IEEE Access
Multidisciplinary : Rapid Review : Open Access Journal

Received December 26, 2021, accepted January 26, 2022, date of publication February 10, 2022, date of current version March 2, 2022.

Digital Object Identifier 10.1109/ACCESS.2022.3150840

GDL90fuzz: Fuzzing - GDL-90 Data Interface Specification Within Aviation Software and Avionics Devices—A Cybersecurity Pentesting Perspective

HANNU TURTIAINEN, ANDREI COSTIN, SYED KHANDKER, AND TIMO HÄMÄLÄINEN
Faculty of Information Technology, University of Jyväskylä, FI-40014 Jyväskylä, Finland

Corresponding author: Hannu Turtiainen (hannu.ht.turtiainen@jyu.fi)

This work was supported in part by the Engage Consortium's Knowledge Transfer Network (KTN) funding for project *"Engage—204—Proof-of-Concept: Practical, Flexible, Affordable Pentesting Platform for ATM/Avionics Cybersecurity"* Single European Sky ATM Research (SESAR) Joint Undertaking under the European Union's Horizon 2020 Research and Innovation Program under Grant 783287, in part by the Finnish Grid and Cloud Infrastructure (FGCI) Persistent Identifier under Grant urn:nbn:fi:research-infras-2016072533, in part by the Decision of the Research Dean on Research funding within the Faculty of Information Technology of the University of Jyväskylä, and in part by the Finnish Cultural Foundation under Grant 00211119.

ABSTRACT As the core technology of next-generation air transportation systems, the Automatic Dependent Surveillance-Broadcast (ADS-B) is becoming very popular. However, many (if not most) ADS-B devices and implementations support and rely on Garmin's Datalink 90 (GDL-90) protocol for data exchange and encapsulation. This makes it essential to investigate the integrity of the GDL-90 protocol especially against attacks on the core subsystem availability, such as denial-of-service (DoS), which pose high risks to safety-critical and mission-critical systems such as in avionics and aerospace. In this paper, we consider GDL-90 protocol fuzzing options and demonstrate practical DoS attacks on popular electronic flight bag (EFB) software operating on mobile devices. Then we present our own specially configured avionics pentesting platform and the GDL-90 protocol. We captured legitimate traffic from ADS-B avionics devices. We ran our samples through the state-of-the-art fuzzing platform American Fuzzy Lop (AFL) and fed the AFL's output to EFB apps and the GDL-90 decoding software via the network in the same manner as legitimate GDL-90 traffic would be sent from ADS-B and other avionics devices. The results showed worrying and critical lack of security in many EFB applications where the security is directly related to the aircraft's safe navigation. Out of the 16 tested configurations, our avionics pentesting platform managed to crash or otherwise impact 9 (56%). The observed problems manifested as crashes, hangs, and abnormal behaviors of the EFB apps and GDL-90 decoders during the fuzzing test. Our developed and proposed systematic pentesting methodology for avionics devices, protocols, and software can be used to discover and report vulnerabilities as early as possible.

INDEX TERMS GDL-90, ADS-B, attacks, cybersecurity, pentesting, resiliency, DoS, aviation, avionics.

I. INTRODUCTION

In the United States aviation sector, the Federal Aviation Administration (FAA) is pushing a shift from secondary surveillance radar (SSR) interrogations to the more modern Automatic Dependent Surveillance-Broadcast (ADS-B) technology in air traffic control. As of January 2020, all aircraft operating in the continental United States are required to use ADS-B [1]. European aviation is following suit – the gradual shift to mandatory ADS-B broadcasting already started in June 2020 [2]. ADS-B offers many benefits over SSR, such as enhanced and fully automatic situational awareness of all aircraft and air traffic control (ATC) in the vicinity, increased system efficiency by eliminating interrogation processes, and cost-effective implementation. Moreover, FAA and its stakeholders are actively experimenting with ADS-B for

The associate editor coordinating the review of this manuscript and approving it for publication was Yang Liu.

commercial space transportation applications [3]. Due to ADS-B's efficiency, light weight, and cost-efficient features, it is gaining popularity among all types of users. Using a portable ADS-B transceiver (e.g., SkyEcho2, Sentry, and echoUAT) as a mobile cockpit solution is very trendy nowadays, especially in the general aviation sector. Such portable ADS-B devices provide services through and electronic flight bag (EFB) application hosted on a mobile tablet or smartphone. ADS-B devices (e.g., SkyEcho2, Sentry, and echoUAT) exchange data mainly using the Garmin DataLink 90 (GDL-90) protocol, one of the de facto standard technologies that are leading in the avionics industry. GDL-90 is also used in many integrated flight deck (IFD) systems and electronic flight instrumentation systems (EFISs) such as Garmin's G1000, Avidyne's IFD440/540, and EX5000, as well as in many mobile cockpit devices and EFB applications (such as AvPlan, Naviator, and Airmate). Due to the wide use of GDL-90, any potential vulnerability in it poses elevated cybersecurity risks to avionics systems as well as safety risks to the passengers and crew lives. Researchers have reported several types of security threats involving ADS-B, such as ghost aircraft, aircraft disappearance, denial-of-service (DoS) [4]–[6]. However, protocol fuzzing in mobile cockpit systems has not been thoroughly investigated yet, which has motivated us to conduct this study. This study is important as it systematically addresses the discovery of potential bugs and cybersecurity vulnerabilities within GDL-90 implementations. Our main contributions with this work are as follows.

1) To the best of our knowledge, we are the first to propose, develop, and execute a systematic fuzzing platform and experiments aimed specifically at the GDL-90 protocol (although our method is easily extensible to more avionics and aerospace data-link protocols).
2) We are the first to discover and report safety-critical DoS vulnerabilities in a handful of the most popular aviation apps and mobile EFBs as a result from fuzzing the GDL-90 inputs.

The rest of this article is organized as follows. Different fuzzing aspects are discussed in Section II. In Section III, we introduce our attack strategy. We present the results in Section IV. We discuss related works in Section V. Finally, in Section VI, we discuss possible workarounds and future work as we conclude this paper.

II. BACKGROUND

In this section we briefly present background technologies and techniques used in our experiments.

A. FUZZING

Fuzzing (or fuzz testing) is an automated software testing method for finding implementation and input sanitization bugs by using intentionally malformed or randomized inputs. It was originally developed by Professor Barton Miller and his team of students at the University of Wisconsin Madison in 1989 [7]. With fuzzing, a generator is used to create random and semi-random (known to be dangerous) data usually sampled from real inputs. Such data are inputed in to the software being tested, and the software's behaviour is observed. Fuzzing is based on the premise that bugs exist in every program and therefore, a consistent and systematic approach will eventually cover them [8]. Fuzzing is a blind testing technique with caveats, such as the possibility of missed program paths due to the random nature of the input mutations [9]. In our experiments, we targeted the GDL-90 protocol, which means that we used protocol fuzzing by forging packets with a real protocol-like format but with some parts malformed. (This topic will be discussed further in Section III-D).

In this study, we used the American Fuzzy Lop (AFL) as our core fuzzing toolset. AFL is a security-oriented greybox fuzzer originally developed by Michal Zalewski [10]. It is a proven, easy-to-use, stable, and effective fuzzer that utilizes performance optimizations to decrease unnecessary runtime. It uses an instrumentation-guided genetic algorithm to fuzz the software being tested with brute force. In essence, AFL takes the user-supplied sample test cases one by one,trims them, and mutates the trimmed versions with traditional fuzzing strategies. The behavior of the software being tested is recorded, and interesting test cases are recorded for further use and for runtime modifications of the fuzzer [9]. AFL is currently maintained by Google Open Source and is licensed with Apache License 2.0 [9], [11].

Flag Byte	Message ID	Message Data	CRC	Flag Byte

FIGURE 1. GDL-90 message format.

B. GDL-90 PROTOCOL

The Garmin DataLink 90 (GDL-90) format is supported by many aviation hardware and software (see Table 3). It is described in the RTCA DO-267A standard as a messaging structure based on asynchronous high-level data link control (HDLC), with some modifications to better suit avionics data interfaces [12], [13]. The basic GDL-90 message format is presented in Figure 1.

The message starts with a Flag Byte (0 × 7E), followed by a one-byte Message ID, which specifies the type of message being transmitted. The message type sets the message data content and length. All the message definitions are listed in Table 1.

A two-byte frame check sequence (16-bit CRC, LSB first) is calculated for the data and appended to the message, and the message ends with another flag byte. If a flag byte (0 × 7E) or a control-escape character (CEC, 0 × 7D) is present in the original message, the message byte is XOR'd with 0 × 20, and a CEC is prefixed to it. Thus, the integrity of the message is preserved. The receiving end checks the incoming traffic for the Flag Bytes and captures the data between them. The captured data are inspected for CECs. If a CEC is found, it is

TABLE 1. GDL-90 message IDs.

Message ID	Message Name
0	Heartbeat
2	Initialization
7	Uplink Data
9	Height Above Terrain
10	Ownship Report
10	Ownship Geometric Altitude
20	Traffic Report
30	Basic Report
31	Long Report

discarded, and the byte after it is XOR'd again to return its old form properly. The CRC for the message data part of the full GDL-90 message is calculated and verified. If it is deemed valid, the message is ready for use. GDL-90 devices in operation transmit a heartbeat message once every second, followed by an ownship report. In between these "pulses" other messages such as traffic reports can be transmitted. In our experiments, we focused on three types of messages:

- Heartbeat messages,
- Traffic reports; and
- Ownship reports

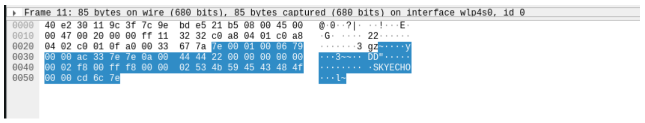
A heartbeat message is used for the devices to indicate that they are operational and to submit information about their status. Two status bytes in the message tell information about the transmitter in Boolean fashion. This information includes "battery low," "Global Positioning System (GPS) fix," "maintenance requirement," etc. flags. A timestamp is also present in the message after the status bytes.

Traffic reports are at the output in each second for each proximate target. GDL-90 expects at least 32 simultaneous targets to be handled, but more can be processed depending on the uplink configurations and the interface baud rate. Traffic report data use 27 bytes to represent each needed attribute. Table 2 shows the fields of the traffic report data in order.

An ownship report message follows the traffic report format. It is always in the output even without a proper GPS fix. It broadcasts the transmitter information to the network.

C. GDL-90 PROTOCOL EXTENSIONS

Some vendors have their own interpretation of the protocol outside of the Garmin standard. For example, Uavionix's SkyEcho2 mainly uses the standard messaging types, but it outputs its ownship message with the message type code 101. On the other hand, ForeFlight's Sentry extends the protocol and does not communicate with the standard message types. Sentry transmits messages with IDs 37 and 38, which are longer than the standard heartbeat, ownship, and traffic messages and most likely contain multiple message types



```
▸ Frame 11: 85 bytes on wire (680 bits), 85 bytes captured (680 bits) on interface wlp4s0, id 0
0000  40 e2 30 11 9c 3f 7c 9e  bd e5 21 b5 08 00 45 00   @·0··?|· ··!···E·
0010  00 47 00 20 00 00 ff 11  32 32 c0 a8 04 01 c0 a8   ·G· ···· 22······
0020  04 02 c0 01 0f a0 00 33  67 7a 7e 00 01 00 06 79   ·······3 gz~····y
0030  00 00 ac 33 7e 7e 0a 00  44 44 22 00 00 00 00 00   ···3~~·· DD"·····
0040  00 02 f8 00 ff f8 00 00  02 53 4b 59 45 43 48 4f   ········ ·SKYECHO
0050  00 00 cd 6c 7e                                     ···l~
```

FIGURE 2. Heartbeat messages of SkyEcho2 proprietary GDL-90 extension as captured and decoded by wireshark software.

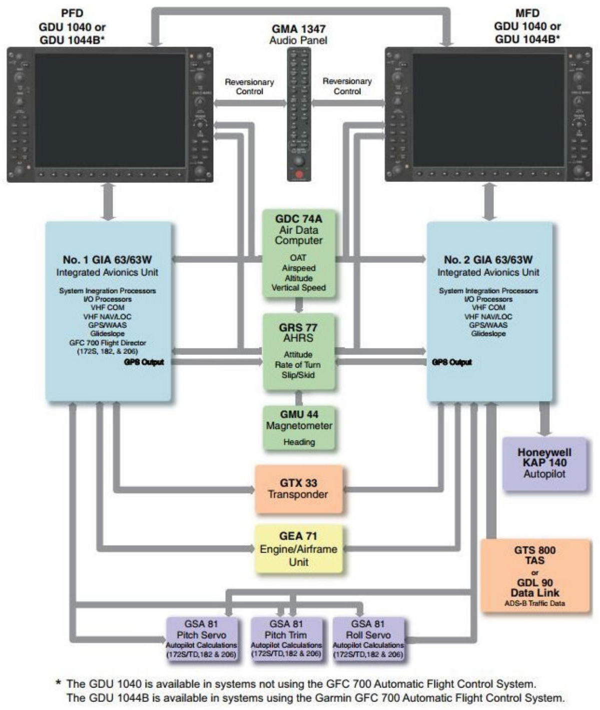


FIGURE 3. System diagram of Garmin G1000 EFIS/IFD [15]. Note the GDL-90 inputs going into No. 2 GIA 63/63W that, in turn, controls the auto-pilot Honeywell KAP 140 [14].

in a single packet. The ForeFlight EFB supports both devices. It broadcasts messages to the network. When the app is accepting traffic, it sends *"i-want-to-play-ffm-udp"*; and when it goes to sleep it sends *"i-cannot-play-ffm-udp."* It also identifies itself to the network by broadcasting *"App: ForeFlight, GDL90: port:4000"* messages. For our experiments, we did not delve deeper into the ForeFlight protocol as it was not necessary. We were able to capture, modify, resend, and receive Sentry packets just like with the other devices. Thus, the integration with AFL was quite straightforward. Figure 2 shows a Skyecho-ecoded heartbeat packet in Wireshark.

Figure 3 depicts the system diagram of Garmin G1000 – a real-world EFIS/IFD/avionics system. It is important to note that GDL-90 inputs go to the GIA 63/63W avionics unit that is also *directly controlling the auto-pilot systems* such as Bendix/King KAP-140 [14]. Therefore, any GDL-90 vulnerabilities present within the avionics units *have a potential direct effect on the auto-pilot systems*. Therefore, it is important to discover such GDL-90 (and other data-link protocol) vulnerabilities as fast and as efficiently as possible, for example, using our approach and results.

TABLE 2. GDL-90 traffic/ownship report fields.

Field	Description	Length (bits)
Traffic Alert Status	0: No alert, 1: Traffic alert for this target, 2–15: Reserved	4
Target Identity (type)	0: ADS-B with ICAO address, 1: ADS-B with self-assigned address, 2: TIS-B with ICAO address, 3: TIS-B with track file ID, 4: Surface Vehicle, 5: Ground station beacon, 6–15: Reserved	4
Participant Address	Unique address	24
Latitude and Longitude	Encoded range -180 to 180 degrees, resolution approximately 2.14577×10^{-5}	24×2
Altitude	Pressure altitude (referenced to 29.92 inches Hg), encoded using a 25-foot resolution, offset by 1,000 feet. 0xFFF is invalid/unavailable.	12
Miscellaneous Indicators	Bits 0 and 1: Additional information for the Track/Heading field, Bit 2: Report derived from ADS-B or extrapolated from the previous report, Bit 4: Air/ground state	4
Integrity and Accuracy	Integrity and accuracy of the traffic reported (in nautical mile thresholds)	4×2
Horizontal Velocity	Velocity in knots. Above 4094 knots, the value will hold at 0xFFE.	12
Vertical Velocity	Velocity in 64 feet per minute, +/- 32,578 feet per minute	12
Track/Heading	Weighted heading value, resolution 360/256 (approx. 1.4 degrees)	8
Emitter Category	Type of vehicle in use represented by an integer. Values 0–21 are in use and the rest are reserved.	8
Call Sign	8 ASCII characters (0–9, A–Z). The space is only used for padding in the end.	64
Emergency/Priority Code	0: No emergency, 1: General emergency, 2: Medical emergency, 3: Minimum fuel, 4: No communication, 5: Unlawful interference, 6: Downed aircraft, 7–15: Reserved	4
Reserved	Reserved for future use	4

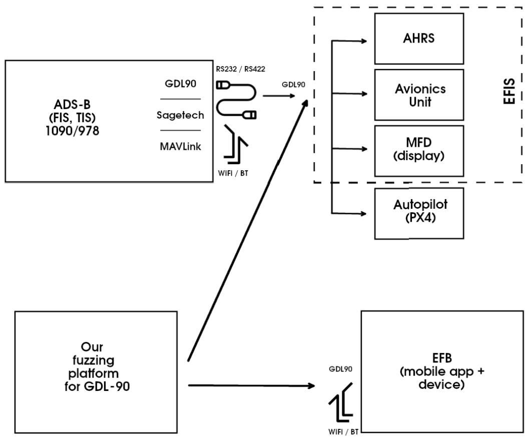


FIGURE 4. Overview of the GDL-90 test-bench and positioning of our fuzzing platform (for GDL-90 and similar avionics data-link protocols).

III. FUZZING ATTACKS ON GDL-90

A. DIAGRAMS OF OUR APPROACH

In Figure 4 we present a high-level diagram[1] of where GDL-90 outputs and inputs are connected in real-world systems and where our platform can be connected during the execution of GDL-90 fuzzing. It is important to note that discovering or triggering such protocol implementation vulnerabilities does not necessarily require physical or adjacent proximity. In another study of ours, we demonstrated that carefully crafted wireless ADS-B communications can be used to achieve the same goals, crash EFB/ADS-B apps or ADS-B avionics devices, which can be due to the GDL-90 or ADS-B vulnerabilities, or a handful of other reasons [5], [6]. This is possible because many ADS-B devices with an ADS-B IN function provide processed data using GDL-90 protocol encoding.

[1]This setup is part of a larger pentesting platform for aviation/avionics and maritime technologies [5], [6].

B. ADVANTAGES OF OUR APPROACH

Using the GDL-90 fuzzing approach that we developed and propose in this paper has the following main advantages:

1) Does not require aviation-spectrum wireless transmission (e.g., ADS-B) and thus, avoids any radio interference and lowers the costs, as SDR devices are not required (i.e., it works directly at the GDL-90 receiving point);
2) Is not limited to the capacity of radio channels and thus, can perform fuzzing/testing at considerably higher speeds (e.g., WiFi/ethernet has higher a throughput than the ADS-B RF link);
3) Works closer to the source of the possible GDL-90 implementation problems and thus, avoids the extra layer(s) introduced by higher protocols' (such as ADS-B's) processing chains, which could be sources of bottlenecks, false negatives/positives, and air-transmission regulatory challenges.

C. OVERALL HARDWARE-SOFTWARE SETUP

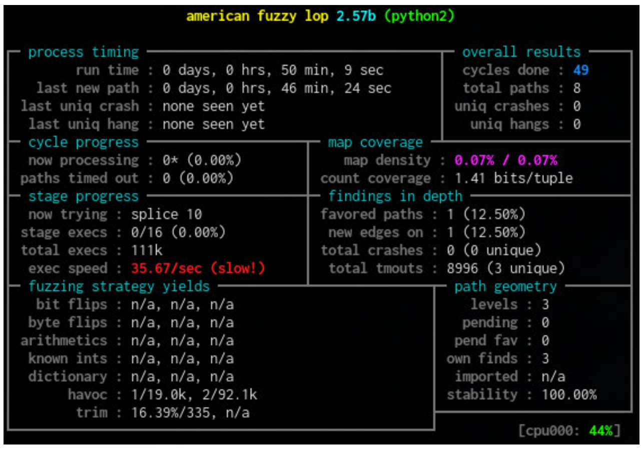
Our attacks were made simple by the fact that the common GDL-90 enables WiFi ADS-B devices (such as SkyEcho2, echoUAT, and Sentry) using connectionless UDP packets to send data. Therefore, we were able to easily capture, manipulate, and resend the packets to the applications without issues. First, we observed the packets transmitted in the WiFi networks created by the Sentry and SkyEcho2 with the Wireshark [16] network packet inspection tool. We applied the GDL-90 dissector [17] lua-script to Wireshark to identify and analyze the packets. We also transmitted ADS-B traffic messages via HackRFOne to the receivers. We copied the required messages from the packet captures and saved them as samples for the fuzzer. Depending on the device and its configuration, we either left the different message types as separate samples or left them as one in the case of Sentry. In addition to the samples that we gathered from real device networks, we also utilized Eric Dey's GDL-90 code [18] to simulate Stratus [19] and SkyRadar [20] ADS-B receivers and created samples for those. In total, we tried four different samples with the applications. Some applications worked with only one sample-specific sample set. The simulated SkyRadar sample set was deemed the best generalization of the four, due to which it was the most widely used in our tests.

We were inspired by Eric Dey's GDL-90 code [18] and made our own GDL-90 sender script for fuzzing purposes. We chose AFL as our fuzzer of choice since we were adamant that the input coverage with AFL would be sufficient. We set up our environment as a Docker container with AFL and our sender/fuzzing script. With our sender script, the target IP address and the target port must be set at the beginning. When the parameters are set, we can start fuzzing. As we used UDP packets over WiFi, the applications at the mobile phone end were not aware that the device at the other end was not legitimate; therefore, the testing was realistic. However, as we had no feedback from the mobile device through the network to the fuzzer, we could not have AFL recording the exact input that made an app crash. We could only observe the applications. Running the fuzzer over the network with a packet sending delay made the fuzzing quite slow for AFL standards. However, the applications that were affected the most crashed within the first 60 minutes of the test. For the initial test, the target and the attacking PC were both connected to the same home network via a WiFi access point that ran OpenWRT 17.01.0 [21] or via an ethernet to a router.

Overall, our test setup works on the one-click-test principle. After the Docker container is built, a test can be started by running a script with four arguments: the IP address of the attacked device, the UDP port (4000 or 43211 in our tests), the sample folder (one of our four offerings), and the output folder (arbitrary and useful for resuming long fuzzing sessions). Logs are saved to the specified output folder. With the inclusion of Docker, the setup is easy, as each component is installed automatically. Figure 5 shows a status display during the test.

D. AFL SETUP

We used AFL's Python implementation (python-afl v.0.7.3) and the latest AFL as of date (afl-fuzz v.2.57b) in our tests. As our test setup was quite slow, we specified "quick and dirty" mode (-d option), which skips deterministic steps and usually yields faster results. This limited the depth that we could achieve with the tests; however, we discovered that this mode was perfectly adequate for many applications to falter. With the non-deterministic mode on and with the sample variety low, our longest (one-hour) fuzzing sessions reached



```
                    american fuzzy lop 2.57b (python2)

┌─ process timing ─────────────────────────────────────┬─ overall results ─┐
│        run time : 0 days, 0 hrs, 50 min, 9 sec        │  cycles done : 49 │
│   last new path : 0 days, 0 hrs, 46 min, 24 sec       │  total paths : 8  │
│ last uniq crash : none seen yet                       │ uniq crashes : 0  │
│  last uniq hang : none seen yet                       │   uniq hangs : 0  │
├─ cycle progress ───────────────────┬─ map coverage ─┴───────────────────┤
│  now processing : 0* (0.00%)       │    map density : 0.07% / 0.07%      │
│ paths timed out : 0 (0.00%)        │ count coverage : 1.41 bits/tuple    │
├─ stage progress ───────────────────┼─ findings in depth ────────────────┤
│  now trying : splice 10            │ favored paths : 1 (12.50%)          │
│ stage execs : 0/16 (0.00%)         │  new edges on : 1 (12.50%)          │
│ total execs : 111k                 │ total crashes : 0 (0 unique)        │
│  exec speed : 35.67/sec (slow!)    │  total tmouts : 8996 (3 unique)     │
├─ fuzzing strategy yields ──────────┴───────────┬─ path geometry ────────┤
│   bit flips : n/a, n/a, n/a                    │    levels : 3          │
│  byte flips : n/a, n/a, n/a                    │   pending : 0          │
│ arithmetics : n/a, n/a, n/a                    │  pend fav : 0          │
│  known ints : n/a, n/a, n/a                    │ own finds : 3          │
│  dictionary : n/a, n/a, n/a                    │  imported : n/a        │
│       havoc : 1/19.0k, 2/92.1k                 │ stability : 100.00%    │
│        trim : 16.39%/335, n/a                  ├────────────────────────┘
└────────────────────────────────────────────────┘          [cpu000: 44%]
```

FIGURE 5. Example of an AFL run status.

TABLE 3. List of software exposed to fuzzing attacks ("software under test").

Name	Price	Version (Android / iOS)	Installations (Android / iOS)
OzRunways	Free	v.4.5.6 / v.10.1.9	50k+ / –
iFlightPlanner	Free	– / v.4.5.6	– / –
Airmate	Free	v.1.6.1 / v.2.3	50k+ / –
AvPlan	$76.16/year	v.1.3.28 / v.8.1.2	5k+ / –
Levil Aviation	Free	– / v.1.3	– / –
FLYQ EFB	Free	– / v.5.0	– / –
EasyVFR4	Free	v.4.0.870 / v.4.0.899	100+ / –
Horizon	Free	v.3.0 / v.3.0	10k+ / –
ForeFlight	Free	– / v.13.4	– / –
Pilots Atlas	$89.99/year	– / v.5.13.0	– / –
Xavion	Free	– / v.2.81	– / –
SkyDemon	$162/year	v.3.15.0 / v.3.15.3	100k+ / –
Stratus Insight	$99.99/year	– / v.5.17.3	– / –
Naviator	$34.99/year	v.4.2.2 / –	100k+ / –
Traffic (by Control-J) traffic	Free	v.0.0.2.4 / –	10+ / –
Eric Dey's GDL-90 gdl90etdey	Free	github repo commit d7e5936	–

at least 50 cycles. A cycle in AFL means that the fuzzer went through all the interesting test cases [22]. Therefore, we argue that the tests were quite thorough within the limitations of the samples we acquired. We observed that the crashes occurred at several stages of the fuzzing cycles. Even if the test applications did not crash, the usability of the data they presented was greatly hindered due to the malformed input data (see the details in the results in Section IV).

E. GDL-90 FUZZING TARGETS

In Table 3, we present a comprehensive list of the targeted software. We targeted mostly mobile EFB apps, but we also tested some open-source tools. For Eric Dey's GDL-90 code [18], we targeted only the decoding script.

F. HIGH-LEVEL GDL-90 ATTACK DESCRIPTION

A possible cybersecurity attack involving vulnerable GDL-90 implementations could look as follows:

1) *At the research time*: An exploitable GDL-90 vulnerability is first discovered (e.g., using our implementation-independent GDL-90 protocol fuzzing approach).
2) *At the design/manufacturing time*: An adversary designs and puts on the market an ADS-B-capable and GDL-90-compatible "backdoored" device that contains the GDL-90 exploitation payloads and attack vectors. The "backdoor" could be implemented at the hardware or at the firmware level in such a way to avoid the detection at the (re-)certification time (similar to the Volkswagen emission engine control unit manipulation scandal [23]).
3) *At the usage time*: The "backdoored" ADS-B-capable device sends or activates the GDL-90 exploitation payload. Such exploitation payloads could be activated conditionally, such as at certain altitudes, within certain geo-fence areas, and upon receiving a "secret knock" ADS-B message.
4) *At the usage time*: Alternatively, the discovered GDL-90 vulnerability can be reconstructed back to a specially crafted triggering ADS-B payload/message. Therefore, it may even be possible to trigger the GDL-90 vulnerability without "backdoored" hardware, by simply sending a specially crafted ADS-B payload/message.
5) Ultimately, backdoors have been shown to be implanted even in military-grade chips [24]. Therefore, it is more than reasonable to believe that backdoor implanting is also feasible for ADS-B devices destined for avionics/EFIS/IFD/EFB setups within commercial/general aviation and amateur aircraft.

IV. RESULTS

The fuzzing results are presented in Table 4. Of the 15 tested mobile EFB applications, 6 crashed (4 iOS-only and 2 iOS+Android) and 2 became unresponsive (1 iOS-only and 1 Android-only). In addition to mobile the EFB apps, Eric Dey's open-source GDL-90 [18] decoder experienced several dozen of unique crashes during a day-long fuzzing session on a normal PC (Linux). We focused only on fuzzing Eric Dey's GDL-90 decoder, leaving its network component out of the equation. The unique errors and crashes that we recorded were related to the different inputs that generated Python assertion statement failures which, in turn, were due to the faulty lengths of the messages. (Finding such issues is exactly the aim of fuzzing tests in general.) These results allow us to assume that Eric Dey's open-source GDL-90 [18] could pose stability, availability, and DoS-resiliency issues if deployed or operated "as-is" in real-world systems and devices.

In one of our recent works [5], [6], we tested almost the same set of mobile apps and devices for DoS attacks

TABLE 4. Details of the mobile applications (apps) considered "attacked software".

App Name	GDL-90: Android	GDL-90: iOS	Comparison with our ADS-B-level DoS attacks [5], [6]
OzRunways	CRA (once)	CRA	CRA
Stratus Insight	NA-P	CRA	CRA
iFlightPlanner	NA-P	CRA	DNW
AirMate	DNW	CRA	CRA
AvPlan	CRA / UNR	CRA	CRA
Levil Aviation	NA-P	CRA	DNT
FlyQ EFB	NA-P	UNR	DNC
EasyVFR4	DNC	DNC	DNC
Horizon	DNC	DNC	DNT
ForeFlight	NA-P	DNC	CRA
Pilots Atlas	NA-P	DNC	DNC
Xavion	NA-P	DNC	DNT
Traffic (by Control-J)	DNC	NA-P	DNT
Naviator	UNR	NA-D	DNW
SkyDemon	DNC	DNC	DNW

Android = Samsung Galaxy A21s, Android v 11
iOS = Apple iPhone SE, iOS v 13.3
Acronyms: CRA = Crash; UNR = Unresponsive/Hang; DNC = Did Not Crash; DNT = Did Not Test; DNW = Did Not Work (e.g., did not connect to hardware, did not receive data); NA-G = Not Available for this Geography/Country/Region; NA-P = Not Available for this Platform; NA-D = Not Available for this Device.

via the ADS-B layer and found that 6 of the mobile apps in Table 4 were impacted by the ADS-B IN DoS attack, which possibly affected over 200,000 mobile application installs worldwide. In [5], [6], we tested a total of 68 different ADS-B configurations (mobile and non-mobile) for the ADS-B IN DoS attack. We managed to crash 25% of them mostly within 2 minutes, while overall, the DoS attack impacted 51.47% of the tested configurations. In comparison, the fuzzing results presented in this paper have similarly worrying results in terms of aviation safety and lack of resiliency to cybersecurity attacks such as DoS. Attacks on core subsystem availability (such as DoS) pose high risks to safety-critical and mission-critical systems such as avionics and aerospace.

A. VISUAL OBSERVATIONS

All the mobile application crashes were observed by visually inspecting the device and software under test. The crashes happened either by themselves or while trying to operate the software (e.g., after any touch input, movement of the map, or zooming in/out) while the test was running. For each tested configuration that was impacted, the crashes were observed and confirmed at least three times (unless noted otherwise) before the result was registered.

Although FlyQ did not crash, it became unresponsive and had to be closed by the user. OzRunways on Android crashed, but the result was not consistently repeated. The Naviator app on Android did not crash during the test. However, it consistently closed the GDL-90 ADS-B input on an error state in each of our attempts and recovered only after a restart. Otherwise, the application remained functional. Most of the test applications showed some abnormal behavior, such as an irrationally flinching map screen, fluctuating GPS data (due to the GPS positioning taken from the GDL-90 messages), alerts (due to plane proximity or altitude readings), and other non-standard or device operator-alerting behavior. Therefore, the apps marked with DNCs (Did Not Crash) in Table 4 should not be considered conclusively stable [25]. The applications that did not crash in our tests in this study may crash with some other sample data or testing methods.

V. RELATED WORKS

A. SOFTWARE FUZZING

Reliable and efficient aerial communication is at the heart of aerospace safety. Any defects in this safety-critical technology may cost human lives and property. However, modern protocols and the accompanying software are not always up to the task. Several studies have shown numerous viable attacks on these protocols and software [4], [26]. Developers, researchers, and hackers are using many tools to find out the security vulnerabilities of this kind of mission-critical system. Here, we discuss a few of them.

The success stories and the open-source nature of AFL have encouraged researchers to customize this fuzzer for different tasks [10]. Numerous studies have added many functionalities to the AFL (e.g., pathfinding, sample creation, and coverage) to improve its performance and effectiveness [27]–[35]. As a result, AFL has been added to commercial off-the-shelf (COTS) binaries [36]–[38]. It has also received modifications for its parallel-run capabilities [39].

B. ATTACKS AND FUZZING ON AVIONICS DATA-LINK

The Micro Air Vehicle Link (MAVLink) communication protocol is a bidirectional communication protocol that is used in drones and ground control stations. It offers different types of messages that can be transmitted reliably in an efficient package [40]. However, Domin *et al.* reported a crash of MAVLink-capable software in their protocol fuzzing tests in 2016 [41]. They were able to crash a virtual drone with a random payload by incrementally increasing the payload bytes from 1 to 255, thus increasing the length of the whole message. An open-source MAVLink fuzzing software is available [42].

PX4 is a widely avaible and extremely popular flight controller that also supports the MAVLink protocol as well as data from ADS-B IN-capable devices (such as Aerobits AERO and uAvionix pingRX). Alias Robotics [43] presented a general cybersecurity overview of PX4 from threat modeling and static analysis perspectives and, in this context, introduced the Robot Vulnerability Database (RVD). Subsequently, Jang *et al.* [44] performed a thorough static analysis of various PX4 firmware codebases.

Other communication protocols are also used for drones in particular. Rudo and Zeng [45] showed fuzzing results on the file transfer protocol/session initiation protocol (FTP/SIP) and session description protocol (SDP) embedded in consumer-grade drones. They raised concerns about the state of security of such drones with commercial drone software. They demonstrated GPS navigation and other subsystem failures (e.g., video feed and motor issues). Multiple studies have shown that the Internet-of-Things (IoT) and embedded devices are quite vulnerable [46], [47].

With regad to drone security issues, Kim *et al.* published their robotic vehicle (RV) fuzzing tool called "RVFuzzer" [48]. This tool was designed to highlight missing or faulty validation checks for control inputs. These bugs and missing features may cause physical disruptions, such as mission failures or crashes, on RVs, such as drones, if exploited. The authors constructed the RVFuzzer to employ three distinct strategies for searching input validation bugs, such as control parameter mutation, one-dimensional mutation, and multidimensional mutation. Throughout their evaluation, they discovered 89 input validation bugs from two control programs. Since the attacks do not require any code injection or other invasive procedures, they cannot be detected by security solutions [48]. Hence more specific code improvements and internal security audits for source codes under development are required.

C. ATTACKS ON AVIONICS SYSTEMS AND PROTOCOLS

The research community has adamantly scrutinized the security of ADS-B communication over the years. In 2004, Korzel *et al.* [49] demonstrated issues with the data integrity of the protocol due to erroneous inputs and data dropouts. Further concerns over the authenticity, security, confidentiality, and integrity of such protocol have been periodically raised since [50]–[52].

Several researcher have frequently demonstrated attacks against the ADS-B protocol. Costin and Francillon [4] conducted the first practical ADS-B message injection and spoofing attacks. Schäfer *et al.* [26] exposed several attacks such as ghost aircraft attacks and virtual trajectory modification on budget devices. Sjödin and Gruneau [53] used HackRF SDR to demonstrate data injection and flooding attacks on the Sentry ADS-B transceiver. They concluded that the device does not validate the messages from the ADS-B protocol. McCallie *et al.* [54] classified such attacks and explored their consequences, which resulted in worrying results.

Portable ADS-B transceivers (e.g., SkyEcho2, Sentry, and echoUAT), which are operated with iPads and other tablets, are favored by many general aviation pilots due to their ease of setup, ease of use, and affordable pricing. As these devices are not part of the onboard avionics per se, Lundberg *et al.* [55], [56] pointed out that they do not, nor do they need to, meet the standards of traditional avionics (e.g., RTCA, ARINC, and EUROCAE). The authors also found vulnerabilities on all of their test samples and recommended further product improvements to the device and software designers.

VI. CONCLUSION

In this paper, we presented our study of the impact of GDL-90 protocol fuzzing on a range of popular mobile EFBs and some standard PC software. Our results showed a worrying lack of security in many EFB applications where the security is directly related to aircraft's safety navigation. Of the 16 configurations that we tested herein, our avionics pentesting platform managed to crash or otherwise impact 9 configurations (56%). During the fuzzing test, we observed crashes, hangs, and other abnormal behaviors of the EFB apps and GDL-90 decoders. The consistency of our test results on a heterogeneous and representative set of EFBs (on the Android, iOS, and Linux platforms) indicates the reliability of our approach and results. DoS attacks can be devastating for mission-critical systems such as in avionics and aerospace, where the availability and reliability of the system are crucial. However, we hope that our results and presented methodology can motivate the standardization and regulatory bodies, as well as the industry and air traffic organizations, to improve the requirements for and the implementation checks of avionics devices and apps with regard to resiliency to cybersecurity attacks, and in particular, resiliency to DoS attacks. To ensure the adequate safety of such mission-critical systems, multidimensional security measures need to be taken. For avionics devices and related software/firmware, upgrading their defence against cyberattacks should be considered a continuous process, and thus, related research and development need to be sustained along with the operation of such devices and technologies.

ACKNOWLEDGMENT

All results, views, and opinions presented herein are only those of the authors and do not reflect the official position of the European Union and its organizations and projects, including the Horizon 2020 Program and Engage KTN. The authors thank Dr. Andrei Costin for facilitating and managing the grant.

The work of Hannu Turtiainen also acknowledges and thanks the Finnish Cultural Foundation / Suomen Kulttuurirahasto (https://skr.fi/en) for supporting his Ph.D. dissertation work and research and the Faculty of Information Technology of JYU, in particular, Prof. Timo Hämäläinen, for partly supporting and supervising his Ph.D. work at JYU in 2021–2022.

REFERENCES

[1] *No Kidding: ADS-B Deadline of Jan. 1, 2020, is Firm*. Accessed: Jun. 11, 2021. [Online]. Available: https://www.faa.gov/news/updates/?newsId=90008

[2] EASA. (2018). *Easa Seasonal Technical Commission*. Accessed: Mar. 2, 2021. [Online]. Available: https://www.easa.europa.eu/sites/default/files/dfu/EASA_STC_NEWS_JUNE_2018.pdf

[3] N. Demidovich, "Federal aviation administration incremental flight testing of automatic dependent surveillance-broadcast (ADS-B) prototype for commercial space transportation applications," in *Proc. ITEA 32nd Annu. Int. Test Eval. Symp.* Washington, DC, USA: Federal Aviation Administration, Aug. 2015.

[4] A. Costin and A. Francillon, "Ghost in the air (Traffic): On insecurity of ADS-B protocol and practical attacks on ADS-B devices," in *Proc. Black Hat USA*, 2012, pp. 1–12.

[5] S. Khandker, H. Turtiainen, A. Costin, and T. Hamalainen, "Cybersecurity attacks on software logic and error handling within ADS-B implementations: Systematic testing of resilience and countermeasures," *IEEE Trans. Aerosp. Electron. Syst.*, early access, Dec. 31, 2021, doi: 10.1109/TAES.2021.3139559.

[6] S. Khandker, H. Turtiainen, and A. Costin, "Practical denial-of-service and combined high-level attacks on real-world ADS-B, ATC, ATM hardware and software," 2021.

[7] B. Miller. *Fuzz Testing of Application Reliability*. Accessed: May 25, 2021. [Online]. Available: http://pages.cs.wisc.edu/~bart/fuzz/

[8] OWASP. *Fuzzing*. Accessed: Jun. 30, 2021. [Online]. Available: https://owasp.org/www-community/Fuzzing

[9] G. O. Source. *Github: Afl*. Accessed: Jun. 30, 2021. [Online]. Available: https://github.com/google/AFL

[10] M. Zalewski. *American Fuzzy Lop*. Accessed: Jun. 30, 2021. [Online]. Available: https://lcamtuf.coredump.cx/afl/

[11] A. S. Foundation. *Apache License, Version 2.0*. Accessed: Jul. 1, 2021. [Online]. Available: https://www.apache.org/licenses/LICENSE-2.0

[12] *RTCA DO-267: Minimum Aviation System Performance Standards (MASPS) for Flight Information Services-Broadcast (FIS-B) Data Link*, RTCA, Washington, DC, USA, 2014.

[13] *GDL 90 Data Interface Specification*, Garmin, Olathe, KS, USA, 2007.

[14] Bendix/King. (2021). *KAP 140 Autopilot System*. [Online]. Available: https://www.bendixking.com/content/dam/bendixking/en/documents/document-lists/downloads-and-manuals/006-18034-0000-KAP-140-Pilots-Guide.pdf

[15] Garmin. (2021). *G1000 System*. [Online]. Available: https://buy.garmin.com/en-U.S./U.S./p/6420

[16] *Wireshark Homepage*. Accessed: May 25, 2021. [Online]. Available: https://www.wireshark.org/

[17] B. Kyser. *Github: Gdl90Dissector*. Accessed: May 25, 2021. [Online]. Available: https://github.com/BrantKyser/gdl90Dissector

[18] E. Dey. *Github: Gdl90*. Accessed: May 5, 2021. [Online]. Available: https://github.com/etdey/gdl90

[19] *Stratus ADS-B Receiver*. Accessed: May 5, 2021. [Online]. Available: https://stratusbyappareo.com/products/stratus-ads-b-receivers/

[20] SkyRadar Radenna LLC. *SkyRadar ADS-B Receiver*. Accessed: May 5, 2021. [Online]. Available: https://www.skyradar.net/skyscope-receiver/receiveroverview.html

[21] OpenWrt. *OpenWrt Project*. Accessed: Jul. 1, 2021. [Online]. Available: https://openwrt.org/

[22] Google. *AFL User Guide*. Accessed: Jul. 1, 2021. [Online]. Available: https://afl-1.readthedocs.io/en/latest/user_guide.html

[23] M. Contag, G. Li, A. Pawlowski, F. Domke, K. Levchenko, T. Holz, and S. Savage, "How they did it: An analysis of emission defeat devices in modern automobiles," in *Proc. IEEE Symp. Secur. Privacy (SP)*, May 2017, pp. 231–250.

[24] S. Skorobogatov and C. Woods, "Breakthrough silicon scanning discovers backdoor in military chip," in *Proc. Int. Workshop Cryptograph. Hardw. Embedded Syst.* London, U.K.: Quo Vadis Labs, 2012, pp. 23–40.

[25] M. Muench, J. Stijohann, F. Kargl, A. Francillon, and D. Balzarotti, "What you corrupt is not what you crash: Challenges in fuzzing embedded devices," in *Proc. NDSS*, 2018, pp. 1–15.

[26] 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, Germany: Springer, 2013, pp. 253–271.

[27] C. Lemieux and K. Sen, "FairFuzz: A targeted mutation strategy for increasing greybox fuzz testing coverage," in *Proc. 33rd ACM/IEEE Int. Conf. Automated Softw. Eng.*, Sep. 2018, pp. 475–485.

[28] N. Nichols, M. Raugas, R. Jasper, and N. Hilliard, "Faster fuzzing: Reinitialization with deep neural models," 2017, *arXiv:1711.02807*.

[29] K. Patil and A. Kanade, "Greybox fuzzing as a contextual bandits problem," 2018, *arXiv:1806.03806*.

[30] R. K. Prakash, I. Vasudevan, I. Indhuja, T. Thangarasan, and C. Krishnan, "Hardiness sensing for susceptibility using American fuzzy lop," in *Proc. ITM Web Conf.*, vol. 37, 2021, pp. 1–4.

[31] M. Rajpal, W. Blum, and R. Singh, "Not all bytes are equal: Neural byte sieve for fuzzing," 2017, *arXiv:1711.04596*.

[32] L. Sun, X. Li, H. Qu, and X. Zhang, "AFLTurbo: Speed up path discovery for greybox fuzzing," in *Proc. IEEE 31st Int. Symp. Softw. Rel. Eng. (ISSRE)*, Oct. 2020, pp. 81–91.

[33] J. Wang, B. Chen, L. Wei, and Y. Liu, "Superion: Grammar-aware greybox fuzzing," in *Proc. IEEE/ACM 41st Int. Conf. Softw. Eng. (ICSE)*, 2019, pp. 724–735.

[34] X. Yuan, L. Pan, and S. Luo, "Binary fuzz testing method based on LSTM," *IOP Conf., Mater. Sci. Eng.*, vol. 612, Oct. 2019, Art. no. 032192.

[35] G. Zhang and X. Zhou, "AFL extended with test case prioritization techniques," *Int. J. Model. Optim.*, vol. 8, no. 1, pp. 41–45, Feb. 2018.

[36] Y. Chen, D. Mu, J. Xu, Z. Sun, W. Shen, X. Xing, L. Lu, and B. Mao, "Ptrix: Efficient hardware-assisted fuzzing for COTS binary," in *Proc. ACM Asia Conf. Comput. Commun. Secur.*, 2019, pp. 633–645.

[37] S. Dinesh, N. Burow, D. Xu, and M. Payer, "RetroWrite: Statically instrumenting COTS binaries for fuzzing and sanitization," in *Proc. IEEE Symp. Secur. Privacy (SP)*, May 2020, pp. 1497–1511.

[38] Y. Zheng, A. Davanian, H. Yin, C. Song, H. Zhu, and L. Sun, "FIRM-AFL: High-throughput greybox fuzzing of iot firmware via augmented process emulation," in *Proc. 28th Secur. Symp.*, 2019, pp. 1099–1114.

[39] J. Ye, B. Zhang, R. Li, C. Feng, and C. Tang, "Program state sensitive parallel fuzzing for real world software," *IEEE Access*, vol. 7, pp. 42557–42564, 2019.

[40] (2021). *Mavlink Developer Guide*. [Online]. Available: https://mavlink.io/en/

[41] K. Domin, I. Symeonidis, and E. Marin, "Security analysis of the drone communication protocol: Fuzzing the MAVLink protocol," in *Proc. ORBIlu*, 2016, pp. 1–7.

[42] Auterion. (2019). *Github: MAVLink Fuzz Testing*. [Online]. Available: https://github.com/Auterion/mavlink-fuzz-testing

[43] Alias Robotics. *The Cybersecurity Status of PX4*. Accessed: Jul. 1, 2021. [Online]. Available: https://aliasrobotics.com/files/cybersecurity_status_PX4.pdf

[44] J.-H. Jang, Y.-S. Kang, and J.-H. Lee, "Static analysis and improvement opportunities for open source of UAV flight control software," *J. Korean Soc. Aeronaut. Space Sci.*, vol. 49, no. 6, pp. 473–480, Jun. 2021.

[45] D. Rudo and D. Kai Zeng, "Consumer UAV cybersecurity vulnerability assessment using fuzzing tests," 2020, *arXiv:2008.03621*.

[46] A. Costin, J. Zaddach, A. Francillon, and D. Balzarotti, "A large-scale analysis of the security of embedded firmwares," in *Proc. 23rd USENIX Secur. Symp.*, 2014, pp. 1–22.

[47] A. Costin, A. Zarras, and A. Francillon, "Automated dynamic firmware analysis at scale: A case study on embedded web interfaces," in *Proc. 11th ACM Asia Conf. Comput. Commun. Secur.*, May 2016, pp. 437–448.

[48] T. Kim, C. H. Kim, J. Rhee, F. Fei, Z. Tu, G. Walkup, X. Zhang, X. Deng, and D. Xu, "RVFuzzer: Finding input validation bugs in robotic vehicles through control-guided testing," in *Proc. 28th Secur. Symp.*, 2019, pp. 425–442.

[49] J. Krozel, D. Andrisani, M. Ayoubi, T. Hoshizaki, and C. Schwalm, "Aircraft ADS-B data integrity check," in *Proc. 4th Aviation Technol., Integr. Oper. (ATIO) Forum*, 2004, p. 6263.

[50] K. Samuelson, E. Valovage, and D. Hall, "Enhanced ADS-B research," in *Proc. IEEE Aerosp. Conf.*, Oct. 2006, pp. 1–7.

[51] R. G. Wood, "A security risk analysis of the data communications network proposed in the nextgen air traffic control system," Ph.D. dissertation, Dept. Inf. Comput. Sci., Oklahoma State Univ., Stillwater, OK, USA, 2009. [Online]. Available: https://search.proquest.com/dissertations-theses/security-risk-analysis-data-communications/docview/305083310/se-2?accountid=11774

[52] L. Purton, H. Abbass, and S. Alam, "Identification of ADS-B system vulnerabilities and threats," *Proc. 33rd Australas. Transp. Res. Forum (ATRF)*, 2010, pp. 1–16.

[53] A. Sjödin and M. Gruneau, "The ADS-B protocol and its' weaknesses: Exploring potential attack vectors," KTH Skolan för Elektroteknik och Datavetenskap, Stockholm, Sweden, Tech. Rep., Jun. 2020.

[54] D. McCallie, J. Butts, and R. Mills, "Security analysis of the ADS-B implementation in the next generation air transportation system," *Int. J. Crit. Infrastruct. Protection*, vol. 4, no. 2, pp. 78–87, Aug. 2011.

[55] D. Lundberg, B. Farinholt, E. Sullivan, R. Mast, S. Checkoway, S. Savage, A. C. Snoeren, and K. Levchenko, "On the security of mobile cockpit information systems," in *Proc. ACM SIGSAC Conf. Comput. Commun. Secur.*, Nov. 2014, pp. 633–645.

[56] D. A. Lundberg, "Security of ADS-B receivers," Ph.D. dissertation, Dept. Comput. Sci. Eng., UC San Diego, San Diego, CA, USA, 2014.

HANNU TURTIAINEN received the B.Sc. degree in electronics engineering from the University of Applied Sciences, Jyväskylä, Finland, and the M.Sc. degree in cybersecurity, in 2020. He is currently pursuing the Ph.D. degree in software and communication technology with the University of Jyväskylä, Finland. He is also working in the IoT field as a Cybersecurity and Software Engineer with Binare.io, a deep-tech cybersecurity spin-off from the University of Jyväskylä. His research interests include machine learning and artificial intelligence in the cybersecurity and digital privacy.

ANDREI COSTIN received the Ph.D. degree from the EURECOM/Telecom ParisTech, under co-supervision of Prof. Francilon and Prof. Balzarotti, in 2015. He is currently a Senior Lecturer/Assistant Professor of cybersecurity at the University of Jyväskylä (Central Finland), with a particular focus on the IoT/firmware cybersecurity and digital privacy. He is also the CEO/Co-Founder of Binare.io, a deep-tech cybersecurity spin-off from the University of Jyväskylä, focused on innovation and tech-transfer related to the IoT cybersecurity. He has been publishing and presenting at more than 45 top international cybersecurity venues, both academic (Usenix Security and ACM ASIACCS) and industrial (BalckHat, CCC, and HackInTheBox). He is the author of the first practical ADS-B attacks (BlackHat 2012) and has literally established the large-scale automated firmware analysis research areas (Usenix Security 2014)-these two works are considered seminal in their respective areas, being also most cited at the same time.

SYED KHANDKER received the M.Sc. degrees in web intelligence and service engineering from the University of Jyväskylä, Finland, in 2016. He is currently pursuing the Ph.D. degree with the Faculty of Information Technology, University of Jyväskylä. Since his childhood, he has been a Radio Enthusiast and holds an Amateur Radio Operator License. He has authored four journal and conference publications. His research interests include RF fingerprint positioning, automatic dependent surveillance-broadcast, automatic identification system, wireless communications, and artificial intelligence.

TIMO HÄMÄLÄINEN has over 25 years of research and teaching experience related to computer networks. He has lead tens of external funded network management related projects. He has launched and leads master's programs with the University of Jyväskylä (currently SW and communication engineering) and teaches network management related courses. He has more than 200 internationally peer-reviewed publications and he has supervised 36 Ph.D. theses. His current research interests include wireless/wired network resource management (the IoT, SDN, and NFV) and network security.