopter) or a bigger deviation (in case the fault is an impact on the behavior of the quadcopter). The runs without fault injection will serve as oracle, to decide the maximum deviation that is considered as correct behavior. This deviation is calculated by comparing the theoretical trajectory of the reference path and the real trajectory of each run. From the fault free runs, the maximum deviation calculated was 0.81 meters. This is considered the *normal margin*, which means the behavior of the quadcopter on that flight is considered **normal** or **fault free**.

In case a fault leads to the quadcopter moving further away than this normal margin, it means a failure has occured. To classify the erroneous behavior, it was defined a *safe margin* from the predefined path, where the quadcopter should not represent safety issues. In case the deviation is comprehended between the normal and the safe margin, it is considered it occurred a **minor failure**. If the deviation goes outside the safe margin, it means the quadcopter becames a threat to it's surroundings and a **major failure** has occurred. The safe margin is arbitrarily defined as being two times the normal margin. The classification of the results based on the deviation is shown on the Table 6.1.

Classification	**Description**
Normal	$deviation \leq normal\ margin$
Minor Failure	$normal\ margin < deviation \leq safe\ margin$
Major Failure	deviation > safe margin

Table 6.1: Results' Deviation Classification

.

The deviation is not the only category to be taken into account. Each attack is performed with intent to cause a specific effect on the quadcopter behavior. In the **hijacking attacks**, it is expected that the quadcopter does not finish the mission within the valid time and has a deviation bigger than the safe margin. This creates the conditions for an attacker to steal the quadcopter or use it for his own actions (e.g. crash into a building, spy) which is the purpose of this type of attacks. In case of the **Landing Attack**, the objective is to prevent the quadcopter of finishing his mission by making it hit the ground. Which is translated by having an altitude value of 0 or less. Finally the **denial of service attacks**, have the intend of preventing the quadcopter of finishing the mission within time or to fulfill the original path specification. If a test, has a deviation classification different than **normal**, the mission trajectory is analyzed in order to see if it matches the success criteria of the performed attack.

The log file of each experimented is compared to the mission file with a Matlab script responsible for:

- Read experiment data from log file.
- Read waypoint data from mission file.
- Compare distance between ideal and real trajectory.
- Plot the two trajectories.
- Save information about maximum deviation and coordinates of the real trajectory into a file.

This page is intentionally left blank.

Chapter 7

Result Analysis

The analysis made on this chapter is based on the experimental results present on the Appendix A.

7.1 Fault Duration Impact

For every type of fault injected, described on Chapter 6, the experiments started with an injection time duration of 1 second. The following experiments increased the duration of the fault by 1 second each time. This allowed to discover how long it takes to an attacker to affect the behavior of the quadcopter. For every different type of attack, the behavior of the quadcopter changed after 9 seconds of the fault being injected, except for message delay. The message delay is different from the other attacks, because it is a permanent fault. The time of injection on this case is not relative to the duration of the fault, but to the delay of the message. This means that the 1 second duration, is the delay of the messages through out the rest of the mission, instead of 1 second of fault injection as in the other attacks.

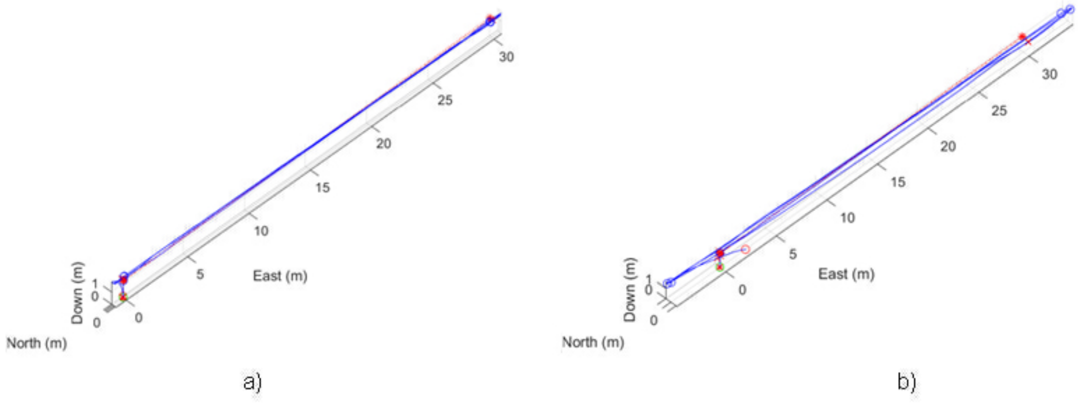

Figure 7.1: a) Random Position Attack b) Delay Message Attack

In Figure 7.1, it is compared the trajectory of the Random Position and Delay Message Attack runs, with 1 second of fault injection. This two trajectories show the fault tolerance mechanisms of the EKF algorithm of the quadcopter. In order to maintain the steadiest flight possible in a very unstable environment, the EKF tries to filter bad readings by comparing the results of two sets of equations:

- Time update equations, based on the previous states of the quadcopter.
- Measurement update equations, based on the sensorial information that is used as input for the EKF.

With this method, the previous states prevent the system from using bad position values, based on noise or tampered readings. But when the window of previous states is persistently occupied by bad values, the reading equations and the time equations will have similar values, and the reading equations output will be used by the system to navigate. Since a 1 second window is to small to affect the update time equations, the Random Position Attack on Figure 7.1 is identical to a gold run (Figure 6.2) and the Delay Message Attack, that persists until the end of the mission, since it is triggered, has already a visible deviation of 4.77 meters, being outside of the *safe margin*.

Since the duration of the fault affects the attack success, an important breakthrough is to find how long the system tolerates the attacks' tampered data. From 1 to 9 seconds, with exception of the Message Delay and the Random Position Attacks, the system was unaffected by the faults injected, and even the Random Position Attacks only has a deviation of 0.22 meters, being inside the safe zone, and real close of the normal behavior. The Message Delay is a permanent duration attack so it was expected it's deviations were different from the other attacks. On Figure 7.2, it is evident the difference between the unaffected trajectory of the run with a fault duration of 9 seconds, and the 10 second duration fault, where the systems' time equations are no longer able of filter the tampered readings, and the systems steps out of the original trajectory, trying to compensate the injected error.

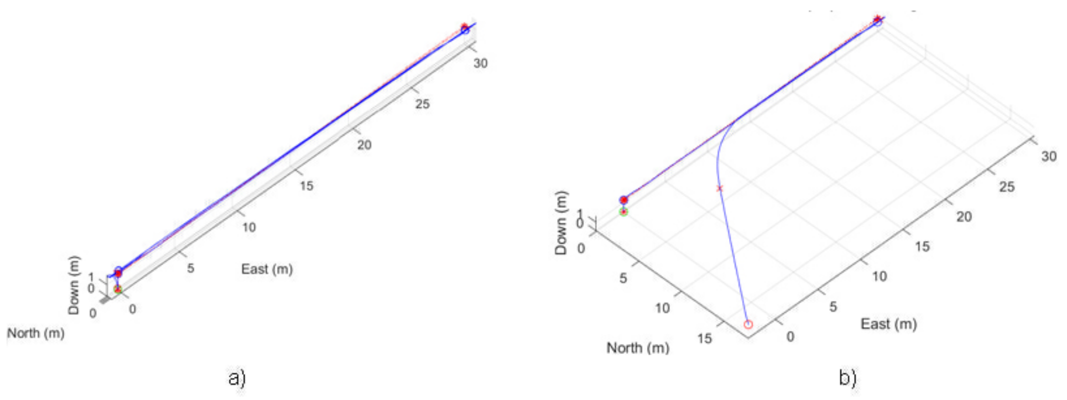

Figure 7.2: Hijacking with attacker position: a) 9 second fault b) 10 second fault

It is possible to conclude that the duration of an attack influences it's success but it is a direct relation between the duration and the deviation values, as it is show, for each type of attack, in Figure 7.3, 7.4 and 7.5.

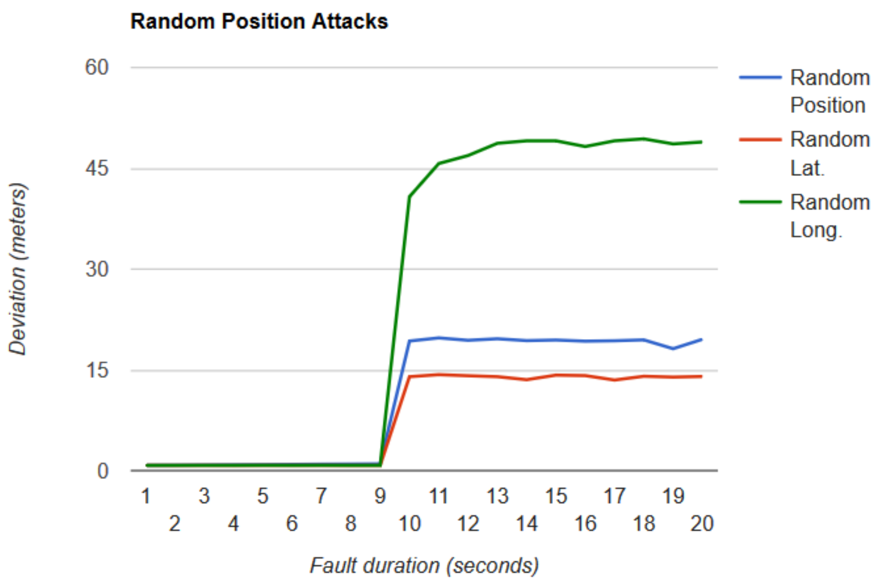

Figure 7.3: Random Position Attacks Deviation

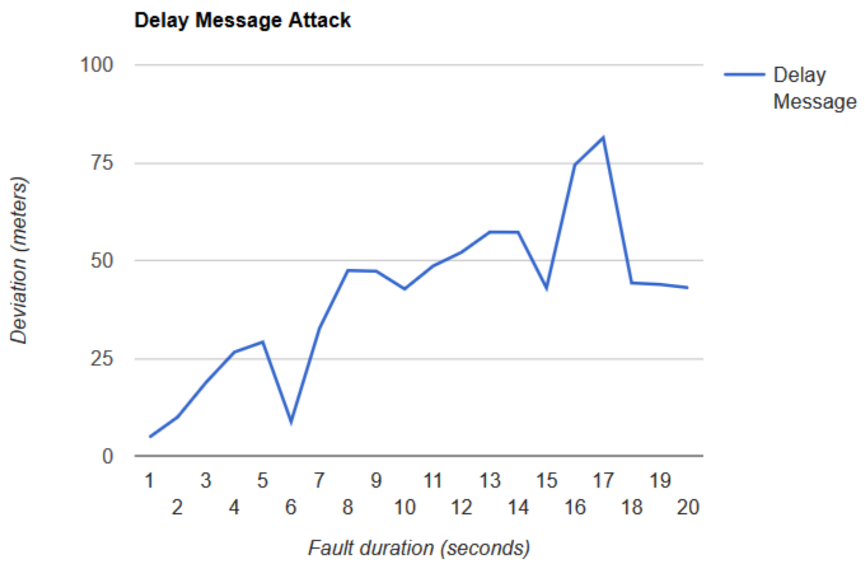

Figure 7.4: Delay Message Attack Deviation

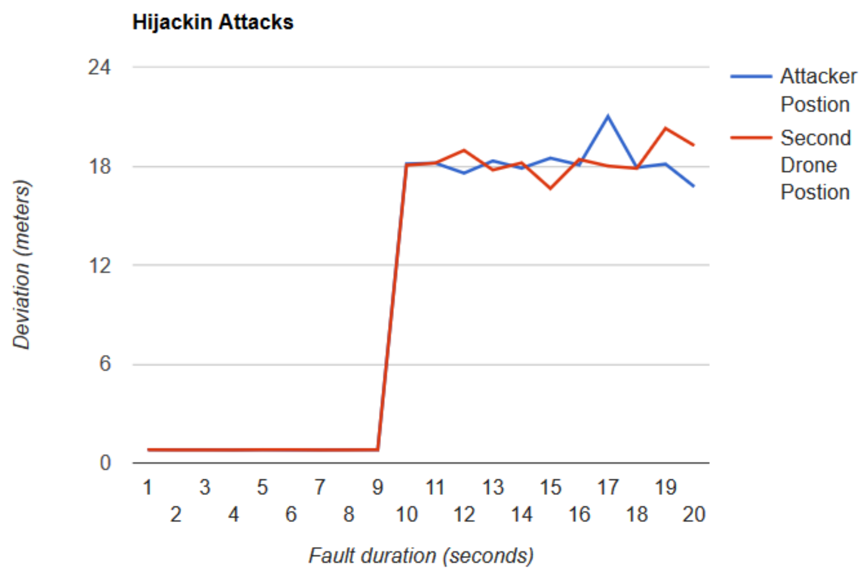

Figure 7.5: Hijacking Attacks Deviation

7.2 Attack Type Impact

Besides the duration of the default, there is other parameter that may affect the behavior of the quadcopter, which is the attack type. Different attacks have different objectives. Hijacking with the attacker position or with a second drone intend to take the quadcopter away from his legitimate user and use it for malicious purposes or to steal it. While other attacks like Random Position, Message Delay or the Landing Attack, try to prevent the correct use of the system by it's rightful user (Denial of Service).

From the different attacks there was only one that did not have a single successful attack, the Landing Attack. In this attack the malicious user, tampers the altitude information of the GPS, with higher values than the measured ones, so the quadcopter to compensate this difference starts to lower it's throttle, going down and hitting the ground. But in all the tests performed, the quadcopter's position never moved away from the mission path more than the normal margin of 0.81 meters. This is due to a characteristic of the flight controller. Since GPS measures have already some error, without any external factor, the system relies in the barometer to make the calculations of the quadcopter's altitude, based on the barometric pressure, and using the temperature to make corrections. Using other sensor, instead the GPS reading, to make the calculations of this position parameter, prevented the success of the Landing Attack.

The Random Position Attack changes both the latitude and the longitude readings in the GPS message, but were also performed tests (Random Latitude and Random Longitude) to study the effect of each of the parameters on the flight. For all of these attacks, since the injected values are random, the runs were repeated 10 times, and the final result are the arithmetic mean of the maximum deviation of all runs. From the results of the Random Latitude and Random Longitude Attacks, it seems that the Longitude has a bigger impact than the Latitude, but although the deviation values from the Random Position, which uses the both attributes tampered, is slightly higher than the Random Latitude Attack, they are considerably smaller than the Random Longitude Attack values. The maximum deviations for this attacks were:

- 14.23 meters, for the Random Latitude Attack.
- 19.77 meters, for the Random Position Attack.
- 49.38 meters, for the Random Longitude Attack.

This results are justified by the trajectory characteristics. For all the duration of the flight is made over the latitude axis, maintain the longitude values almost constant. So the latitude faults injected make the quadcopter to move in a trajectory that overlaps the original movement, while the longitude faults lead the quadcopter to move away from the defined path in perpendicular direction. Since the Random Position Attack mixes the two attributes faults, it's deviation values are higher than the Random Latitude and lower than the Random Longitude, since the direction the quadcopter takes is comprehend between the trajectories of the other two attacks. In the Figure 7.6 it is shown how the Random Latitude and Random Longitude affect the system behavior.

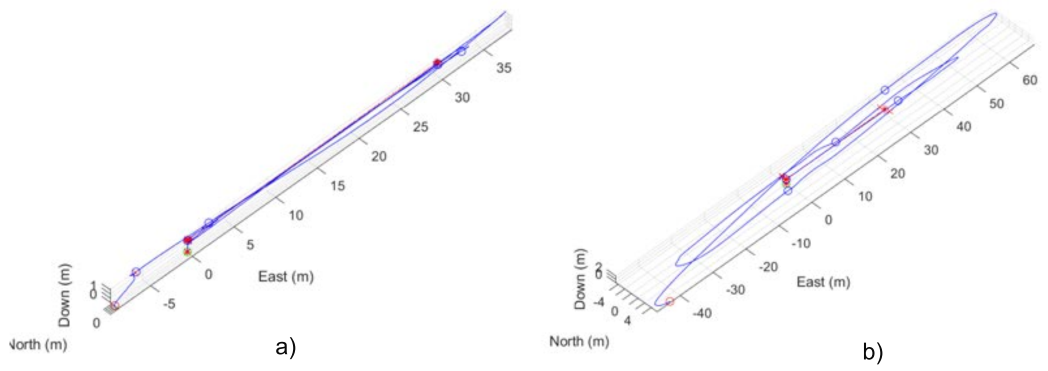

Figure 7.6: Maximum Deviation: a) Random Latitude b) Random Longitude

The Delay Message Attack had the highest deviation values from all the attacks, but it is also the only one with permanent duration after the trigger. Another unique characteristic of this attack is that the tampered values used on the injection are real values from the original trajectory, while the other attacks use valid values, but not necessarily from the original path. This can benefit the attacker and lead to bigger deviations, since the EKF algorithm has a harder time to detect an erroneous measure, and including bad states for it's equation time calculations. This also makes the success of the attack a bit unpredictable, since is the delay duration and the flight characteristics that will affect the EKF decision to use the measures or the time equations. That is why there are some outliers results like 6 second delay having a deviation of 8.8 meters while the 5 second delay mission had a deviation of 29.14 meters, as seen of the Figure 7.7, or the 17 delay second mission having the maximum deviation value of 81.37 meters, and from that the deviation drops to around 45 meters. The Figure 7.7 show the unexpected difference between the 5 second message delay and 6 second message delay runs.

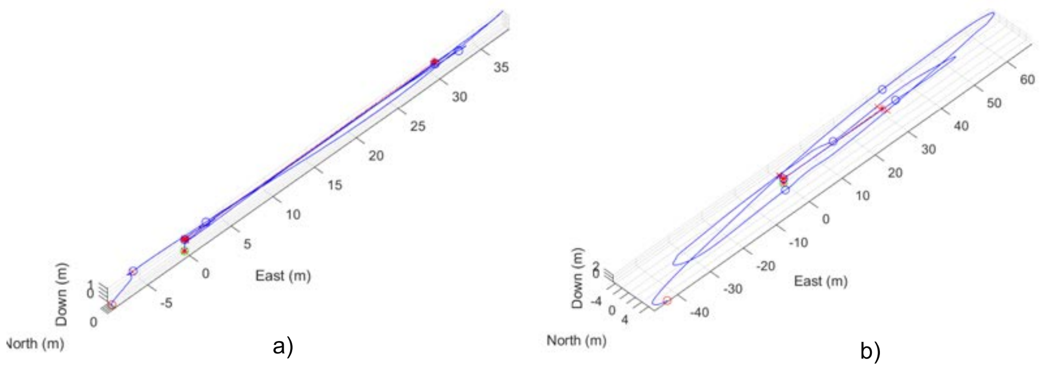

Figure 7.7: Message Delay Attack: a) 6 second delay b) 5 second delay

The hijacking attempts are very similar to each other, it is given a known false position to the quadcopter, in order to lead it's flight to a position chosen by the attacker, but on one of the attacks is used a fixed position, in this case we assume it is the malicious user position , and on the other attack is used a position of another quadcopter, that is flying at the same time, but on another trajectory, in this case is a parallel trajectory, 20 meters away from the original quadcopter. Both attacks were successful, and had similar deviation results, being the maximum deviation for the Hijack using a Second Drone of 20.27 meters and for the Hijack using the Attacker Position of 20.99 meters. In the hijacking attempts the type of attack and the tampered positions influence more the trajectory of the subject quadcopter than the deviation values. In Figure 7.8, is shown the two trajectories with maximum deviation of each attack.

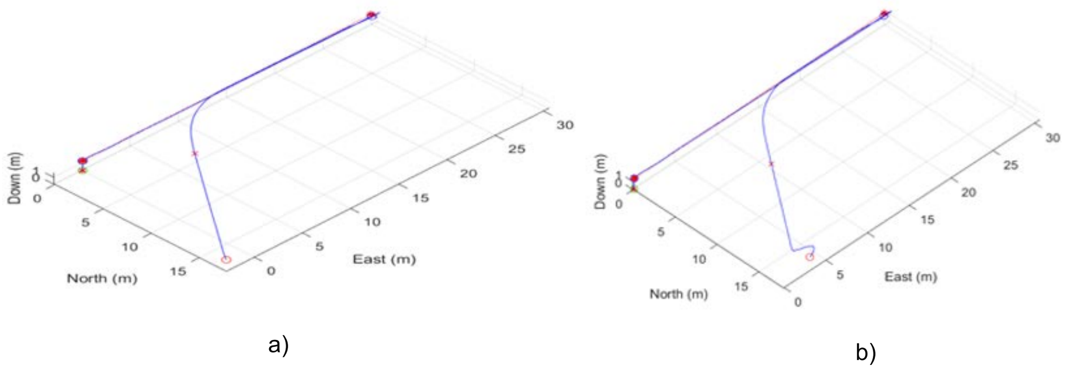

Figure 7.8: Hijacking Attack: a) Attacker Position b) Second Drone

7.3 Defense Mechanisms

One defense mechanism is already present in the system, shown by the Landing Altitude. Using the barometer to make the altitude calculations prevents attacks to the altitude value of the GPS, that would force a landing or a crash. This could be applied to other sensors, which would have a bigger weight in the position calculations, but having electromagnetic fields generated by the motors and the electronic components of the flight board, induce error on these sensors (e.g. magnetometer), being not as reliable as the barometer.

Another possible defense mechanism could use the concept of safe margin. There would be set a radius, that would be measured from the mission path to quadcopter's position. If the quadcopter would go out of the radius value, using the barometer information, the quadcopter could land and prevent a possible attack of being successful. The problems with this approach are:

- The radius value. If it is too small, a wind deviation or other environmental factor could cause a landing, stopping the mission without a valid reason. If it is too big, it gives a margin to the attacker to hijack the quadcopter.
- The attacker could use the radius value to force a landing in order to steal the quadcopter or cause damage.
- Compute de distance of the quadcopter to the original path, and verify the radius would increase the complexity of real time operations, and could cause delays on vital information to the system.

This page is intentionally left blank.

Chapter 8

Conclusion

Risk-driven security testing is very useful on complex systems, like flight controllers, with many different states, since it prioritizes the threats, knowing which ones are more critical and should be tested and improved. To assess our system, it was used the STRIDE model to identify the threats and the DREAD scheme to rate them. Based on the threat classification, the GPS Spoofing attack was chosen to perform a Fault Injection test. A Software In The Loop system, using the same flight controller software as a real quadcopter, to perform the mission flights, with or without tampered data related to the attacks. The several denial of service attacks and the hijacking attacks proved to be successful most of the times, which indicates that most of the commercial quadcopters used by the public are vulnerable to this kind of attacks and cannot guarantee the safety properties. Although some defense mechanism already exists, like the Extend Kalman Filter time equations, or the use of the barometer for the altitude calculations, new mechanisms are needed in order to prevent GPS Spoofing attacks to be successful.

Since the results from the 140 faults injected are from a simulation system, the results are as good as the model of the quadcopter, thus is intended in the future work to validate this results with the real quadcopter, with identical characteristics and to study new defense mechanisms.

This page is intentionally left blank.

References

[1] Mavlink developer guide. In *http://qgroundcontrol.org/mavlink/start*, (2017).

[2] Anurag Agarwal. Vast methodology: Visual, agile, and simple threat modeling. In *Various Interviews. Transformational Opportunities*, Prescott Valley, (2016).

[3] C. Anderson. Ardupilot. In *http://ardupilot.org/copter/*, July, (2010).

[4] Avuzuebus Avizienis, Jean-Claude Laprie, Brian Randell, and Carl Landwehr. Basic concepts and taxonomy of dependable and secure computing. In *IEEE Transactions ON Dependable AND Secure Computing*, VOL. 1, NO. 1,, (2004).

[5] BBC. Prisons drone-delivery drugs plot: Eleven charged. In *https://www.bbc.com/news/uk-39616399*, (2017).

[6] A. F. Benet. A risk driven approach to testing medical device software. In *Dale C., Anderson T. (eds) Advances in Systems Safety*, (2011).

[7] Zachary Birnbaum, Andrey Dolgikh, Victor Skormin, Edward O'Brien, Daniel Muller, and Christina Stracquodaine. Unmanned aerial vehicle security using behavioral profiling. In *International Conference on Unmanned Aircraft Systems (ICUAS)*, (2015).

[8] Hugh Boyes. Cyber security attributes for critical infrastructure systems. In *http://www.cybersecurity-review.com/articles/cyber-security-attributes-for-critical-infrastructure-systems*, (2018).

[9] N. Butcherand, A. Stewart, and S. Biaz. Securing the mavlink communication protocol for unmanned aircraft systems. Appalachian State University, Auburn University, (2013).

[10] Marcel Böhme, Bruno C. d. S. Oliveira, and Abhik Roychoudhury. Regression tests to expose change interaction errors. In *ESEC/FSE'13*, (2013).

[11] Edsger W. Dijkstra. Notes on structured programming. In *http://www.cs.utexas.edu/users/EWD/ewd02xx/EWD249.PDF*, (1970).

[12] N. Dunn and J. Murray. Software security: Building security in. In *IEEE Security & Privacy*, (2006).

[13] Emlid. Navio2. In *https://emlid.com/introducing-navio2/*, (2015).

[14] Gencer Erdogan and Ketil Stølen. Risk-driven security testing versus test-driven security risk analysis. In *First Doctoral Symposium on Engineering Secure Software und Systems*, (2012).

[15] Clifton A. Ericson. Software safety in a nutshell. In *http://www.dcs.gla.ac.uk/˜johnson/teaching/safety/reports/Clif_Ericson1.htm.*

[16] Electronic Frontier Foundation. Surveillance drones. In *https://www.eff.org/issues/surveillance-drones.*

[17] Lei Gong and Shuguang Zhang. Safety requirements analysis for control law development of uav flight control systems. In *In: Robotics Research*, 2nd International Symposium on Aircraft Airworthiness, (2011).

[18] CVSS Special Interest Group. Common vulnerability scoring system v3.0: Specification document. In *https://www.first.org/cvss/specification-document*, The US Department of Homeland Security, (2015).

[19] Jürgen Großmann and Fredrik Seehusen. Combining security risk assessment and security testing based on standards. In *Risk Assessment and Risk-Driven Testing: Third International Workshop*, Berlin, (2015).

[20] S. Grzonka and W. Grisetti, G.and Burgard. A fully autonomous indoor quadrotor. In *IEEE Transactions on Robotics 28(1)*, 90-100, (2012).

[21] W. Harrington. Learning raspbian. In *Packt Publishing Ltd*, (2015).

[22] Kim Hartmann and Christoph Steup. The vulnerability of uavs to cyber attacks - an approach to the risk assessment. In *5th International Conference on Cyber Conflict*, (2013).

[23] Ahmad Y. Javaid, Weiqing Sun, Vijay K. Devabhaktuni, and Mansoor Alam. Cyber security threat analysis and modeling of an unmanned aerial vehicle system. In *IEEE Conference on Technologies for Homeland Security (HST)*, (2012).

[24] Loren Kohnfelder and Praerit Garg. Threats to our products. In *Microsoft*, (2016).

[25] V. Kumar and N. Michael. Opportunities and challenges with autonomous micro aerial vehicles. In *In: Robotics Research*, pp. 41-58, Springer (2017).

[26] Tom Mendelsohn. Sweden's highest court bans drones with cameras. In *https://arstechnica.com/tech-policy/2016/10/camera-spy-drones-banned-sweden-highest-court/*, (2016).

[27] D. Mendes, J. Nunes, S. Patrão, N. Ivaki, P. Amaro, and J. Cunha. Assessing the robustness of a quadcopter's flight controller to sensor failures. In *INForum*, Department of Informatics Engineering, University of Coimbra, Portugal, (2017).

[28] A. Muñoz, R. Harjani, A. Maña, and R. Díaz. Dynamic security monitoring and accounting for virtualized environments. In *FTRA International Conference on Secure and Trust Computing, Data Management, and Application*, (2011).

[29] S. Northcutt, J. Shenk, D. Shackleford, T. Rosenberg, R. Siles, and S. Mancini. Automated defect prevention: Best practices in software management. In *Wiley-IEEE Computer Society Press*, Kolawa, A. and Huizinga, D., (2007).

[30] S. Northcutt, J. Shenk, D. Shackleford, T. Rosenberg, R. Siles, and S. Mancini. Penetration testing: Assessing your overall security before attackers do. In *SANS Institute InfoSec Reading Room*, Retrieved 16 January 2014.

[31] OWASP. Threat risk modeling. In *https://www.owasp.org/index.php/Threat_Risk_Modeling*, (2018).

[32] Pierluigi Paganini. Hacking drones overview of the main threats. In *General Security, Infosec Institue*, (2013).

[33] M. Palanivel and K. Selvadurai. Risk-driven security testing using risk analysis with threat modeling approach. In *https://doi.org/10.1186/2193-1801-3-754*, (2014).

[34] Maragathavalli Palanivel and Kanmani Selvadurai. Risk-driven security testing using risk analysis with threat modeling approach. In *SpringerPlus*, (2014).

[35] 3D Robotics. Dronekit-python api. In *http://python.dronekit.io/about/index.html*, (2015-2016).

[36] F. Samland, J. Fruth, M. Hildebrandt, T. Hoppe, and J. Dittmann. Ar.drone: Security threat analysis and exemplary attack to track persons. In *Proceedings of SPIE - The International Society for Optical Engineering*, (2012).

[37] Seong-Hun Seo, Byung-Hyun Lee, Sung-Hyuck Im, and Gyu-In Jee. Effect of spoofing on unmanned aerial vehicle using counterfeited gps signal. In *Journal of Positioning, Navigation, and Timing*, pages 57–65, 06 2015.

[38] Daniel J. Solove. The digital person technology and privacy in the information age. In *NyU Press*, (2004).

[39] T. UcedaVelez and Marco M. Morana. Risk centric threat modeling process for attack simulation and threat analysis. In *John Wiley & Sons*, Hobekin, (2015).

[40] E. Upton and G. Halfacree. Raspberry pi user guide. In *John Wiley & Sons*, (2014).

[41] Junia Valente and Alvaro A. Cardenas. Understanding security threats in consumer drones through the lens of the discovery quadcopter family. In *IoT S&P'17*, Dallas, TX, USA, (2017).

[42] Lisa Vas. Sweden bans cameras on drones, deeming it illegal surveillance. In *https://nakedsecurity.sophos.com/2016/10/27/sweden-bans-cameras-ondrones-deeming-it-illegal-surveillance/amp*, (2016).

[43] Philipp Zech, Michael Felderer, and Ruth Breu. Towards risk–driven security testing of service centric systems. In *Quality Software (QSIC), 2012 12th International Conference on*, (2012).

Appendices

Appendix A

The following tables show the results of all the simulated runs.

Attack	**Duration** *(seconds)*	**Maximum Deviation** *(meters)*	**Classification**	**Attack Success**
Random Longitude	1	0.8	Normal	Failure
Random Longitude	2	0.8	Normal	Failure
Random Longitude	3	0.8	Normal	Failure
Random Longitude	4	0.8	Normal	Failure
Random Longitude	5	0.81	Normal	Failure
Random Longitude	6	0.8	Normal	Failure
Random Longitude	7	0.81	Normal	Failure
Random Longitude	8	0.8	Normal	Failure
Random Longitude	9	0.8	Normal	Failure
Random Longitude	10	40.78	Major Failure	Success
Random Longitude	11	45.71	Major Failure	Success
Random Longitude	12	46.9	Major Failure	Success
Random Longitude	13	48.72	Major Failure	Success
Random Longitude	14	49.09	Major Failure	Success
Random Longitude	15	49.09	Major Failure	Success
Random Longitude	16	48.24	Major Failure	Success
Random Longitude	17	49.09	Major Failure	Success
Random Longitude	18	49.38	Major Failure	Success
Random Longitude	19	48.63	Major Failure	Success
Random Longitude	20	48.91	Major Failure	Success
Random Latitude	1	0.81	Normal	Failure
Random Latitude	2	0.8	Normal	Failure
Random Latitude	3	0.81	Normal	Failure
Random Latitude	4	0.8	Normal	Failure
Random Latitude	5	0.81	Normal	Failure
Random Latitude	6	0.81	Normal	Failure
Random Latitude	7	0.81	Normal	Failure
Random Latitude	8	0.8	Normal	Failure
Random Latitude	9	0.8	Normal	Failure
Random Latitude	10	14.0	Major Failure	Success
Random Latitude	11	14.3	Major Failure	Success
Random Latitude	12	14.14	Major Failure	Success
Random Latitude	13	14	Major Failure	Success
Random Latitude	14	13.55	Major Failure	Success
Random Latitude	15	14.23	Major Failure	Success
Random Latitude	16	14.16	Major Failure	Success
Random Latitude	17	13.5	Major Failure	Success
Random Latitude	18	14.06	Major Failure	Success
Random Latitude	19	13.94	Major Failure	Success
Random Latitude	20	14.03	Major Failure	Success

Table 1: Fault injection test results 1/5

Attack	**Duration** *(seconds)*	**Maximum Deviation** *(meters)*	**Classification**	**Attack Success**
Random Position	1	0.81	Normal	Failure
Random Position	2	0.81	Normal	Failure
Random Position	3	0.86	Minor Failure	Failure
Random Position	4	0.89	Minor Failure	Failure
Random Position	5	0.91	Minor Failure	Failure
Random Position	6	0.95	Minor Failure	Failure
Random Position	7	0.98	Minor Failure	Failure
Random Position	8	1	Minor Failure	Failure
Random Position	9	1.03	Minor Failure	Failure
Random Position	10	19.3	Major Failure	Success
Random Position	11	19.77	Major Failure	Success
Random Position	12	19.41	Major Failure	Success
Random Position	13	19.64	Major Failure	Success
Random Position	14	19.36	Major Failure	Success
Random Position	15	19.45	Major Failure	Success
Random Position	16	19.28	Major Failure	Success
Random Position	17	19.34	Major Failure	Success
Random Position	18	19.47	Major Failure	Success
Random Position	19	18.16	Major Failure	Success
Random Position	20	19.56	Major Failure	Success
Landing	1	0.8	Normal	Failure
Landing	2	0.81	Normal	Failure
Landing	3	0.81	Normal	Failure
Landing	4	0.8	Normal	Failure
Landing	5	0.81	Normal	Failure
Landing	6	0.8	Normal	Failure
Landing	7	0.81	Normal	Failure
Landing	8	0.8	Normal	Failure
Landing	9	0.81	Normal	Failure
Landing	10	0.81	Normal	Failure
Landing	11	0.81	Normal	Failure
Landing	12	0.8	Normal	Failure
Landing	13	0.81	Normal	Failure
Landing	14	0.81	Normal	Failure
Landing	15	0.81	Normal	Failure
Landing	16	0.8	Normal	Failure
Landing	17	0.81	Normal	Failure
Landing	18	0.8	Normal	Failure
Landing	19	0.81	Normal	Failure
Landing	20	0.8	Normal	Failure

Table 2: Fault injection test results 2/5

Attack	**Duration** ***(seconds)***	**Maximum Deviation** ***(meters)***	**Classification**	**Attack Success**
Hijack (attacker position)	1	0.81	Normal	Failure
Hijack (attacker position)	2	0.81	Normal	Failure
Hijack (attacker position)	3	0.81	Normal	Failure
Hijack (attacker position)	4	0.8	Normal	Failure
Hijack (attacker position)	5	0.81	Normal	Failure
Hijack (attacker position)	6	0.8	Normal	Failure
Hijack (attacker position)	7	0.8	Normal	Failure
Hijack (attacker position)	8	0.81	Normal	Failure
Hijack (attacker position)	9	0.81	Normal	Failure
Hijack (attacker position)	10	18.12	Major Failure	Success
Hijack (attacker position)	11	18.17	Major Failure	Success
Hijack (attacker position)	12	17.56	Major Failure	Success
Hijack (attacker position)	13	18.3	Major Failure	Success
Hijack (attacker position)	14	17.86	Major Failure	Success
Hijack (attacker position)	15	18.47	Major Failure	Success
Hijack (attacker position)	16	18.06	Major Failure	Success
Hijack (attacker position)	17	20.99	Major Failure	Success
Hijack (attacker position)	18	17.9	Major Failure	Success
Hijack (attacker position)	19	18.11	Major Failure	Success
Hijack (attacker position)	20	16.74	Major Failure	Success

Table 3: Fault injection test results 3/5

Attack	**Duration** *(seconds)*	**Maximum Deviation** *(meters)*	**Classification**	**Attack Success**
Hijack (second drone)	1	0.81	Normal	Failure
Hijack (second drone)	2	0.8	Normal	Failure
Hijack (second drone)	3	0.8	Normal	Failure
Hijack (second drone)	4	0.8	Normal	Failure
Hijack (second drone)	5	0.81	Normal	Failure
Hijack (second drone)	6	0.81	Normal	Failure
Hijack (second drone)	7	0.8	Normal	Failure
Hijack (second drone)	8	0.8	Normal	Failure
Hijack (second drone)	9	0.81	Normal	Failure
Hijack (second drone)	10	18.04	Major Failure	Success
Hijack (second drone)	11	18.17	Major Failure	Success
Hijack (second drone)	12	18.94	Major Failure	Success
Hijack (second drone)	13	17.75	Major Failure	Success
Hijack (second drone)	14	18.18	Major Failure	Success
Hijack (second drone)	15	16.63	Major Failure	Success
Hijack (second drone)	16	18.39	Major Failure	Success
Hijack (second drone)	17	17.99	Major Failure	Success
Hijack (second drone)	18	17.85	Major Failure	Success
Hijack (second drone)	19	20.27	Major Failure	Success
Hijack (second drone)	20	19.23	Major Failure	Success

Table 4: Fault injection test results 4/5

Attack	**Duration** ***(seconds)***	**Maximum Deviation** ***(meters)***	**Classification**	**Attack Success**
Message Delay	1	4.77	Major Failure	Success
Message Delay	2	9.96	Major Failure	Success
Message Delay	3	18.81	Major Failure	Success
Message Delay	4	26.56	Major Failure	Success
Message Delay	5	29.14	Major Failure	Success
Message Delay	6	8.8	Major Failure	Success
Message Delay	7	32.65	Major Failure	Success
Message Delay	8	47.43	Major Failure	Success
Message Delay	9	47.26	Major Failure	Success
Message Delay	10	42.71	Major Failure	Success
Message Delay	11	48.57	Major Failure	Success
Message Delay	12	52.08	Major Failure	Success
Message Delay	13	57.25	Major Failure	Success
Message Delay	14	57.21	Major Failure	Success
Message Delay	15	42.97	Major Failure	Success
Message Delay	16	74.44	Major Failure	Success
Message Delay	17	81.37	Major Failure	Success
Message Delay	18	44.25	Major Failure	Success
Message Delay	19	43.87	Major Failure	Success
Message Delay	20	43.04	Major Failure	Success

Table 5: Fault injection test results 5/5