Simulating ADS-B Attacks in Air Traffic Management

Anton Blåberg
Department of Computer and Information Science
Linköping University
Linköping, Sweden
antbl294@student.liu.se

Gustav Lindahl
Department of Computer and Information Science
Linköping University
Linköping, Sweden
gusli687@student.liu.se

Andrei Gurtov
Department of Computer and Information Science
Linköping University
Linköping, Sweden
andrei.gurtov@liu.se

Billy Josefsson
Luftfartsverket
Norrköping, Sweden
billy.josefsson@lfv.se

***Abstract*—In Air Traffic Management (ATM) training, simulations of real air traffic control (ATC) scenarios are a key part of practical teaching. On the internet one may find multiple different ATM simulators available to the public with open source code. Today most aircraft transmit data about position, altitude, and speed into the atmosphere that practically are unencrypted data points. This data is called automatic dependant surveillance broadcast (ADS-B) data. The lack of security means that potential attackers could project "fake" ADS-B data and spoof existing data to air traffic controllers (ATCO) if the right equipment is used. We see this as a security flaw and we want to prepare ATCO for cyberattacks by modifying an ATM simulator with cyberattacks. First, OpenScope was chosen as the ATM simulator to be modified. Subsequently, three types of attacks were chosen for the simulator to be equipped with, based on ADS-B weaknesses from existing literature: aircraft not responding to commands, aircraft with altering positional data, and aircraft with incorrect speed and altitude data. The recorded parameters were the written command lines and corresponding aircraft type it was applied to. Using this modified simulator, ATCO can now be evaluated against cyberattacks.**

***Index Terms*—ADS-B, Security, Aviation, Cyberattack, openScope, Simulation**

I. Introduction

Global interest in cybersecurity has been steadily growing for the past seven years [1]. Today, it is common for enterprises to practice network security routines and people are realizing the benefits and necessity of protection against cyberattacks. This can be seen in the growing global interest in Virtual Private Networks services and cybersecurity [2], [1]. If people are so guarded regarding their personal data, should we not put as much focus on protecting vital information about the aircraft we travel with? Information regarding aircraft should be of much higher priority because cyberattacks in air traffic management (ATM) may lead to disaster.

Aircraft transmit data such as altitude, position, and speed to other aircraft and ground stations. This surveillance system is called Automatic Dependent Surveillance Broadcast (ADS-B). There has been scientific discussion about the flaws of ADS-B recently, but there is not as much discourse as there was a few years ago. In 2012, a team of security researchers proved that is was possible to spoof ADS-B data with a low-cost hardware setup combined with moderate software technology. Their research also concluded that it is impossible to verify the data points as real, or a spoof [3].

The same security researchers also claimed that the security flaws of ADS-B have been widely covered in other studies, but the core problems have never been addressed [3]. If this was known eight years ago and the system remains unmodified, one may argue that ADS-B is even more unsafe now. The reasoning behind this logic is that the attackers would have had more time to learn how to make more efficient attacks.

The number of problems found in the system are many, with few solutions found to solve them. In an article by Gurtov et. al. from 2018, the authors discuss solving a similar problem [4]. This team was researching how to make Controller-Pilot Data Link Communication (CPDLC) safer. Three different solutions are then presented that could possibly be used for securing ADS-B.

The primary aim of our paper is to prepare air traffic controllers (ATCO) against cyberattacks. The secondary aim is to further present the security flaws in Air Traffic Management (ATM) originating from ADS-B.

To reach these goals, the following research questions were formulated:

- How could a known ATM simulator be modified to track ATCOs responses to cyberattacks?
- How can a modified ATM simulator expose the weaknesses of ADS-B?

By answering these research questions we hope to get a simulator that will help preparing future generations of ATCO for cyberattacks.

The project was supported by Automation Program II, Trafikverket

This is an area that can easily get quite large and therefore it is important to know the delimitation of this paper. One such delimitation is that this paper will not survey multiple ATCOs with the simulator. This paper will also only focus on attacks on ATM that is made possible because of proven weaknesses in ADS-B. Instead this paper will give the tools needed for simulation training and evaluation of ATCOs.

The rest of the paper is organized as follows. Section II presents theory of ADS-B and Section III compares the two different simulators. Section IV says how the simulator was modified and Section V shows the end result of the simulator. In Section VI a discussion is made about the simulator. Lastly, the conclusion is presented in Section VII.

II. Background

In order to begin answering our research questions, the collection of background knowledge in a few areas was first necessary. The following section contains information on the communication technology ADS-B used in ATM and showcases its weaknesses. Later, different attacks on ATM originating from the weaknesses of ADS-B are presented. Lastly, the necessity of ATCOs and other works in the area of ATCO evaluation is discussed.

A. ADS-B

Automatic Dependent Surveillance Broadcast (ADS-B) is the technology that mainly is used in today's airlines for communication between aircraft and the controllers working in ATC. This system is used to improve many factors for both pilots and air traffic controllers, including collision avoidance and situational awareness [5]. While ADS-B improves communication, it also exhibits weaknesses, mainly that ADS-B is not encrypted and that any person with the correct gear can both read and send ADS-B data without any difficulty [6], which calls for improvement.

Before ADS-B, a few different technologies were used, including procedural air traffic control, primary radar surveillance, and secondary radar surveillance. These three methods work in different ways, which made it hard for both pilots and ATCOs to perform their duties. Procedural air traffic control is a very human dependent system because the pilot reports their position to the ATCOs with their voice. This system is tedious to use and prone to mistakes. Primary radar surveillance is less dependant on humans as it does not require any human input. Primary radar surveillance uses ground bases that calculate the position of aircraft and send it forward to other bases. This may be easier, but causes less accuracy. Secondary radar surveillance is a mixture of procedural air traffic control and primary radar surveillance. Secondary radar surveillance has ground bases that calculate the position, but the aircraft provides the altitude and identification [5].

The difficulty in handling such diverse and distinct systems made apparent the necessity of a standardized method. From this necessity came ADS-B, which was meant to redefine air traffic communication as a whole.

ADS-B stands for Automatic Dependant Surveillance Broadcast. It is automatic because it requires no input from the pilot or any other human work. It is dependent on a working navigation system, most often GPS, for sending its position and velocity. Surveillance comes from the fact that the aircraft is providing information to facilities that require it. Finally, it broadcasts the information to other aircraft and ground stations every half-second [7]. Another big difference between ADS-B and older systems is the accuracy of the system. At 60 nautical miles (≈111 km) ADS-B has a positional accuracy of 20 m, which is fifteen times better than radar technology. If the

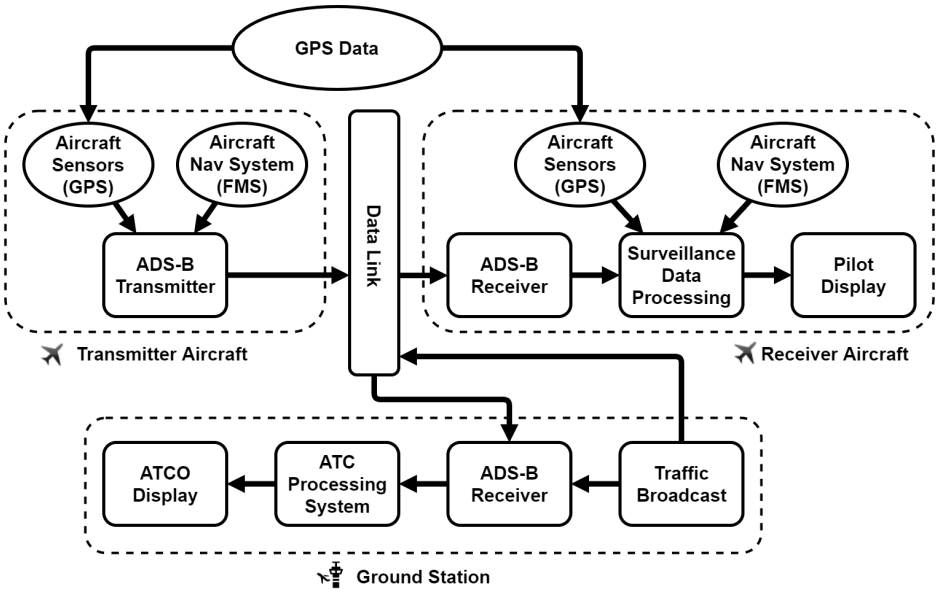

Fig. 1: Components of ADS-B (derived from [5]).

distance increases, the positional accuracy for ADS-B would not decrease unlike to radar technology. Radar technology has decreases in positional accuracy every kilometer [8].

1) Workings: For ADS-B to work, two things need to exist: a way to send information and a way to receive information. The standard is to have two operations: ADS-B OUT and ADS-B IN. The OUT component takes information from the GPS and other equipment and broadcasts the information to everyone near the aircraft. Other aircraft will receive the signal with their ADS-B IN, which will then display the information to the pilots. There are also ground stations that receive the signal. The ground stations will take the signal, process the information and send it to ATC, which then displays the information. Meanwhile, the ground station will broadcast the signal for further spread. Important to apprehend is that ADS-B OUT and ADS-B IN can work separately, meaning ADS-B OUT and ADS-B IN are not dependant on each other to send or receive data [5]. A generalized image of the structure of communication in ADS-B is provided in Figure 1.

The ADS-B broadcast includes information such as the flight identification, which is a flight number or call sign and the ICAO 24-bit Aircraft Address, which is a unique code for each aircraft. The broadcast also contains the aircraft's latitude and longitude, the accuracy of the position, and the rate of either climb or descent, [7]. This information will be shown graphically for both pilots and ATCOs.

For the messages to be transmitted, a form of a data link is required. For ADS-B, there are mainly two different types of data links to use. The more popular one of these two sends on the 1090 MHz link and uses the already in-place Mode S transponders. This feature for Mode S is called Extended Squitter, and altogether this method is called 1090ES. The other method is called Universal Access Transceiver (UAT) which was created for ADS-B and uses the 978 MHz frequency. When using this method it is possible to get 1 Mbps in bandwidth. Determining the best data link out of the two is hard, 1090ES is more common since it can use Mode S, which many aircraft already have installed, while UAT needs newer hardware, therefore most people prefer 1090ES [9].

When using Mode S, the message is either 56 or 112 bits long compared to 1090ES which always is 112 bits long. In these 112 bits, the first five bits define the type of message and for 1090ES it is set to 17 in binary. Afterward, there are three bits for the communication capability for the transponder. Next is 24 bits for the ICAO aircraft address, followed by 56 bits that contain the rest of the data. Lastly, there are 24 bits of parity check to detect transmission errors. These 112 bits are what make ADS-B special, and weak. This is because the transmitter has a function that allows it to control who will see the message and does not use a keep-alive function [5].

B. Attacks on ATM

Attackers may choose many different methods of disturbing the communication of ATM, such as Distributed Denial of Service attacks (DDoS) and radio jamming, but this paper focuses on attacks originating from the weaknesses of ADS-B. The main problems of ADS-B stem from its lack of security mechanisms, in the form of:

- lack of authentication codes to protect against impersonating aircraft.
- lack of message encryption for protection against eavesdropping.
- lack of entity authentication for protection against man-in-the-middle attacks [3].

Summarizing these weaknesses of ADS-B, one may categorize three different forms of attacks on ATM through ADS-B.

- Impersonating aircraft with a radio transmitter transmitting ADS-B data.
- Eavesdropping on ADS-B data.
- Modifying existing ADS-B data.

The structure of ADS-B communication is shown in Figure 1. An attacker may impersonate as an aircraft if the attacker manages to send data both to the ground station and to receiving aircraft, according to Figure 1. Schäfer et al discusses different kinds of active attacks on ADS-B. In the paper, the writers discuss an attack associated with impersonating an aircraft which they called *Ghost Aircraft Injection* and this attack is based on fake message injection. The ghost aircraft must have realistic properties of altitude, speed, and position to be indistinguishable from real aircraft. This attack could confuse ATCOs working on the ground stations [10]. It is not hard to give the attacker realistic properties of the ghost aircraft because there is no encryption on real ADS-B data. The attacker may collect data from a long period of time to improve the properties of the ghost aircraft and increase its indistinguishability.

Schäfer et al also discuss attacks regarding modifying existing ADS-B data. *Virtual Trajectory Modification* is an attack that aims to modify the trajectory of an existing aircraft [10]. The authors assert this can be done in two ways. Firstly, combining the deletion and injection of positional data to remove the data and replace it with modified positional data. The latter way modifies the positional data directly in the air. If the takeover is smooth, the attack might remain undetected and could make ATCOs give out incorrect directions [10].

C. Preparing Future ATCOs

1) The necessity of ATCOs: For an aircraft to fly safely, there is a demand for competent ATCOs capable of directing aircraft from takeoff to their final destination [11]. ATCO students will learn through multiple different simulation environments. For example, students in Malmö at Entry Point North ATS Academy experience simulations in areas such as air traffic service, air traffic rules, flight navigation, meteorology, and flight-English [12]. Examples of communication between ATCOs and pilots consist of departure clearance, takeoff clearance, and landing clearance [13]. We believe it is of great importance to give ATCOs a simulator capable of handling cyberattacks to prepare for real scenarios.

2) Studies in the area of ATCO evaluation: Brookings et al have previously studied the responses of ATCOs during

changes in workload from a psychophysiological perspective in 1996 [14]. In the essence of ADS-B, the paper above is not very relevant as ADS-B was never discussed. We believe our paper and our modified simulator could improve current ATCOs and ATCO students and support future investigations in the area of ATCOs response to cyberattacks.

III. Choosing a simulator

Testing and comparisons between simulators had to be done to find the most suitable simulator for modification. We want to modify the greatest ATC simulator we can find and we want the simulator to be easy to deploy for simulation training. By choosing to modify the best suitable simulator, we give ourselves the best foundation for a successful modification in a successful simulator. Today there are multiple free ATC simulators available, but not all simulators are suitable for modification, due to not not being open-sourced. An example of a non open-sourced simulator is ATC-SIM, founded in 2007 [15]. OpenScope Air Traffic Control Simulator (openScope) and BlueSky Open Air Traffic Simulator (BlueSky) are two examples of simulators that are open-sourced and available on GitHub. These are our two finalists as we believe these are the only open-sourced simulators available and we will now compare them further. To have the ability to modify a simulator, we need access to the source code running the simulator and also the right to modify said code, hence the need for an open-sourced simulator.

A. OpenScope

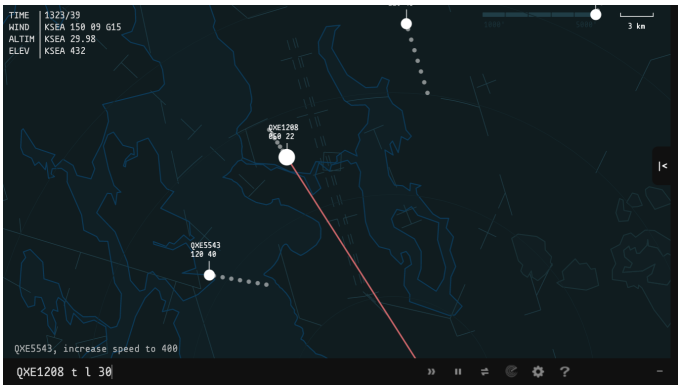

Fig. 2: Gameplay footage of openScope ATC Simulator.

The openScope ATC simulator is an open-source ATC simulating software from 2016 [16]. This project was originally founded by Jon Ross under the name ZLSA ATC Simulator in 2014. Later in 2016, Jon Ross met Eric Quinn who saw great potential and he had big visions for the simulator. It was then imported to GitHub and more easily accessible to the public. Since then the simulator has steadily been developed by enthusiasts, both in aviation and programming. As of May 26, 2020, the main contributors of openScope are Eric Quinn and Nate Geslin [17]. Both Eric and Nate have great knowledge in aviation and programming but Eric has the greatest knowledge in aviation and Nate's talents reside in programming.

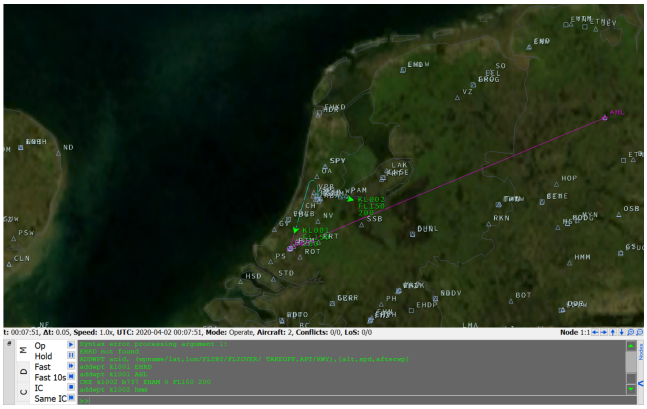

Fig. 3: Gameplay footage of BlueSky ATC Simulator.

OpenScope can be accessed and played online via their website[1], directly in the browser. A screenshot of the gameplay can be seen in Figure 2. The version used for this comparison was v6.16.0, the latest version as of April 5, 2020. From our observations, we felt openScope mainly serves as a practice tool for future ATCOs, while also being able to serve as an analysis tool in the area of air traffic management. 98.4 % of the source code of openScope is written in JavaScript, the last 1.6 % is written in CSS and HTML [17].

B. BlueSky

BlueSky originates from Delft University of Technology (TU Delft) in the faculty of Aerospace Engineering. As of May 26, 2020 the main contributors of BlueSky are Joost Ellerbroek and Jacco Hoekstra, both currently working at TU Delft [18]. This simulator is also open-sourced but its development has not reached the same popularity as openScope. More comparisons between the development processes can be seen in the next section.

BlueSky has very similar functionalities to openScope, such as, controlling the rate of aircraft and commanding them to land or takeoff. On the other hand, BlueSky cannot run in-browser, like openScope. The process of starting the simulator of BlueSky is a much bigger task, as the user needs to download all the files, install Python and a couple of frameworks to run the program in a python environment. According to BlueSky, the purpose of the simulator is to provide everybody who wants to visualize, analyze, and simulate air traffic with a tool to do so without any restrictions [19]. BlueSky may be a hassle to install but 100% of the source code for BlueSky is written in Python which may have benefits for developers [18]. Figure 3 shows a screenshot of the gameplay for BlueSky.

C. Comparison

This section will discuss the differences between the simulators and the reasoning for the chosen simulator.

We want to find the simulator with the greatest popularity and stability together with a simple process of modification. In

[1] https://www.openscope.co

TABLE I: Software comparison between openScope and BlueSky [17], [18].

Comparison		
Simulator	openScope	BlueSky
Main programming language	JavaScript	Python
Installation time in seconds	210	680
Contributors	75	18
Active branches	15	2
Forks	111	102
GitHub stars	304	110
LOC (Main language)	15861	21044
LOC (Total)	400399	34437

Table I we can see where the two simulators differ. Programming languages do not make a major difference but are more of a preferred language for the developer. The second variable in Table I is the time it took for us to install the simulator and run it locally on our computers. This value is heavily dependant on internet and hardware speed as well as the amount of pre-installed programs on the computer running the simulator. For example, to run BlueSky locally, Python and Anaconda had to be installed, which is 0.8 and 5.8 GB respectively. If this was already installed the installation time would be drastically faster. The time to install on a fresh environment was of great interest because we want the chosen simulator to be simple to install for future testers and developers.

The next four variables in Table I, Contributors, Active branches, Forks, and GitHub stars tell the simulators' popularity. All of these values can be seen on the simulators' respective GitHub. Contributors are the number of users that have contributed to writing code for the simulator. While the number of contributors tells us how many have worked on the project, the number of active branches tells us how popular the simulator currently is. Meaning if there are more branches, more people are currently working on the program. Meanwhile, the number of forks tells us the amount of people having worked on the project on their own, since the start of the project. With the number of GitHub stars, it is possible to see how many people are interested in the project. In all these four categories openScope is ahead, which shows that it is a more popular simulator over BlueSky.

The final two rows in Table I show how many lines of code (LOC) each simulator has. While it is possible to see that BlueSky has more lines in its main language compared to openScope, it does not necessarily mean that it is a more complex simulator. If one looks at the number of total lines one can see that openScope is miles ahead in the total amount of code. The reason for the massive amount of LOC in openScope is because the simulator has a lot of external data written with JavaScript Object Notation (JSON). OpenScope has almost 900 files with information about different types of aircraft, airlines, and airports. Out Of these files almost 600 of them are data about different airports, where each airport has anything from a few hundred to a few thousand LOC.

Two clear differences between openScope and BlueSky are installation process and popularity and openScope is the winner in both categories. We want to choose a popular simulator with a simple setup process for programming and testing. BlueSky is more of an open-world ATM simulator where users can go where they want while openScope is locked at a specific airport at a time. OpenScope has a built-in traffic generator which means the game can be played endlessly while BlueSky does not have a traffic generator. Instead, BlueSky has prepared scenarios that the user can play and the user may also add an aircraft at any time of playing. This is a major difference between the simulators and we believe BlueSky gives the user more control of the settings. Giving the user deep control of the settings is of value but is not our primary goal. OpenScope may not give the user the same amount of tools, but the developer can still change anything in the code. To add aircraft with ADS-B attacks, the code still needs to be rewritten greatly in any simulator. With this in mind, we made our decision based on popularity and a simple installation process. Therefore, openScope was selected as our simulator of choice to perform threat modification.

IV. Types of attacks

This section will explain the method that was used for choosing simulator and attack types as well as the programming process.

A. Choosing simulator and attack types

The pre-study of this paper consists mostly of choosing the most suitable attack types to program into the simulator and comparisons between ATC simulators. The chosen attack types were based on what was deemed possible with ADS-B attacks, which itself was based on known weaknesses of ADS-B from literature. If one wants to answer the questions at hand in the future, one should choose the type of ADS-B attack that is the most common or most plausible of happening in a real scenario at that given time. In this paper, the chosen attack types were established from the present weaknesses of ADS-B. The chosen attack types to program in the simulator are stated below.

- Aircraft not responding to commands.
- Aircraft with sudden changes to their position data.
- Aircraft with incorrect altitude and speed data.

To define the most suitable ATC simulator for modification it was first needed to define what makes a simulator great. Two things that were mainly valued were great documentation of code and a simple setup process of the program. At the time of writing this paper, the only open-sourced ATC simulators

available were BlueSky and openScope. One should try to compare all available open-sourced ATC simulators to better find the best simulator for modification. Documentation can often be found on the simulators website or a potential Github site for the simulator. Documentations having a dedicated section for developers or people wanting to contribute to the code is of great value.

Comparing the setup process was accomplished by setting up both simulators on a computer. The faster the process of installation, the better. A common obstacle in the installation process was different types of plugins and additional software needed to run the simulator. Some software can be easy to install and some can be more complex and time-consuming. When selecting the simulator with the simplest setup process, it is important to try the setup process on different operating systems. Both the developers and future controllers testing the simulator benefit from a simpler setup process. During development, computers running both Windows and macOS were available, and both simulators were tested on both operating systems. After a thorough comparison, a single simulator has to be selected for the implementation of the attacks. A wider comparison between the two simulators can be found in section III-C.

B. Implementation

The implementation part of the method will describe how the different attack types were implemented in the chosen simulator openScope. The first thing that was added was a way to differentiate different aircraft, e.g. if it is of attack type ”non-responsive” or attack type ”incorrect altitude and speed”. This was done by giving each aircraft a variable called $attackType$ that will change value between 0 and 3 depending on what type that specific aircraft is (regular, non responsive, changes position, or altering data).

1) Non-responsive: The next step was to implement the attacks, starting with the non-responsive aircraft. For this attack it was wanted to change how the method for giving commands to different aircraft was working, e.g. if the user wants an aircraft to turn or change speed. The desired goal was to make sure that no aircraft with $attackType$ 1 would go beyond this point. This was solved easily with a simple if-statement that makes sure that the $attackType$ is not 1. If $attackType$ is not 1, the given command is executed as normal. If it is 1, the output of the given command will be ”No response” instead of the normal output when a command is successful.

2) Changing position: For the next attack with changing position, the goal was for the aircraft to change its position around the airport seemingly at random. This part of the implementation was made in each aircraft's individual $updatePhysics$ function which normally moves the aircraft multiple times each second. To make an aircraft change position, the code is checking every two seconds for three separate values. The first value to be checked is if the $attackType$ is 2. The second value is if the aircraft's position was in the range of the airport to save some computing power for the rest of the program, as aircraft jumping around outside of the airport area was not desired. The last value that the code looks at was if it had tried to make a jump already during any given second. This is because the developers decided that any given aircraft can only make one jump every other second. Since we do not want all of the aircraft with $attackType$ 2 to jump at the same time, another if-statement was added, this time to make it harder for an aircraft to jump. In this if-statement a random number is generated and if that number is equal to 1, that aircraft will perform a jump. The random number will be generated in different ranges depending on how often the user wants the aircraft to jump. Five intervals with different ranges were chosen and 0 to 120 was the smallest interval and 0 to 60 000 was the greatest interval.

If a jump is to occur the following would happen. A random point in a circle with a certain radius is generated. Next it is decided where this point is in relative to the airports center and finally the aircraft changes its position to this new position.

3) False information: The last attack that was implemented was aircraft with wrong altitude and speed data. This means that a given aircraft will be flying at a specific altitude with a specific speed, but for the user that altitude and speed is something else, i.e. if an aircraft has a speed of 300 knots and the user observes a speed of 400 knots. For the implementation of this attack, each aircraft was given two extra variables; $fakeAltitude$, and $fakeSpeed$. These variables will be assigned a random value at the beginning of the simulation. To get the possible ranges for these variables, we first needed to know what are reasonable values. To figure this out we looked at other aircraft's altitude and speed in the simulator and took the maximum and minimum of each category. This gave us the following intervals. Variable $fakeAltitude$'s interval is between 5000 and $40,000$ feet and this number should also be divisible by 100. It needs to be divisible to keep the similarity for the real values. On the other hand $fakeSpeed$s interval is between 280 and 600 knots and this value has to be divisible by 10 for the same reason as above. Each aircraft will have these variables with assigned values, regardless of what kind of $attackType$ they are, but only aircraft with $attackType$ 3 will ever use this variable.

When the code is showing the user what altitude and speed an aircraft has in the graphical user interface (GUI), it checks if the aircraft has $attackType$ 3. If that is the case, it shows fakeAltitude and fakeSpeed, otherwise, it shows the real altitude and speed.

4) Tracking and logging: The second-to-last thing that had to be implemented in the code was a way for the program to keep track of how a user interacts with different aircraft. The information that is given to the user needs to be relevant as well as easy to work with. This is because the user would want to study the interactions after they used the simulator, the information had to be stored in a downloadable file. Since JavaScript does not support writing to a local file for safety reasons, we had to implement a button that downloads a file with the information. To implement this button, an input of type ”button” was added to the footer. When someone then

presses the button, an event listener will notice the button press and will call a method. In this method, a temporary HTML a tag is created which will get the attributes "download" and the chosen filename. This element will also have an attribute containing the data we want to save. We will then add this temporary element to the document and simulate a button press to download the file. Finally we remove the element from the document again. To keep track of the information, we have a log variable of type String that will have all the information we want to save in the logfile.

5) GUI: The last thing left for implementation was the GUI. We wanted to give the user a way to select how many attacking aircraft there would be and also how to distribute the different attack types. This was done by creating a new menu similar to the already existing settings menu. We achieved this by creating a new button in the footer labeled "Attacks". When someone presses the button the new menu will show up. This was then followed by creating the content of this new menu. There were six different settings we wanted, with the fifth one being the radius of the jumps and the sixth one being able to change the probability for a jump. For each setting a drop-down menu was used. With a new class for these drop-downs, it was now easy to add new settings. For a new setting to be created it needed a description, an event handler that will be called when the value changes, and the different options for that specific setting. Each option needs a few things; the text that will be shown and what value that option will represent. To distribute the attack types we used a weighing scale between zero to five, e.g. if the value of non-responsive is one and the value of changing position is two, there will be twice as many aircraft changing positions as there will be non-responsive ones.

In the class which controls the logic behind the game, the event handlers were created. Each event handler changes a few variables that keep track of the weight, radius, or probability. With a now working weighting system, the next thing was to choose which aircraft is which attack type. This is done by taking each weight and adding them together. Each attack will then have a percentage chosen by taking its weight and dividing by the total weight. This number is then multiplied by 100 to get a value between 0 and 100. A random number is then created, between 0 and

$$\frac{100}{\text{selected ratio between attack aircraft and total aircraft}}.$$

If the random number is below 100, it will be an attack aircraft. If it is not, it will be a normal aircraft. The next step is to decide what attack type it will be, if the random number is below 100. If this number is below the percentage for non-responsive it will be of that type. If it is above the mentioned percentage, but below the percentage for non-responsive added by the percentage for changing position aircraft, it will be of type changing position. If it is neither of these it will be of type incorrect altitude and speed. Now, a chosen portion of aircraft will be assigned to attacking aircraft and the attack types will be distributed according to their weights.

V. Attack implementation

The following section presents the results from the development of the simulator.

A. Implementation

A new symbol has been connected to the options menu and by pressing it, the user will be greeted by a graphical user interface (GUI) for deployments of ADS-B attacks. This GUI contains all eligible options and preferences for the simulation of ADS-B attacks. Connected to this settings menu is also a manual informing the user how to operate the deployment of ADS-B attacks into the simulator. This GUI can been seen in Figure 4.

The user will be able to change six different parameters in the GUI. The first one called "Percentage of Aircraft Affected" is a drop-down menu where the user can choose the percentage of aircraft affected by ADS-B attacks. The values are static at 90%, 50%, 20%, 5% and 0%. The next three parameters are all connected to the weighting of which types of attacks the user wants to deploy. Each attack type has its weighting value from zero to five. If all of the numbers are of the same value, the spread of attack types will strive to be evenly distributed. Let's say attack type 1 is given weighting value one, attack type 2 is given weighting value two, and attack type 3 is given weighting value three. Then the most common attack types, in order, will be, attack type 3, attack type 2, and lastly attack type 1. "Probability of jumps" will set the jumping probability of position altering aircraft. This value can be set to very low, low, medium, high, and very high. The last parameter is called "Distance of jumps". This radius can be set to three values, small, moderate, and large. This value will set the jumping distance of position altering aircraft. At the bottom part of Figure 4 the simulator tells the user how to save his or her actions from the current session in a logfile by pressing the button "Download logfile".

When the "Download logfile" button is pressed, the user will download a text file called "log.txt" containing a list of all commands the user has executed. This text file contains timestamp, command type, command value, which aircraft it was called to and what attack type the aircraft was. If the aircraft is not affected by ADS-B attacks, the type will be Regular. The four aircraft types are, "Jumping", "Non-responisve", "False Information" and "Regular".

B. Attack types

In this section, the different attack types are presented. Aircraft affected by ADS-B attacks will look the same as aircraft not affected by ADS-B attacks in the simulator. The only way to distinguish real aircraft from aircraft affected by ADS-B attacks is through the actions of the aircraft.

If the user enters a command to a non-listening aircraft, the simulator will not do anything with the command as seen in Figure 5. Commanding other aircraft will give a response but with this aircraft type no response is given. The aircraft will not obey any commands given to it from the user and will only roam around the airport waiting for instructions. The second

Fig. 4: A closer look at the Graphical User Interface for deploying attacks.

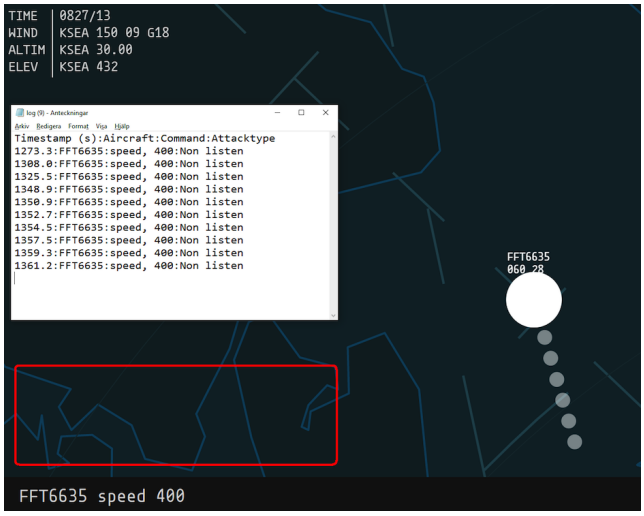

Fig. 5: Example of an aircraft not responding to commands. As the red square is empty, no response is given even when the aircraft is contacted multiple times. The log file proves that commands have been given to the aircraft.

attack type is the positional altering aircraft. When positional altering aircraft is enabled, the user will randomly observe positional jumps of some aircraft. The interval between each jump and the position are both random. In Figure 6, an example of a jump is shown. The third and last attack type is aircraft with false information about altitude and speed. These aircraft will always display a static value on altitude and speed, even when they change speed and altitude, the displayed value will always stay the same. An aircraft with false information

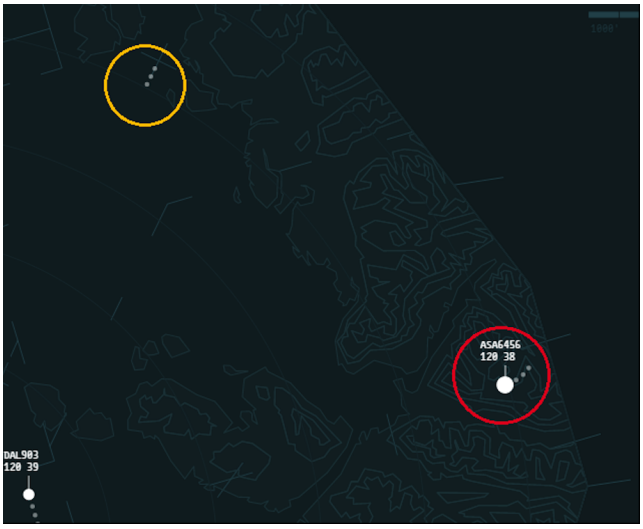

Fig. 6: Example of an aircraft changing position with highlighted areas. The yellow circle shows previous position and the red circle shows current position.

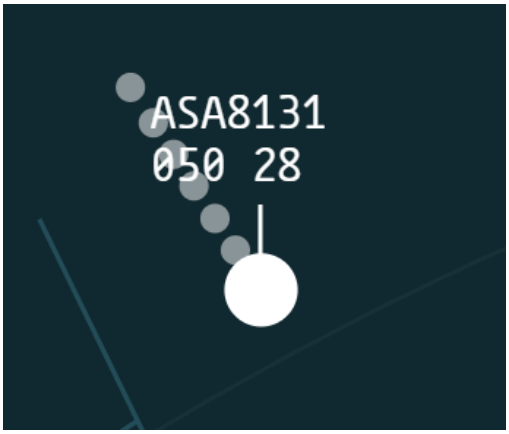

Fig. 7: Example of an aircraft displaying false altitude. The aircraft has just entered the airport area and the left value shows altitude of 5000 feet.

will be assigned a set value between strict ranges. The altitude ranges from 5000 (displayed as 50) and 40 000 (displayed as 400) feet and the speed ranges from 280 (displayed as 28) to 600 (displayed as 60) knots. Two numbers can be found above all aircraft in the simulator. The left number is the altitude and the right number is the speed. An example of an aircraft showing false information is shown Figure 7. Since all aircraft enter the airport area at around 12 000 feet, and the aircraft in Figure 7 is at 5000 feet while entering the airport area, it is clear that this aircraft is showing false information.

VI. Discussion

We discuss the results in Section VI-A, the method in Section VI-B and initial testing with a real ATCO in Section VI-C.

A. Results

When we look at our result, it is more or less what one would expect. There are still a few things that need some clarification, for example how one would use this program. It can be used in different ways, e.g., as an extension to the original openScope or as a tool to learn more about how different attacks to ATM would look like. But the greatest use of the simulator would be to see how ATCO reacts to these attacks. For this kind of use, we let the user play the game for half an hour without any attacks and save the logfile. Afterward, let them play for another half an hour with attacks and save another logfile. It would then be possible to compare their responses, with and without attacks. Then compare how many times the user normally calls on a specific aircraft and compare that value to how many times the user calls on an attack aircraft, do they ignore the attack aircraft or do they put a lot of effort on them?

If we try to compare our results to the weaknesses to ADS-B as described in Section II-B, we can see that we are impersonating aircraft with the attack type of altering position. This is by pretending to be a real aircraft until we suddenly change our position in a random pattern. Another way we are impersonating aircraft is by creating spoofed aircraft. We are also modifying the already existing ADS-B data by giving the user false altitude and speed information. The only way we are not abusing the weaknesses of ADS-B in this result is the possibility of eavesdropping. This is a real threat, that is one of the easiest to do in reality. The only reason for this threat to not be implemented is that there is no way for the ATCO to know if someone has eavesdropped on the communication and thus there is no need for it in the application. One could argument that this kind of attack is also implemented in the form of the logfile, as the logfile contains all communication between the user and pilots. This information can be seen as a third party secretly gathering the communication information.

B. Method

For the time that we had at our disposal to complete the project we believe we got a good result. However, even if it is a good result it does not mean it could not be better. Many things could be added for better visualization of the attacks. One thing that could be improved is how we add attack aircraft. The way it works now is functional, but if a great number of attacking aircraft is used there will not be many regular aircraft left. The way it works is that each airport has its spawning of aircraft algorithm and when we ”create” new attack aircraft we simply change the aircraft that would have been present, regardless of our input. If we instead added new aircraft to the spawning algorithm we could see more variance in the spawning. We would also be able to do real flooding of non-responsive aircraft, for example by adding fifty non-responsive aircraft to the already thirty or so regular aircraft. This would be a much harder task for an ATCO and it would probably be a more realistic case, since adding just a few fake aircraft that do not respond can be easier to figure out than adding many new aircraft. This is the biggest fault in our method and if the implementation would be done again, this would be changed. If we were to do it again with this change to the method a new option in the attacks menu would be needed too. This option would let the user choose the exact amount of attacking aircraft.

The chosen attack types programmed in the simulator may be further from reality than expected. Going forward, we will have more discussions with ATCOs to validate the chosen attack types possibility in a real scenario.

Another thing that could have been improved in the method is to add another button. When pressed, this new button would highlight all the attack aircraft in a different color from the regular ones. With this new feature it could create a new test for the user. In this test the user would get to use the program for a while, after this said time they could guess which aircraft are attacking aircraft and which are not. With a press of a button the user would see which are attack aircraft and which are not. This button will help to determine how good the ATCOs are at recognizing the attack aircraft in another way, compared to just a logfile.

Yet another thing that could be implemented is additional types of attack. For example, adding an aircraft that would stop contact with other surrounding aircraft. This would simulate a Denial of Service attack. With this kind of attack, the user would not be able to do anything but would have to find a different solution to communicate with the aircraft. If this type of attack or other types would be implemented the ATCO that uses the program would get an even wider perception of the problems with ADS-B. It would also be interesting to see how they would handle this kind of problem.

These are not all the things we could have executed better. For example, one of the smaller things that could be improved is how we calculate the new position of the aircraft when the attack type is of altering position. Currently we are calculating a vector in which the aircraft will move in before a jump is executed. This method is correct mathematically, but to save on computing power we could simply just set the position to the new position instead of calculating a vector. Another improvement in the simulator could be made in making fake data more realistic. For instance, instead of giving each aircraft a static value for the fake altitude and fake speed, these values will change with real values, e.g. if the altitude goes down by maybe 2000 feet the fake altitude would maybe go down by 1500 feet. If the fake values would be more flexible and not static it would be harder to detect these aircraft. This could lead to a delay in the realization of an attacking aircraft.

The last thing that could change has to do with logging. Currently the information that is saved is fixed, but maybe some users would need different information than others. This problem could be solved by making a small menu where the user could specify what data to be saved in the logfile.

C. Testing on ATCO

We have been given assistance in the development process from Ms. Supathida Boonsong who has worked in ATM. She was the first ATCO who tested our simulator and she

contributed with great feedback. She pointed out flaws and improvement areas for the simulator as well as giving us information about the ATCO security routine for inadequate aircraft. With her help we now understand more how an ATCO works. She also stated that the simulator was useful for educating ATCOs.

VII. Conclusion and future work

A. Conclusion

The purpose of this work is to help prepare ATCOs to cyberattacks and to further establish the security flaws of ATM originating from ADS-B. The following two research questions were then created: *How could a known ATM simulator be modified to track ATCOs responses to cyberattacks?* and *Can a modified ATM simulator expose the weaknesses of ADS-B?* Conclusions and answers to these questions can be seen in the following sections.

By developing this simulator and by sending it to the public there is now a possibility for real ATCOs to practice against cyberattacks. Though one cannot know for sure how well this will help real ATCOs until it has been tested with them. To better determine how helpful the simulator is, it would have been wise to gather feedback from many real ATCOs during and after the development of the modified simulator.

Our second goal, "further establish the security flaws on ATM originating from ADS-B", has only been fulfilled to a certain extent. The paper does not introduce any new areas in ADS-B where security is faltering. This work is more of a reminder that ADS-B is lacking in security by bringing up papers proving the security flaws.

The first research question is best explained in Section IV, as this is almost the entirety of our work. Of course, there are multiple ways to answer this question but our way is explained most thoroughly in Section IV. More simply, search for an open-source ATM simulator, research for plausible ADS-B attacks, and implement them into the simulator by rewriting the code running the simulator. Then, add a way of saving the actions of the user playing the simulator in a downloadable logfile. Make sure the logfile contains the commands called to the aircraft what what attack type the aircraft was.

We deem that the simulator is a great tool to showcase the weaknesses of ADS-B. The attack types inside the simulator are not fictional or made up, instead, the attack types are based on the real weaknesses of ADS-B from the works of Costin et. al and Schäfer et. al [3], [10]. To make sure the user understands the weaknesses of ADS-B, the simulator could have told the user that the attacks in the simulator are based on real studies and not made up.

Future systems such as L-band Digital Aeronautical Communications System (LDACS) can fix the security issues with ADS-B. However, their development, standardization and deployment is likely to take decades. Meanwhile, we strive to make both ADS-B and aviation as safe as possible and encourage to take flaws of ADS-B seriously. We want attention to be brought upon our modified simulator and its use as a tool to help prepare ATCOs for real cyberattacks.

B. Future work

To further help ATCOs with the preparation of cyberattacks with ADS-B, more studies need to be done on real ATCOs with the help of our simulator. It can also be of great value to further explore the programming possibilities of more attack types, greater control of the attack types, and better observation of the users' actions against it.

References

[1] Google, "Google Trends on Cyber security," https://trends.google.com/trends/explore?date=allgeo=USq=cybersecurity, 03 2020, (Accessed on 31/03/2020).

[2] ——, "Google Trends on Virtual Private Networks," https://trends.google.com/trends/explore?date=allq=%2Fm%2F012t0g, 03 2020, (Accessed on 31/03/2020).

[3] A. Costin and A. Francillon, "Ghost in the Air (Traffic): On insecurity of ADS-B protocol and practical attacks on ADS-B devices," in *BLACKHAT 2012, July 21-26, 2012, Las Vegas, NV, USA*, Las Vegas, UNITED STATES, 07 2012. [Online]. Available: http://www.eurecom.fr/publication/3788

[4] A. Gurtov, T. Polishchuk, and M. Wernberg, "Controller-Pilot Data Link Communication Security," *Sensors (Basel)*, vol. 18, no. 5, 05 2018.

[5] D. McCallie, J. Butts, and R. Mills, "Security analysis of theADS-B implementation in the next generation air transportation system," *International Journal of Critical Infrastructure Protection*, vol. 4, no. 2, pp. 78 – 87, 2011. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S1874548211000229

[6] Federal Aviation Admission, "Frequently Asked Questions (faqs)," https://www.faa.gov/nextgen/programs/adsb/faq/#o3, 08 2019, (Accessed on 30/03/2020).

[7] Airservices Australia, "How ADS-B works," http://www.airservicesaustralia.com/projects/ads-b/how-ads-b-works/, 09 2015, (Accessed on 02/04/2020).

[8] H. A. L. Purton and S. Alam, "Identification of ADS-B System Vulnerabilities and Threats," Australasian Transport Research Forum. 35 Stirling Highway (M087) Crawley, (Perth) Western Australia 6009: Patrec, 10 2010.

[9] M. Strohmeier, V. Lenders, and I. Martinovic, "On the Security of the Automatic Dependent Surveillance-Broadcast Protocol," *IEEE Communications Surveys Tutorials*, vol. 17, no. 2, pp. 1066–1087, Secondquarter 2015.

[10] M. Schäfer, V. Lenders, and I. Martinovic, "Experimental Analysis of Attacks on Next Generation Air Traffic Communication," in *Applied Cryptography and Network Security*, M. Jacobson, M. Locasto, P. Mohassel, and R. Safavi Naini, Eds. Berlin, Heidelberg: Springer Berlin Heidelberg, 2013, pp. 253–271.

[11] LFV, "LFV on Air Traffic Controller," https://www.lfv.se/en/services/airport-services/air-traffic-control, 03 2020, (Accessed on 18/3/2020).

[12] ——, "LFV on Air Traffic Controller education," http://www.lfv.se/bli-flygledare/utbildning, 03 2020, (Accessed on 18/3/2020).

[13] OSM, "OSM Aviation Academy on ATC-pilot communication," https://www.osmaviationacademy.com/blog/learn-to-talk-like-a-pilot-part-3-clearances, 03 2020, (Accessed on 18/3/2020).

[14] J. B. Brookings, G. F. Wilson, and C. R. Swain, "Psychophysiological responses to changes in workload during simulated air traffic control," *Biological Psychology*, vol. 42, no. 3, pp. 361 – 377, 1996, Psychophysiology of Workload. [Online]. Available: http://www.sciencedirect.com/science/article/pii/0301051195051678

[15] ATC-Sim, "Website of atc-sim.com," https://atc-sim.com/, 05 2020, (Accessed on 14/05/2020).

[16] openScope, "Homepage of openScope Air Traffic Management Simulator," https://www.openscope.co, 03 2020, (Accessed on 27/3/2020).

[17] ——, "openScope on GitHub," https://github.com/openscope, 03 2020, (Accessed on 27/3/2020).

[18] BlueSky, "Bluesky on GitHub," https://github.com/bluesky/bluesky, 03 2020, (Accessed on 27/3/2020).

[19] BlueSky, "Homepage of BlueSky Open Air Traffic Simulator," http://homepage.tudelft.nl/7p97s/BlueSky/, 03 2020, (Accessed on 27/3/2020).