UNIVERSITÀ DEGLI STUDI DI MILANO

DIPARTIMENTO DI INFORMATICA

Corso di Laurea in
Sicurezza dei Sistemi e delle Reti Informatiche

Security aspects of avionics protocols

RELATORE
Prof. Valerio Bellandi

TESI DI LAUREA DI
Giulio Ginesi
Matr. 872795

CORRELATORE
Prof. Ernesto Damiani

Anno Accademico 2017/2018

Ai miei genitori

Ringraziamenti

Contents

Acronyms

ACARS Aircraft Communications Addressing and Reporting System.

ACAS Airborne Collision Avoidance System.

ADS-B Automated Dependent Surveillance - Broadcast.

ADS-C Automatic Dependent Surveillance - Contract.

AFL American Fuzzy Lop.

AOC Air and Space Operations Center.

ATC Air Traffic Control.

ATM Air Traffic Management.

ATS Air Traffic Service.

CRC Cyclic Redundancy Check.

DF Downlink Format.

EHS Enhanced Surveillance.

ELS Elementary Surveillance.

ENAC Ente Nazionale Aviazione Civile.

ENAV Ente Nazionale Assistenza Volo.

EUROCAE European Organization for Civil Aviation Equipment.

FAA Federal Aviation Administration.

FIS-B Flight Information Services - Broadcast.

GA General Aviation.

HF High Frequency.

IC Interrogator Code.

ICAO International Civil Aviation Organization.

IFF Identification Friend or Foe.

IFR Instrument Flight Rules.

IoT Internet of Things.

MTOW Maximum Takeoff Weight.

NOTAM Notice To AirMen.

RA Resolution Advisory.

RTCA Radio Technical Commission for Aeronautics.

SAR Search and Rescue.

SESAR Single European Sky ATM Research.

SPI Special Identification Pulse.

SSR Second Surveillance Radar.

TIS-B Traffic Information Services - Broadcast.

UAT Universal Access Transceiver.

UAV Unmanned Aerial Vehicle.

VHF Very High Frequency.

Chapter 1

Introduction

T0D0 1: Short introduction

The rest of the thesis is organized as follows:
In *Chapter 2: Scenario* I analyze the current state of the art of the avionics protocols that will be tested during the research as well as the previous versions.
Chapter 3: Testing and Validation
Chapter 4: Analysis
Chapter 5: Results
Chapter 6: Conclusions and Future Work

Chapter 2

Scenario

The technological explosion of the last years influenced also the aviation sector. Better planning, more efficient aircrafts and many other management improvements brought the price of airline tickets down making traveling by plane accessible to anyone. More recently the drone market rapidly grew to the point where every professional and amateur video maker must have at least one drone to capture great aerial shots that in the past could only be done by using helicopters. Moreover companies like *Amazon* and *DHL* successfully tested a parcel delivery service operated by drones [1] while other companies like *Facebook* and *Google* started testing an affordable and reliable method to bring internet connection to remote areas of the planet [2]. Furthermore, companies like PrecisionHawk [3] developed accurate sensors to be used with different drones that allow the user to perform many different measurements. Unmanned Aerial Vehicles (**UAV**) are spreading also in the 4.0 agriculture, Search and Rescue (**SAR**) and general emergency fields[4]. Nowadays Military operations are more and more drone-centered, therefore armed forces, from infantry to flying squadrons, are equipped with different kinds of **UAV**. All this sudden progress has led to overcrowded skies in which the innovation in the Air Traffic Management (**ATM**) field must be continuously updated. New systems have been built with a strong push of implementing IoT paradigm in respect with the **ATM**, **UAV** and avionics systems.

Airplanes can no longer be seen as the main users of the sky. A wider spectrum of flying objects should be considered given the different nature of their functions. Aircrafts are divided into different categories depending on the maximum altitude, their weight and other parameters. Commercial drones, such as *Amazon* or cinematography ones are now developing capabilities similar to small aircrafts in terms of performance. Similarly *Facebook* Aquila drones have the same wingspan as a Boeing 737 and they can fly at a considerably high altitude[5]. Also Project Loon internet balloons (*Google*) travel at high altitude and through busy airspace. In addition, there are many other objects like gas balloons, helicopters and gliders operating in specific areas and airspace class that must be tracked, managed and monitored efficiently to ensure high safety standards.

The development of a new technology brings a lot of challenges, some of them in the cybersecurity realm. In particular, Costin and Francillon [6] demonstrated in 2012 that it is easy and cheap to perform various operational attacks on ADS-

B protocols in particular, and **ATM** and **ATC** systems in general. Subsequently, various weaknesses were demonstrated in various Air Traffic Control (ATC) protocols [7, 8] and avionics sub-systems. More recently, in November 2017 the news broke that back in September 2016 a team from Department of Homeland Security (DHS) was able to hack a Boeing 757 parked at the airport. The hack was described as *"remote, non-cooperative, penetration"* [9]. Though the details were scarce because of the classified label, it was acknowledged that the team, consisting of industry experts and academics, accomplished the hack by accessing the aircraft communication systems through radio frequency communications.

Aiming at more security and in order to overcome some limitations of the old generation protocols like the absence of radar coverage in certain areas of the planet (e.g. over the ocean), a modernization of the radar and communication system was launched starting from the 80s up to the present days with the development of two generations of protocols. Standards and guidance documents for aeronautics are defined by different cooperative organizations, the major of which are Radio Technical Commission for Aeronautics (**RTCA**) in the United States, European Organization for Civil Aviation Equipment (**EUROCAE**) in Europe and International Civil Aviation Organization (**ICAO**) which is a UN organization. The above mentioned organizations are non governative therefore they produce standards, guidelines or rules that will have to be adopted by the Federal Aviation Administration (**FAA**) in the united states, by the **EUROCONTROL**[1] and the single national aviation authority of the various European countries (*ENAC/ENAV* in Italy). This fragmentation of the establishment, the absence of a globally shared guidance on the standardization of the aviation sector (which is partially an ICAO task.[2]) and the development of different standards to accomplish the same task have created an unclear situation about standards and regulations. NextGen has shown problems in the definition of a globally accepted family of protocols as it can be seen in the relevant section.

Aircrafts can be tracked either by a **cooperative** way and a **non-cooperative** one:

- The classic radar (or Primary Radar), based on electromagnetic waves is able to give information only on the position of an object in the sky. Such system gives a real-time image of the portion of sky it is observing, including aircrafts, birds, clouds, and any other object in its visual range making it a basic and fairly unreliable system. Since this method requires no interaction with the aircraft that is being tracked, it is called **non-cooperative** system.

- Second Surveillance Radar (**SSR**) is an evolution of the above system. In addition to the spatial position of the aircraft, it gives further information depending on its mode of operation. This system requires an active cooperation from the aircraft which must reply to the interrogations received making it a **cooperative** system.

[1] The organization that provides unique centralized platform for civil and military aviation coordination in Europe.

[2] Art 1. The contracting States recognize that every State has complete and exclusive sovereignty over the airspace above its territory.[10]

The cooperative systems allow to have a bidirectional data flow between aircraft and ground as well as aircraft and aircraft. This system stems from the military Identification Friend or Foe (**IFF**) one which was developed during the Second World War. The actual civil system uses a 4 digit code called "Transponder Code" or **"Squawk"** to identify the aircraft. This communicates via a **transponder** which handles the incoming interrogations and the responses. In Figure 2.1 is an overview of all the protocols and their division between the old generation and the new one which is still in deployment.

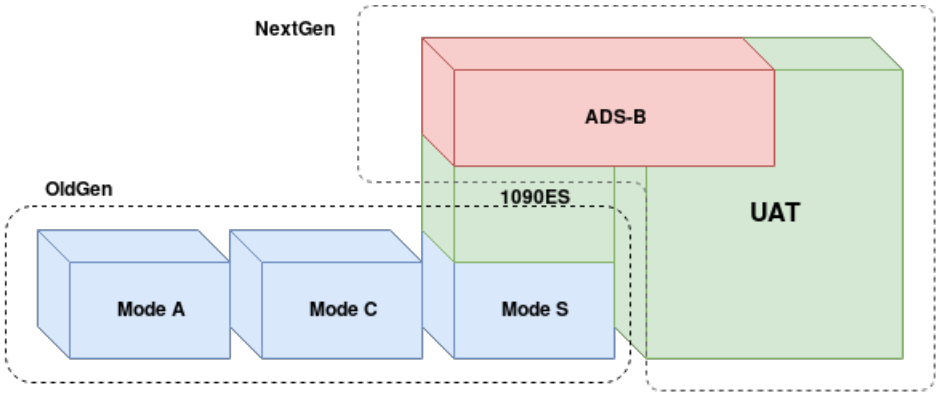

Figure 2.1: All generation of avionics protocols

Having a continuous data flow between the aircraft and the ground is beneficial not only for the ATM/ATC but also for the airline. As a matter of fact:

- Flight controllers can obtain much more information about a single aircraft, his intentions and the status of the flight. Moreover satellite based ADS-B allow to trace planes in areas where no radar coverage is available.
- The airline can keep track of its fleet in real time thus allowing a better planning of the turnover and, if needed, quick deployment of a substitute airplane.
- The maintenance branch of the airline can track in real time any anomalies reported by the instruments and sensors performing a remote analysis which will help pilots make better decisions in the troubleshooting process.

As previously mentioned the current avionics protocols can be divided in two families: **OldGen** and **NextGen**. The **OldGen** is currently deployed and used globally while the **NextGen** is standardized and set to be fully deployed by 2020. For this reason, to comply with regulations and to enhance safety Project Loon balloons are equipped with both OldGen and NextGen hardware[11] while some DJI drones are compatible with NextGen protocols[12].

In addition to their use in aircrafts both OldGen and NextGen protocols are now integrated in a big network of IoT sensors. It is currently estimated there are no less than 32000 air-traffic sensors/receivers integrated into crowd-sourced projects, in particular about 17000 in FlightRadar24 [13], around 15000 in Flightaware [14], and at least 700 in OpenSky-Network [15]. These sensors/receivers can be either general purpose devices such as RaspberryPi (RPi) and routers plugged with USB

RTL-SDR dongles, or can be special-purpose ADS-B receivers using specialized FPGA implementations. Moreover, it is expected the number of aircraft, including air carrier and General Aviation (GA), that are equipped or upgraded with avionics embedded devices supporting ADS-B and other NextGen protocols will surpass 100000 [16]. These numbers do not account for the considerable number of devices that are deployed at various ATC towers and that process avionics RF protocols. Therefore, given the number and the Critical Infrastructure (CI) functions of those devices, it is important to secure those systems and to be sure that no flaws exist in the protocols.

2.1 OldGen

The first generation of protocols is fairly simple; it uses the 1030MHz frequency for interrogation and the 1090MHz frequency for replies. The first two protocols are **Mode A** and **Mode C**.

Mode A is the simplest because it responds to an interrogation request just by broadcasting the **squawk** code which was previously assigned to the pilot by the controller. In this way the controller can identify the aircraft on his screen by such code. In addition to this, the pilot can manually generate a special response called *"Ident"* or *"SPI"* (Special Identification Pulse) which is used to highlight the aircraft on the controller screen.

Mode C is an extension of the previous protocol which, in addition to the transponder code, sends information about the altitude and the pressure. This kind of transponders are often referred to as **Mode A/C**.

Mode S is the newest protocol of this family and it is an hybrid between the two generations. It was born as part of the OldGen family but at the same time it is also part of the NextGen as it can be seen in Figure 2.1. As clearly explained further in the text **Mode S** in the years has undergone various enhancements and refinements in order to be used as an easy-to-deploy and affordable NextGen protocol. **Mode S** ground stations and transponders support both all call interrogations and selective interrogations, in particular each aircraft is identified by a unique 24-bit address which is part of the aircraft registration documents and should never be changed. The address is transmitted with every **Mode S** reply, which allows a SSR station to perform an all call interrogation in order to acquire the address of each aircraft which can then be used to selectively interrogate them. Each ground station is identified by a 4-bit[3] Interrogator Code (IC). In this way single aircraft interrogation as well as lock-out of the aircraft can be performed in order to avoid multiple pickups of the same aircraft by different SSR stations. **Mode S** messages can be of 56 or 112 bits and they are structured as in Figure 2.2.

In **Mode S** messages the first 5 bits identify the type of message called Downlink Format (DF), then the other bits, except for the last 24, are message dependent. **Mode S** comes in two versions: Elementary Surveillance (**ELS**) and Enhanced Surveillance (**EHS**). **ELS** has been described above and is the minimum

[3]For aircraft complying with ICAO Annex 10, Volume IV Amendment 73 a 6-bits code can be used

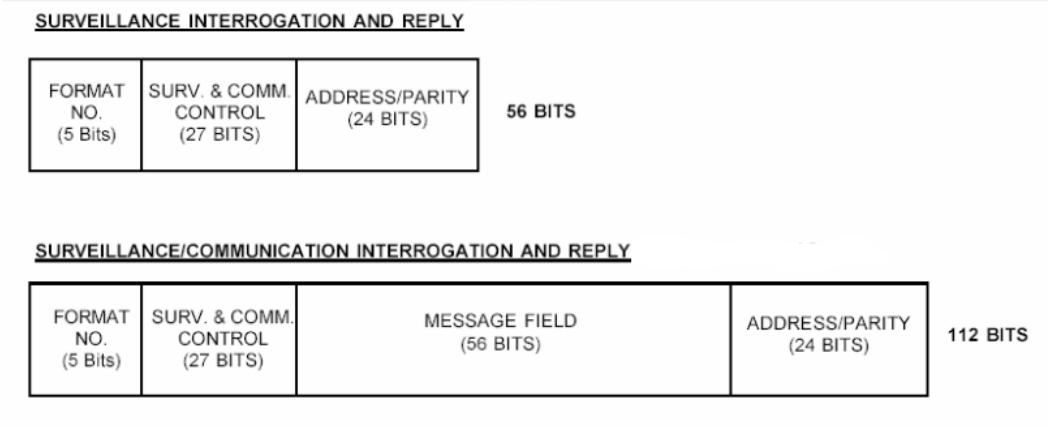

Figure 2.2: Mode S messages

version required in European Mode S airspace. **EHS** introduces some improvements:

- Reduced workload for the pilots and the controller, since some radio communications can be avoided when the controller has access to information such as the Magnetic Heading or the Selected Altitude.

- Improved Situation Awareness, since the controller can have access to a greater amount of data like Vertical Climb Rate, Magnetic Heading, Selected Altitude and similar.

- Safety Enhancements brought by data such as Selected Altitude which permits the controller to check if the aircraft is following the instructions and avoid potential collisions or level bust in advance [17].

Mode S EHS is a minimum requirement for some categories of aircraft flying in European Mode S airspace.

All the above protocols are at the base of the Airborne Collision Avoidance System (**ACAS**). This system requires all flying objects to be equipped with **Mode A/C**, **Mode C** or **Mode S** transponders. In this way it can keep track of the surrounding airplanes and, in case of colliding traffic, propose vertical Resolution Advisory (**RA**). NextGen systems have enhanced **RA** capabilities. It is clear that this is a critical part of the system and, if compromised, can lead to dramatic results such as mid air collision.

Mode A/C and **S** are the current standards and they are widely used, notably according to the Italian regulation all aircrafts must be equipped with at least a **Mode A/C** transponder and in some particular cases with a **Mode S** transponder as it can be seen in GEN 1.5-3.1 of AIP Italia[18].

Beside the basic protocols described here, a more sophisticated one lays between the two generations. The Aircraft Communications Addressing and Reporting System (**ACARS**) has been in use since 1978. At first it relied only on VHF radio channels but in the years it has been improved to add other transmission means to expand coverage. It is also now deeply integrated into the aircraft systems giving it access to a large number of data and the ability to operate autonomously. Acars messages can be delivered via 3 different transmission means: VHF Data Link, HF Data Link and satellite. Depending on the position of the aircraft, one method can be better than the other, specifically VHF works

only in line of sight while satellite communication (SATCOM) is not available at the poles. **ACARS** messages can be of 3 different types:

- Air Traffic Control messages. Used in some busy airports as an alternative to the radio. And in routes where no radio contact is possible (e.g oceanic routes).
- Aeronautical Operational Control (AOC) and Airline Administrative Control (AAC) used to send documents to the aircraft as well as to receive error messages or information on the status of the flight.
- Free Text Message.

Messages from the aircraft can be pre-configured so that they are automatically delivered to the appropriate recipient based on the message type; in the same way ground-originated messages can be configured to reach the correct aircraft. This message covers a fundamental part of the bidirectional aircraft-ground data link and their content can be of utmost importance. For example the Air France flight 447 which disappeared from the radars on 1 June 2009 sent in his final moments 24 **ACARS** messages, some of them indicating anomalies and errors. In the first moments those messages were the only clue to understand what had happened and to locate the aircraft [19].

Current minimum system requirements of an airliner can be seen in Figure 2.3. However by 2020 all the **Mode S** transponders must be upgraded to support **1090ES** NextGen protocol.

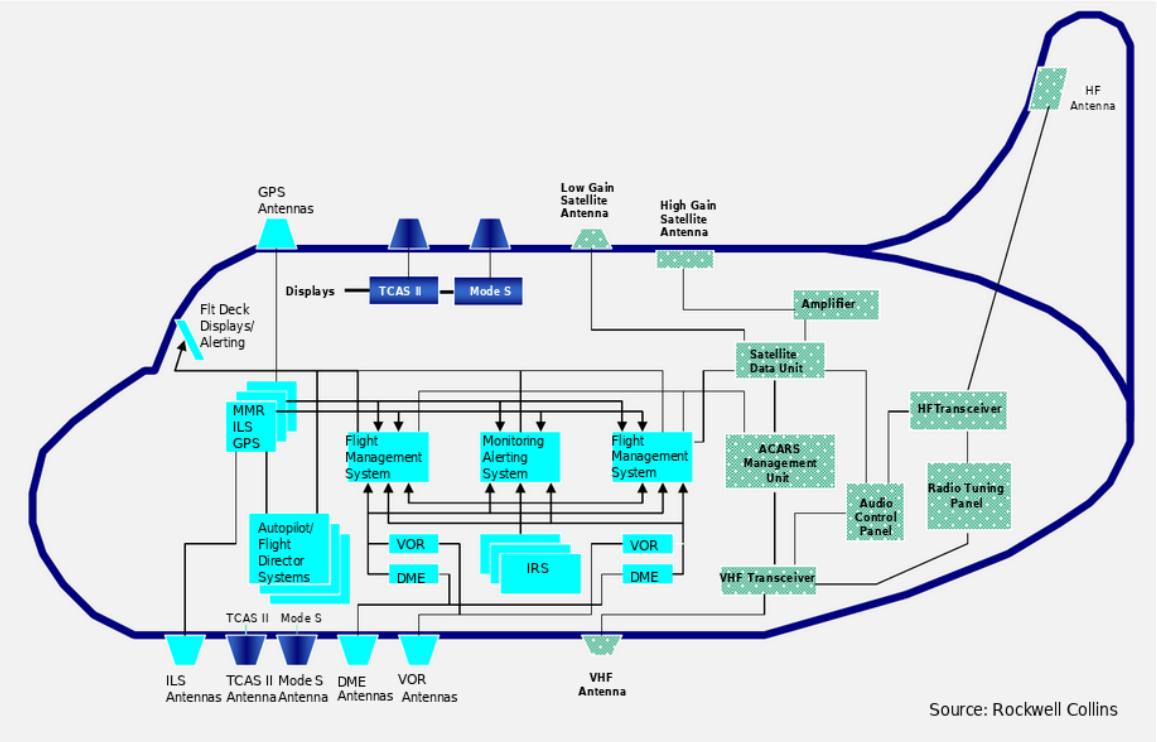

Figure 2.3: Current avionics minimum requirements

Although the OldGen provides some benefits in term of position accuracy and details for the controllers, there are still some problems:

- The information provided is still low and, except for the enhancements introduced with **Mode S**, it only provides a small aid during the flight.

- Using such protocols is still better than the radar alone; however, the accuracy level is still low and usually not much information can be carried so a intense verbal interaction with the controller is still required.

- There is no effective way to confirm the identity of an aircraft and the validity of the data that it is sending, thus allowing for no effort Man in the Middle and other attacks.

Different approaches to secure such protocols [20] (both Old an NextGen) have been proposed. Unfortunately, none of them will be adopted in the transition to the NextGen but hopefully some may be used in the near future since there are studies and official documents about that [21].

2.2 NextGen and SESAR

Both **NextGen** and **SESAR** (Single European Sky ATM Research) are efforts to modernize the air transportation system respectively from the United States government and from the European Union along with other private parties. Both programs aim to achieve a higher degree of safety enhancing communications, navigation, surveillance technologies thus enabling all the users of the sky, even the future ones, to virtually see each other. The introduction of those new standards will also bring a reduction of costs through better ATM and enabling low visibility operations. The efforts are coordinated in order to assure a globally accepted common standard. For this reason from now on I will make no distinction and refer to both protocols as **NextGen**. However, many different protocols go under the **NextGen** family and some of them are not shared between Europe and America even if there are plans to do that.

The main component of this system is Automated Dependent Surveillance - Broadcast (**ADS-B**). *"The American Federal Aviation Administration (FAA) as well as its European counterpart EUROCONTROL named ADS-B as the satellite-based successor of radar."*[22] **ADS-B** is an automatic system which broadcasts aircraft sensors information to the outside world. It is divided in "ADS-B Out" which requires just a transponder able to properly encode messages and "ADS-B In" which requires a receiver, a computer and an interface to display the data. **ADS-B** messages are also picked up by ground stations which feed the data to a central system where they are used, in combination with other data (e.g radars), to create a Traffic Situation Picture.

Two main protocols were proposed to deliver **ADS-B** messages:

- **1090ES** (Extended Squitter)
- **UAT** (Universal Access Transceiver)

1090ES it is the current global standard for **ADS-B** in commercial aviation. In Europe **1090ES** is required for IFR aircraft with a MTOW exceeding 12,566 pounds or maximum cruise airspeed faster than 250 KTAS. All aircrafts must comply with this regulation by 2020. [23]

In USA **1090ES** will be mandatory for all aircrafts after June 2020 and is the only technology that should be used when flying above 18,000 feet. [24]. An overview of the USA supported technologies in the various airspace can be seen in Figure 2.4

This protocol is backward compatible with the **OldGen** since it uses the same frequencies and, in particular, a transponder supporting **Mode S** messages can be easily updated to support **ADS-B**. Therefore **ADS-B** information is simply carried inside a 112 bit **Mode S** message as it can be seen in Figure 2.5. However, the message structure is different: The first 8 bits are used to identify the message and in particular the first 5 bits identify the type of message (10001 and 10010 for an **ADS-B** message) while the next 3 bits represent the capabilities which are different for every message type. The next 24 bits are the ICAO address of the aircraft. Then there are 56 bits containing data and the last 24 bits of CRC code. These last bits are particularly important as they allow to correct several errors in one message. All these characteristics are common with the **Mode S** messages so **ADS-B** messages encode their type and information in the 56 bits reserved to data. Data segments contains the first 5 bits indicating a type code for the **ADS-B** message and then other bits encode specific information depending on each message.[17]

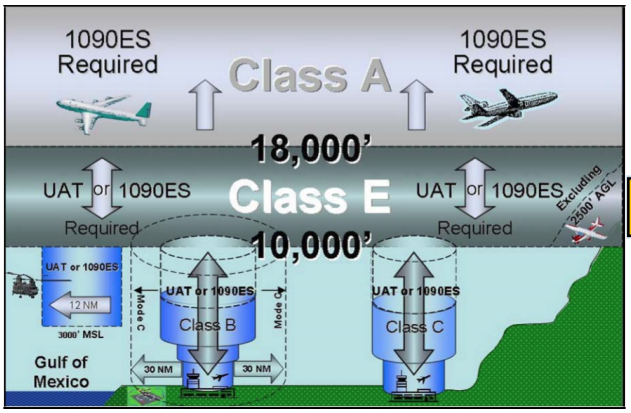

Figure 2.4: USA airspace

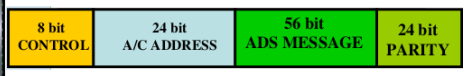

Figure 2.5: 1090ES Message

UAT-Universal Access Transceiver is not actually Universal as it is not widespread yet like **1090ES**. This protocol is mainly deployed in North America and it is starting to be deployed also in China. It is not backward compatible with the **OldGen** and it was developed specifically to be used with **ADS-B**. It uses the 978MHz frequency and requires an entire new hardware to work.

However, since it is a newly designed protocol built to be future proof it allows to carry more information compared to **1090ES**, such as weather reports, pilots reports and other messages (like NOTAMs) using **FIS-B** (Flight Information Services - Broadcast) and **TIS-B** (Traffic Information Services - Broadcast), which are ground services that broadcast to "ADS-B In" equipped aircraft information on weather and traffic detected by ground radars as well as coming from other ADS-B equipped planes. Moreover **FIS-B** service is also capable to deliver **NexRad** images captured from the United States Weather Surveillance Radar through **ADS-B** messages. US government is encouraging General Aviation (GA) aircrafts, not flying in Class A airspace, to use a **UAT** transponder. In

this way there is less pollution of the 1090MHz frequency and they can receive more information such as weather data.

Multilateration is a technology which uses different signals to accurately locate an aircraft. It was initially developed for military purposes but was then adopted to confirm the position transmitted by **ADS-B**. It employs strategically placed antennas that listen to different replies (Mode A,C,S, military IFF and ADS-B) transmitted from an aircraft. Since a single aircraft will be at a different distance from every ground station, the replies to each station will have a different time of arrival; this difference can then be used to compute the precise position of an aircraft. This is the inverse of triangulation and requires no additional hardware since the ground stations are able to acquire replies from a multitude of different protocols. This also makes **multilateration** a faster method to locate an airplane since information can be acquired at a considerable higher rate. However, it has been demonstrated by Bran Haines in [25] that such a system is not mandatory and rarely used.

NextGen protocols will also replace a big part of the voice communications not only between pilots and air traffic control but also between air traffic controllers themselves using the **UAT** data link and the previously mentioned **ACARS** messages. In particular **ADS-C** (Automatic Dependent Surveillance - Contract) automatically aggregate data such as aircraft position, altitude, speed, intent and meteorological data from on-board sensors. The generated report can then be sent to an ATS (Air Traffic Service) unit or AOC (Air and Space Operations Center) facility ground system for surveillance and route conformance monitoring [26]. An ATS ground station must authenticate itself on the aircraft system to require a contract. Contracts can be of 3 types:

- Periodic: the ATS specify the time interval at which the aircraft sends the report.
- Demand: a single report requested from the ground station. This request does not affect any other contracts that might be present.
- Emergency: these reports are tagged as "emergency" reports and will be highlighted to the ATC. Emergency reports can be generated manually by the crew or as a consequence of triggering another type of emergency system.
- Event: the ATS unit can specify the event (limited to 1 per aircraft) at which the report will be sent. The contract can contain multiple event types (e.g lateral deviation, vertical rate change ecc).

Security researchers in this field are focusing on **ADS-B** and various researches [6, 7] have already demonstrated the insecurity of such protocols in the **ATC/ATM** world. Few researches have been done on the aircraft side and it is now public and clear that this is a feasible entry point to an aircraft system [9].

Chapter 3

Testing and Validation

3.1 Software testing

Software testing can be conceptually divided in two main groups: **Static Testing** and **Dynamic Testing**. While **Static Testing** includes only having the code reviewed by a human being the **Dynamic Testing** is mainly based on an automated approach. In particular many different automated tests can be conducted on a single piece of software by an autonomous program. The results of such tests are then used to correct bugs in production (e.g. agile development) or to exploit such bugs if the test is conducted by a malicious counterpart.

In the research I mainly used fuzz testing mixed with others methods such as source code review and reverse engineering. **Dynamic Testing** can be accomplished using different approaches which are: **Black Box**, **White Box** and **Gray Box**.

3.1.1 Black Box

For this methodology no access to the source code or knowledge of the design specifications and internal mechanisms of the program is required. The test is based only on what can be observed which means giving to the target different inputs and monitoring his behavior. This definition reflects exactly the interaction that a typical user might have with the program. This are exactly the cases that this kind of testing wants to examine and it should test typical and atypical user behavior. Black box testing is widely used and it includes all the techniques where the only available information comes from the user interaction with the application. One example among all is SQL injection where the test is conducted against a plain web page without any previous knowledge of the logic of the servers and the various scripts. Only interacting with them allows the user to get an idea of the inner working mechanisms. The test can be performed directly by the user (manual testing) or using ad hoc programs (automated testing) such as SQLmap[1] in this case. Black Box testing has many advantages: it does not require access to the source code so it is always applicable, moreover even in the presence of source code it can still be effective. Since it is not source code dependent an

[1] http://sqlmap.org/

effective test case against one program (for example a zip extractor) can be easily reused to test any program that implements the same functionality. However due to his simplicity this approach can only scratch the surface of the application being tested, achieving a extremely low code coverage. Moreover complex vulnerabilities which requires multiple attack vectors will never be triggered and can only be discovered with a White Box or static approach.

3.1.2 White Box

This methodology is nothing more than a source code review or analysis, it can be done either by a human or by automated tools. Human analysis in not always feasible and might be inaccurate since, especially in large projects, it is easy to miss one line or make simple mistakes. For this reasons automated source code analysis is the way to go in a white box approach. This method has many advantages mainly regarding code coverage since it is possible to test all the paths for vulnerabilities. However, source code rarely is available for projects outside the open source community and this approach still requires a human interaction to review the results and identify the parts that are actually related to security problems. Moreover reviewing a large project, even with the use of automated tools, might require significant time to complete, effectively bringing the performance down to the level of the BlackBox approach.

3.1.3 Gray Box

The definition of this technique is quite trivial because a lot of different methods can be seen as Gray Box approach. This takes the Black Box method as a starting point augmenting it with the use of common reverse engineering and binary analysis techniques using disassemblers, debuggers and similar tools. The majority of those tools still requires human interaction and, reverse engineering a program can often be a tedious and hard job. However there are some automated tools that aims to facilitate this analysis, for example one of them is Ropper[2] which provides information about a binary and automatically searches for particular vulnerabilities(ROP and JOP). For this reasons the Gray Box technique can efficiently improve the coverage produced by the classic Black Box approach still without requiring access to the source code. However reverse engineering is a complex task and requires specific set of skills and tools which might not always be available.

3.2 The fuzzing world

"Fuzzing is the process of sending intentionally invalid data to a product in the hopes of triggering an error condition or fault." [27]

Fuzz Testing dates back to 1989 when it started as a project of the operating systems course at the University of Wisconsin-Madison. At his first stages it was nothing more than a simple black box technique to test the robustness

[2]https://github.com/sashs/Ropper

of some UNIX utilities. The first two fuzzers *fuzz* and *ptyjig*, developed by two students, where incredibly successful in finding previously unknown vulnerabilities: they where able to crash around 30% of the considered utilities using random inputs[28]. It quickly become clear that this was a powerful and strong technique to test the robustness of a piece of software.

During the years fuzz testing evolved and developed gaining popularity and expanding to more and more fields (networks, files, wireless and more) until it became a proper field of research and engineering. As for today this field is quite vast and, using this kind of tools in the software testing phase should be a best practice. However it is still uncommon to find fuzzing outside the security world, for this reason is hard to give a precise and unique definition for this technique. I will try to give an overview of the actual fuzzing world focusing in particular to the binary fuzzing and explaining how the previously mentioned techniques are applied to the fuzzing world.

Fuzzers can be at first divided in two main categories:

- **Mutation Based (or Dumb)**: apply random mutations to the given input without any knowledge of the data that are being manipulated. Common mutations include bitflipping, shortening or expanding the input and similar. This method is a really simple and pure brute force approach, it can be applied to user provided inputs as well as to automatically generated ones to progressively create a valid input.

- **Generation Based (or Smart)**: requires some knowledge of the data or protocol that is being fuzzed, this information are then used to generate proper inputs. Such information are usually provided by the user which defines a specific set of rules or autonomously extracted from a sample by the fuzzer. For example generating valid CRC codes for mutated strings or keeping the correct structure for particular files (jpg). Obviously this approach has some advantages but in some cases a mutation based approach can give better results as it can test programs even outside their functioning boundaries.

As always the best sits in the middle so a fuzzer that combines the two methods is the one that can give the most comprehensive results.

We can then relate fuzzers to the previously mentioned testing methods, in this way fuzzers can be further differentiated by the approach that they have to the problem and the information they require to test the binary. This divides the fuzzing world in three theoretical categories: **BlackBox Fuzzing**, **WhiteBox Fuzzing** and **GrayBox Fuzzing**. **BlackBox Fuzzing** is the pure black box approach that we discussed earlier, it can be both mutation and generation based and it can usually only scratch the surface of a program. **WhiteBox Fuzzing** requires access to the source code and is a really sophisticated approach which is rarely used since it uses code analysis to generate test cases that will produce full code coverage. For this reason this approach is usually resource and time intensive. **GrayBox Fuzzing** sits in between the two and it usually works with or without access to the source code. This method consists in injecting special code during the compilation phase or sandboxing the binary inside an emulator in order to extract information on the execution status and the code coverage achieved.

However fuzzing is not an exact science, in the real world choosing the right approach can be trivial and usually there is not a clear distinction between the different categories.

3.3 State of the art

There are many software available for many different fuzzing jobs. The choice is vast and goes from the simple hobby project to the much more complex and advanced commercial software. Choosing a good fuzzer is a crucial step for the success of the task, a fuzzer which is too basic and relays solely on a pure brute force approach (like *ptyjig*) can require ages to find a simple bug on a modern program where, a "smarter" fuzzer, will take just few minutes or hours.

The following two software represents the actual state of the art in term of binary fuzzers.

3.3.1 American Fuzzy Lop

American Fuzzy Lop or **AFL** [3] represents the state of the art among binary fuzzers, moreover it has a high rate of vulnerabilities discovery as stated int the "bug-o-rama trophy case" section of the website, it is widely used and in active development.

AFL has many different features that makes it a quite sophisticated software. It uses gray box approach which require access to the source code in order to perform what it calls "instrumentation". This process consist in adding some custom code to the program that is being compiled; this automatic operation, performed by the custom compilers `afl-gcc` and `afl-clang-fast`, adds small code snippets in critical positions without weighting down the program but allowing the fuzzer to obtain information on the execution flow at run time. It also uses **Mutation Based** approach which however uses some statistics methods to optimize the generation making it efficient and faster to setup.

Beside instrumentation **AFL** can fuzz black box binaries leverage the QEMU emulator to obtain information about the execution state of the binary and adjust the generation method accordingly. Adopting a black box approach can be useful also when the source code is available since it can highlight vulnerabilities introduced in the compilation/optimization phase. However this goes at cost of execution speed which considerably slows down the testing.

The information gathered by the fuzzer at execution time will then be used to meaningfully mutate the input. **AFL** has two different input generation techniques depending on the information supplied: if a file containing a valid input for the program is supplied it is used as a starting point to the next stages otherwise if a blank file is supplied afl starts a "0-day" input generation. This last technique has been proven to be particularly successful, even if more time consuming then the other one, and interestingly **AFL** was able to generate valid URLs, JPG headers, XML syntax and more [29]. More specifically the tool uses a genetic algorithm approach keeping a queue of interesting inputs that will then

[3] `http://lcamtuf.coredump.cx/afl/`

be mutated and appending a file to the queue if it triggers a previously unknown internal state. Three different deterministic mutation strategies are applied:

1. Sequential bit flips with varying lengths and stepovers: this is a slow but very effective way of mutating the input as it has been observed that with different length of bitflipping it is possible to generate around 130 new paths per million input.

2. Sequential addition and subtraction of small integers: this method consists of incrementing or decrementing existing integer values in the input file. The operation is performed in 3 different stages: firstly on 8-bit values, then on 16-bit values in both endianess and lastly on 32-bit values. Those last two methods are performed only if during the operation the most significant byte is changed otherwise the value has already been tested in one of the previous cases.

3. Sequential insertion of known interesting integers: this method uses known particular values that are known for their ability to trigger interesting cases, for example -1, MAX_INT, MAX_INT-1 and so on. The values are tested in both endianess and in 8,16,32 bits.

There are also non deterministic strategies applied in a never ending loop and which includes insertions, deletions, arithmetics, and splicing of different test cases [30].

One of the best characteristic of **AFL** is that is open source meaning that every person can improve the code and modify it to meet different needs. For example one really active researcher in the fuzzing field, Dr. Marcel Böhme[4], created some really interesting forks of afl trying to achieve better code coverage[31][32], and better path discovery[33].

afl-unicorn

afl-unicorn is a fork of **AFL** that was build to leverage the power of **AFL** to fuzz trivial binaries *"For example, maybe you want to fuzz a embedded system that receives input via RF and isn't easily debugged. Maybe the code you're interested in is buried deep within a complex, slow program that you can't easily fuzz through any traditional tools."*[34]. In particular it uses the unicorn engine[**unicorn**], which is a CPU emulator, to run the extracted context of a previously running instance of the same program making it architecture independent and easy to selectively fuzz. The unicorn engine allows to fuzz a program in the exact same way as before but, in addition to this, one have now control over the memory, the CPU registers, and the code of the program. This is useful in many different situations, for example when there is no access to the source code or when the sources are available but you want to fuzz a pre-compiled binary for many different reasons (to test the behavior of a compiler, to test a specific version for a specific architecture and so on). Moreover *afl-unicorn* allows the user to specify portions of code to be fuzzed, for example a single function, in addition to this the fuzzing takes place directly inside the emulated memory manipulating a user defined region. It is also possible

[4] `https://comp.nus.edu.sg/~mboehme/`

to write custom "hooks" when certain functions are called or particular addresses are reached so to skip specific functions or to trigger a defined behavior, such as a crash, if the program reaches a defined portion of the code.

While it is true that this approach is very powerful it requires a template to be hand written for the unicorn engine. This requires specific information like the heap addresses, the register values and the content as well as the addresses of all the memory regions. The extraction of such information is a trivial process since a binary must be running inside a debugger and, usually, the relevant portion of code must be identified, only then the template can be written. This process is time consuming and requires very specific knowledge and hardware as a matter of fact the context of the process must be dumped from the memory and the binary must be disassembled to find the boundaries inside which the emulation must take place. For those tasks some tools like *unicorn_dumper.py* and a basic template where provided withe the tool, other tools like *unicorn_template_generator.py* and *extract_from_memory.py* where specifically wrote for the task by me and will be illustrated in Chapter 4.

3.3.2 Peach

Peach is another very popular fuzzer and is the commercial counterpart of **AFL** however the two software are a bit different since this one is capable of fuzzing different targets such as network protocols, device drivers, file consumers, embedded devices, external devices natively while in **AFL** all those capabilities are added by specific forks. Another difference comes in the generation of inputs as a matter of fact `Peach` is a **Generation Based** fuzzer that requires a *"Peach Pit"* which is a template describing the protocol or the data format that is being tested. A *Peach Pit* contains different information:

- A description of the data layout to test
- The agent, monitors and I/O adapter to use in the fuzzing session
- Setup parameters, such as Port or Log Folder, to use in the fuzzing session

As a plus *Pits* are open source and shared inside the **Peach** community which makes finding the perfect *Peach Pit* highly likely. Moreover form the Whitepaper is possible to understand that there are *"over 50 mutation algorithms"* but there is no explanation on what those algorithms are doing or how [35].

Peach was open source but from version 3 the policy changed as it is stated on the website: *"Peach is not open source software, it's FREE software. Peach is licensed under the MIT License which places no limitations on it's use. This software license was selected to guarantee that companies and individuals do not have to worry about license tainting issues."* [36]. This license is ambiguous in fact on the company website a Community Edition of **Peach 3** is available and it is stated to be open source (source code is also available). This is probably due to the fact that the company sells fuzzing services using **Peach** so the commercial version might have more features. However an in development fork of **Peach 2.7** is maintained by Mozilla.[5]

[5]https://github.com/MozillaSecurity/peach

Chapter 4

Analysis

Accessing real avionics hardware and software was out of reach for this research, therefore the analysis was carried out on open source implementations of **ADS-B**, **FIS-B** and **NexRad** as found in many popular source-tree, binary-package and device-firmware releases such as Stratux, FlightRadar24 and FlightAware. Testing on those software has also secondary relevant implications as it allows to test at the same time avionics protocols as well as IoT devices. As a matter of fact, even though avionics software must comply with strict regulations and security standards as described in DO-178 (ED-12 in europe) and DO-278, it is highly likely that using one of the previously mentioned protocols as attack vector will have the same effect on both implementations.

Fuzzing was choosen as test method since, as far as it was known at the beginning of the research, no previous attempt of applying this test method to avionics protocols had been carried out. This research acquire even more importance if we examine the latest news about the hacking of a Boeing 757 and the Cyber Grand Challenge (CGC) 2016 [37]. In particular, the CGC contained a specific challenge (`FSK_Messaging_Service`) [38] tailored to identify techniques and systems able to discover vulnerabilities in the RF software using processed data after the RF front-ends. Aircraft and avionics radio-communication interfaces are a perfect example of that type of design. *afl-unicron* has been demonstrated to be particularly effective on solving this specific challenge of the CGC.

There is no official linking between the CGC and the Boeing hack however, the circumstances indicate that there is a search and a need for knowledge and tools that can exploit (and therefore eventually protect) RF facing software, embedded devices in general and avionics and NextGen devices in particular.

T0D0 2: motivations

4.1 Hardware Setup

The hardware used during the research was the following:

- *DVB-T (RTL-SDR) dongle:* a simple and cheap tv receiver equipped with the *RTL2832U* or compatible chip it can be tuned on a very wide range of frequencies, not only on the tv ones.

- *RaspberryPi (RPi) 3 Model B:* used to acquire the data using RTL-SDR dongles, and also to run some tests on the precompiled binaries. The RPi was running the latest version of Raspbian, or the particular flavor/version of Linux or Raspbian bundled as part of Stratux and FlightRadar24 SD-card firmware image files.

- *Standard laptop:* used for dry-run testing, for initial code testing and for initial fuzzing experiments monitoring. The laptop has the following specifications: Intel Core i7 5500 (4 cores), 12 GB of RAM. Since the fuzzing is time and resource consuming, when the running experiments were considered promising or really important, they were moved to run on a multi-core server.

- *Multi-core server:* with following specifications: Intel Xeon CPU E7-8837 @ 2.67GHz (64 cores, 32 cores used for a single given afl-fuzz instance), 1 TB of RAM, running Centos7.4 Linux.

T0D0 3: why

4.2 Software Setup

T0D0 4: Explain why we used what we used.

The focus was posed on two of the most widespread and common implementations: *dump1090* and *dump978* obviously the first one decode **OldGen** and **1090ES** messages while the second one decode **UAT** messages. Both of them uses *librtlsdr* to communicate with the rtl-sdr dongle

Tested software have two main components:

1. A *demodulator* that converts the raw signal coming from the dongle into a properly encoded string. This basically means converting waves into bits.

2. A *decoder* that decodes the messages coming from the demodulator. Basically extracting the information from the incoming packet.

dump1090 is a big monolithic software which has undergone major reworks and edits from many different authors. For this reason it is not possible to test individually each components.

4.2.1 Datasets

In order to test the above mentioned programs some data were needed.

4.2.2 Tools

During the project I wrote a set of tools and scripts that played a major role in automating and simplifying some of the data processing and fuzzing tasks. Such tools are divided in two categories: **Data Tools** used to manage, modify and create the datasets. **Fuzzing Tools** used to facilitate and speed up the fuzzing process. The details and descriptions of the tools are as follows.

Data Tools:

- **converter.sh** and **runner.sh** this scripts are meant to interact with the data provided by the DO-358 zip file.This archive contains different files which are designed to be used with a dedicated test tool for real avionics hardware, for this reason there are 18 subfolders (called groups) and for each one of them there are 3 files: `TestGroupXX Procedures.doc`, `TestGroupXX Stimulus.csv` and a `bin` folder containing the actual data files. Each one of this folders contain many different files, each representing a unique type of information which has already been demodulated and is stored in a binary format. Since the program that we want to test only accepts data in a uplink(or downlink) format we wrote **converter.sh** to convert one single file or an entire directory in the appropriate format. Every file contains just one encoded type of data and with many files feeding them into a program will be long and tedious. The **runner.sh** script will feed each file contained inside a specified directory through the specified program either interactively, namely stopping after each file and asking if the user wants to continue, or simply running all files at one.

- **message_generator.py** is a simple script that given the first part of a demodulated Mode-S message will calculate a correct CRC, it can do this virtually for every random string with a correct length. This is only for test purpose since we wanted to see what would happen when the test program is fed with messages composed by all 1 or all 0 having a valid CRC.

Fuzzing Tools:

- **start.sh**: afl can be parallelized on many cores, to do that a different command with slightly different parameters must be issued for every instance that you want to spawn. For this reason I wrote **start.sh** which takes as input the number of cores that the user wants to use and the other parameters required by the fuzzer. The script will then generate the required directories and then start the chosen number of fuzzers: 1 Master and $n - 1$ Slave.

- **unicorn_template_generator.py**: `afl-unicorn` requires quite some time to set up all the environment and gather all the information needed to properly write the template for the Unicorn engine. For this reason we wrote this script which will create the template for the Unicorn engine and populate it with proper addresses. The script is designed to be sourced into GDB while the debugger is on a breakpoint inside a function, it will not work if called inside the main since what we want to test is usually a particular function.

- **extract_from_memory.py** will dump the specified memory region from a running program in gdb so it can be used as input to `afl-unicorn`. More information on how `afl-unicorn` works and why this scripts are needed can be found in 3.3.1.
- **afl utility scripts**:
  - **killem_afl.sh** → stop a running instance of afl.
  - **wazzup_afl.sh** → get information and statistics on the running instances of the specified fuzzer.
  - **afl_noroot.sh** → run afl on a system where the user does not have root privileges.

4.3 Proceedings

T0D0 5: which test we run, why and the result

Chapter 5

Results

5.1 Caveats

T0D0 6: Put all the problems we had here

5.2 Results

T0D0 7: Put the signicative Results here and the state of the fuzzers

Chapter 6

Conclusions and Future Work

T0D0 8: Future work and proprietary fuzzer development

Bibliography

[1] Amazon. *Prime Air Service - drone delivery system*. URL: https://www.amazon.com/Amazon-Prime-Air/b?ie=UTF8&node=8037720011.

[2] Google. *Project Loon*. URL: https://x.company/loon/.

[3] PrecisionHawk. *Drones Sensors*. URL: https://www.precisionhawk.com/.

[4] Vigili del Fuoco. *Italian Firefighters adopting drones for SAR operations*. 2017. URL: http://www.vigilfuoco.it/aspx/notizia.aspx?codnews=40594.

[5] Facebook. *Aquila drones for global internet connection*. URL: https://info.internet.org/en/story/connectivity-lab/.

[6] Andrei Costin and Aurélien Francillon. "Ghost in the Air (Traffic): On insecurity of ADS-B protocol and practical attacks on ADS-B devices". In: *BlackHat USA (BHUS)* (2012), pp. 1–12.

[7] Brad Haines. *Hackers + Airplanes*. Defcon 20 Slides. 2012. URL: https://renderlab.net/projects/ADS-B/Hackers-Airplanes-Defcon20-RenderMan.pdf.

[8] Hugo Teso. *Aircraft Hacking*. HTBSECCONF. 2013. URL: https://conference.hitb.org/hitbsecconf2013ams/materials/D1T1%20-%20Hugo%20Teso%20-%20Aircraft%20Hacking%20-%20Practical%20Aero%20Series.pdf.

[9] Ms. Smith. *Homeland Security team remotely hacked a Boeing 757*. URL: https://www.csoonline.com/article/3236721/security/homeland-security-team-remotely-hacked-a-boeing-757.html.

[10] International Civil Aviation Organization (ICAO). *Convention on International Civil Aviation (doc. 7300/9)*. 1944. URL: https://www.icao.int/publications/Documents/7300%5C_cons.pdf.

[11] Google. *Project Loon ballons Mode C and ADS-B out*. URL: https://x.company/loon/faq/#flight-section.

[12] DJI. *Equipping drones with ADS-B*. URL: https://www.dji.com/newsroom/news/dji-and-uavionix-to-release-ads-b-collision-avoidance-developer-kit.

[13] flightradar24.com. *How it works*. https://www.flightradar24.com/how-it-works.

[14] flightaware.com. *FlightAware and ADS-B*. https://flightaware.com/adsb/.

[15] Matthias Schäfer et al. "OpenSky report 2017: Mode S and ADS-B usage of military and other state aircraft". In: *IEEE/AIAA Digital Avionics Systems Conference (DASC)*. IEEE. 2017, pp. 1–10.

[16] Woodrow Bellamy III. *Equipping 100,000 Aircraft*. http://interactive.aviationtoday.com/avionicsmagazine/june-july-2016/equipping-100-000-aircraft/.

[17] Junzi Sun. *The 1090MHz Riddle*. URL: http://mode-s.org/decode/book-the_1090mhz_riddle-junzi_sun.pdf.

[18] Ente Nazionale Assistenza Volo (ENAV). *Aeronautical Information Package (AIP) Italia*. URL: https://www.aopa.it/pdf/AIRAC11-14.pdf.

[19] Air France. *Information on ACARS system*. URL: http://corporate.airfrance.com/en/acars-system.

[20] E. Cook. "ADS-B, Friend or Foe: ADS-B Message Authentication for NextGen Aircraft". In: *2015 IEEE 17th International Conference on High Performance Computing and Communications, 2015 IEEE 7th International Symposium on Cyberspace Safety and Security, and 2015 IEEE 12th International Conference on Embedded Software and Systems*. Aug. 2015, pp. 1256–1261. DOI: 10.1109/HPCC-CSS-ICESS.2015.201.

[21] International Civil Aviation Organization (ICAO). *ADS-B Implementation and Operations Guidance Document*. 2014. URL: https://www.icao.int/APAC/Documents/edocs/cns/ADSB_AIGD7.pdf.

[22] Martin Strohmeier et al. "Realities and challenges of nextgen air traffic management: The case of ADS-B". In: 52 (May 2014), pp. 111–118.

[23] European Union. "Commission Implementing Regulation (EU) 2017/386". In: *Official Journal of the European Union* (2017).

[24] federal government of the United States. *Electronic Code of Federal Regulations - Title 14 Aeronautics and Space*. 2017.

[25] Bran Haines. *They are doing WHAT! With Air Traffic Control*. Hackfest. 2014. URL: https://renderlab.net/projects/ADS-B/Hackfest.pdf.

[26] International Civil Aviation Organization (ICAO). *Global Operational Data Link (GOLD) Manual*. 2016.

[27] Pedram Amini Michael Sutton Adam Greene. *Fuzzing: Brute force vulnerability discovery*. Addison Wesley, 2007.

[28] Bryan So Barton P. Miller Lars Fredriksen. "An Empirical Study of the Reliability of UNIX Utilities". In: (1990). URL: ftp://ftp.cs.wisc.edu/paradyn/technical_papers/fuzz.pdf.

[29] Michal Zalewski (lcamtuf). *blogpost about afl input generation*. URL: https://lcamtuf.blogspot.fi/2017/04/afl-experiments-or-please-eat-your.html.

[30] Michal Zalewski (lcamtuf). *Technical Details of AFL*. URL: http://lcamtuf.coredump.cx/afl/technical_details.txt.

[31] mboehme. *AFLFast (extends AFL with Power Schedules)*. URL: https://github.com/mboehme/aflfast.

[32] Van-Thuan Pham, Marcel Böhme, and Abhik Roychoudhury. "Coverage-based Greybox Fuzzing as Markov Chain". In: *ACM SIGSAC Conference on Computer and Communications Security* (2016).

[33] mboehme. *Pythia (extends AFL with Predictions)*. URL: https://github.com/mboehme/pythia.

[34] Nathan Voss. *afl-unicorn article*. URL: https://hackernoon.com/afl-unicorn-fuzzing-arbitrary-binary-code-563ca28936bf.

[35] Peach Fuzzer. *Platform Whitepaper*. URL: https://www.peach.tech/wp-content/uploads/Peach-Fuzzer-Platform-Whitepaper.pdf.

[36] Peach Fuzzer. *Software License*. URL: http://community.peachfuzzer.com/License.html.

[37] Defense Advanced Research Projects Agency (DARPA). *Cyber Grand Challenge (CGC)*. https://www.darpa.mil/program/cyber-grand-challenge.

[38] Jason Williams. *Cyber Grand Challenge (CGC) – FSK_Messaging_Service*. https://github.com/trailofbits/cb-multios/tree/master/challenges/FSK_Messaging_Service.