Available online at www.sciencedirect.com

ScienceDirect

journal homepage: www.elsevier.com/locate/cose

Computers & Security

Exploiting the MIL-STD-1553 avionic data bus with an active cyber device

D. De Santo[b], C.S. Malavenda[c], S.P. Romano[a,*], C. Vecchio[b]

[a] University of Napoli Federico II, Italy
[b] Accademia Aeronautica, Italy
[c] Elettronica S.p.A, Italy

ARTICLE INFO

Article history:
Received 7 April 2020
Revised 26 September 2020
Accepted 23 October 2020
Available online 27 October 2020

Keywords:
Exploitation;
MIL-STD-1553
Avionic data bus communication
Cyber attacks
Software design
Software implementation
Hardware implementation
Testing

ABSTRACT

The ensemble of cyber and electromagnetic activities has become predominant in present-day battlegrounds and armed forces are witnessing the introduction of complex and interconnected weapon systems. According to United States military doctrine, cyberspace has been classified as the fifth dimension of the battlespace and much effort is being devoted by NATO to secure activities in such an environment. This paper delves its roots in a vast research endeavor aimed at granting cyber resilience of current and future assets. Within such a broad and challenging domain, the work has a very precise scope: to address the vulnerabilities of a specific avionic platform protocol, namely the MIL-STD-1553 standard. The discussion will unfold into the design and implementation of a cyber device that is able to exploit the identified vulnerabilities. The device in question will also be tested to prove the effectiveness of the proposed solution. The outcomes will stand as a call to action towards improving the security of state-of-the-art avionic systems.

1. Introduction

Doctrine experts and scholars have summarized the evolution of war waging into a series of evolutionary steps (Lydiate, 2018). We are now in the fifth generation, dominated by Information Technology (IT), Electronic Warfare, asymmetric conflicts and unmanned weapon systems (Lind et al., 1989). The last decades have seen a breakthrough - a new technological revolution - which has transported humanity into the digital era. Indeed, computers and networks have revolutionized the way data are gathered, processed, stored and shared. Such a trend has transformed the military world as well (Hicks, 2004). Military aircraft are now unmanned, controlled through networks and satellites or by AI (Artificial Intelligence). Information exchange and automated processing on the battlefield are inescapable to properly address Command and Control functions. IT is ubiquitous and it has come to define a new domain, independent from the land, sea, air and space, but transverse to all. The new domain, usually referred to as *cyberspace*, includes the Internet, telecommunications networks, computer systems, and embedded processors and controllers.

In the depicted scenario, huge efforts have recently been devoted to securing TCP/IP networks. This is a natural consequence of the concern of the civilian compartment, where the TCP/IP protocol stack is ubiquitous for business, data management and service delivery. As a consequence, a vast literature

* Corresponding author.
E-mail addresses: damiano.desanto@aeronautica.difesa.it (D. De Santo), ClaudioSanto.Malavenda@elt.it (C.S. Malavenda), spromano@unina.it (S.P. Romano), cristian.vecchio@aeronautica.difesa.it (C. Vecchio).
https://doi.org/10.1016/j.cose.2020.102097

is available on the topic (Lehto and Neittaanmäki, 2015; Vacca, 2014). This, however, is not the sole Information Technology deployed in the military compartment. Other systems are in use, specifically designed for being embedded in modern weapons, platforms, sensors and operational links. Constant security assessment of the protocols and devices has to be performed so to assure that tactical operations are not compromised in any way by cyber and electromagnetic activities. One major area of interest is that of military aircraft cyber security, which employs networking systems both internally and externally. The manifold implications of cyber threats have led to a significant research effort on cyber-physical aspects of warfare, especially following the Stuxnet, Georgian and Estonian cases (Andress and Winterfeld, 2014). The consequences of IT-based attacks that have repercussions upon the physical world are handily imaginable and assessable, yet not easily preventable (Ali et al., 2018; Gunduz and Das, 2020; Koç, 2018). In the scope of this article, aircraft and Command & Control systems are considered as cyber-physical environments where any menace can have real-world consequences and play a strategic role in any scenario (Andress and Winterfeld, 2014). Hence, the relevance of cyber warfare in modern day scenarios is apical, as nations around the world are exploring the numerous implications of a novel dimension (Charles Billo and Chang, 2004; Connell and Vogler, 2017).

This work will focus on MIL-BUS-1553 (Data Device Corporation, 2003), a military standard for avionic BUS network infrastructure. We will explore and analyze aspects which will prove the relevance of evaluating avionic protocol security. Literature is abundant of research regarding cybersecurity applied to communication buses. However, MIL-STD-1553 is a niche standard that has not been addressed in a thorough manner as to its cyber characteristics. In particular, in Stan et al. (2017) a possible solution for implementing cyber resilience through statistical intrusion detection systems is illustrated. However, it has not been clarified how feasible it is to exploit possible MIL-STD-1553 vulnerabilities, nor if any of them might indeed have repercussions upon systems.

Modern conflicts require the employment of modern weapon systems. The most recent technical advancements are mainly concentrated on information management. As an example, the Joint Strike Fighter (JSF) - fifth generation aircraft - differentiates from its predecessors in the way it acquires, processes and shares data with other actors in the battlefield, allowing for an augmented situational awareness. However, the enormous advantages provided by IT are counterbalanced by a series of vulnerabilities that it introduces into modern systems. Such vulnerabilities can be exploited at every level (Lydiate, 2018), from the design phase of new systems, through the supply chain and up to the maintenance routines. The impact at the operational level might range from data leakage to the inability of employing specific weapon systems, or even to the loss of assets. It is important to consider that the menace represented by cyber attacks not only affects the most recent systems, but also those systems that have been engineered in the past decades. Potentially, the latter are affected at an even higher degree. Indeed, such systems employ legacy versions of embedded computers, sensors, networks, etc., which have been developed in a time when security was not as relevant as it is nowadays. A potential vulnerability is then represented by the interconnection of sensors, actuators, embedded computers, controllers and weapon systems on an aircraft. In such a scheme, all devices are physically connected to the same wire and communicate according to a predefined protocol. Numerous aircraft, still in service today, employ a copper twisted pair as physical means and the MIL-STD-1553 standard as communication protocol.

This work hence originates from operational needs. Military Air Forces are nowadays devoting their efforts towards acquiring cyber-security skills. Such a process involves assessing and mitigating vulnerabilities in all employed assets. The path is laid out and among the fields to be explored, avionics is predominant.

One of the primary objectives is to assess avionic platform resilience and to provide means for the development of suitable defense tools. To design proper defense strategies, however, one must be aware of both the attack surface and the cost-effectiveness of potential defence strategies.

Based on the above considerations, this paper will focus on vulnerabilities of the MIL-STD-1553 standard. In particular, it will document the development of a Cyber Device designed to perform attacks on a MIL-STD-1553 network. This is expected to provide the Italian Air Force an unpolished and demonstrative, yet functional and meaningful tool, for assessing potential vulnerabilities of avionic buses, and consequently identify any countermeasures that will prove necessary. The final effort will be to test the feasibility of few selected attack procedures, through simulators, rigs and real systems.

2. The MIL-STD-1553 standard

MIL-STD-1553 (Data Device Corporation, 2003) is a military standard for the implementation of data bus communication on critical assets such as aircraft, vehicles and ships. The protocol, first introduced in 1975, is based upon digital internal time-division multiplexing, which provides high reliability and low error rate. It is, therefore, widely applied in numerous systems for the integration of airborne devices such as platforms, weaponry, management, I/O, etc. The latest release, MIL-STD-1553B (from now on, 1553), provides the electrical interface requirements and the communication protocols to be implemented on a 1 Mbps copper-wired serial bus. MIL-STD-1760C (from now on, 1760) defines the requirements for aircraft-store[1] electrical interconnection system. The data communication protocol for stores is indeed a revision of 1553, introducing some exceptions within 1760 to meet interoperability and safety.

Attempting a vulnerability assessment on a bus protocol which was designed to be highly reliable though not intrinsically secure could be regarded as a simple feat. Nevertheless, the standard adopts peculiar solutions - such as a centralized multiplexing scheme and hardwired/hard-coded addresses - which may provide some level of security, or, at least, render a logic-level attack nontrivial. The same characteristics, however, could prove to be the weak spots of the system, and, if

[1] Stores are defined as any external device attached to the aircraft, like, e.g., weapons, external fuel tanks, pods.

not mitigated by terminal logic, they could prove to be a single point of failure (SPoF). As described in the following sections, the most intriguing cyber menaces presented in this paper all relate to hacking the centralized multiplexing scheme of the protocol. Compared to other widespread schemes, such as the Controlled Area Network used in automotive applications, the MIL-STD-1553 has a substantially different logic, so that all considerations that are state of the art (Bozdal et al., 2018; El-Rewini et al., 2020; Wu et al., 2020) are not directly applicable, or assumed to be applicable. Precise and punctual aspects of the standard have to be taken into account (Stan et al., 2017). Nevertheless, the general mechanisms and purpose of the networking systems are comparable and compatible, so that jamming attacks (DoS in this scope) would work similarly.

This said, the discussion on the menaces will be as general as possible, will not refer to any specific implementation and will be based on strong assumptions. This is necessary since we had to strike a balance between clearly describing our work and contribution on one side, and avoiding disclosure of classified military information on the other.

We will provide a general overview of 1553 and its functions, describing the hardware involved in the message exchange phase and illustrating message formatting, timing and synchronization issues as well as BUS management. Electromagnetic issues such as wiring, coupling, isolation and physical requirements are not of interest and the reader is referred to Data Device Corporation (2003) for such details.

Any MIL-STD-1553 infrastructure is composed of three core elements: the bus controller(s) (BC), the remote terminal(s) (RT), and the bus monitor(s) (BM).

The BUS controller's main function is to provide data flow control for all transmissions in the network. Since the system adopts a command/response mechanism, the BC is the exclusive source of communication. As such, it is formally defined as "the terminal assigned the task of initiating information transfer on the data bus." The BC is not being described according to its physical design, but solely with reference to its assigned task. Consequently, any terminal connected to the physical infrastructure might perform the role of BC, while simultaneously addressing other functions.

The BM is defined as "the terminal assigned the task of receiving traffic and extracting selected information to be used at a later time." Therefore, the device is a passive element on the BUS, as it does not initiate communication nor respond to any message issued by the BC. Indeed, it does not require a transmitter and may be conceived as a *receive-only* terminal. However, this is not generally the case, since monitors usually have RT capabilities (see further below) to allow the BC to check upon the status and health of the device, as well as to command operational modes. The primary application of BM is to collect data for analysis, storage, or transmission and eventually to serve as a back-up BC in the event of a fault of the primary BC. Employing a BM as a back-up BC has its advantages. In fact, by listening to all messages being transmitted on the BUS, the device can produce a picture of the environment within which it is operating. Consequently, it can continuously watch over the status of the network, thus reducing start-up times if a sudden need for stepping in as BC arises.

An RT is defined as "any terminal not operating as BC or as BM." Such definition implies that RTs cannot initiate communication on the BUS, but can only transmit when specifically commanded to do so. RTs are identified by a unique address that allows the BC and other RTs to route messages to it. They may implement the necessary logic to behave as BC in the case of a fault in the primary BC or if the network system adopts dynamic BUS control. State of the art 1553 terminals have sufficient processing power to be implemented as fully integrated components. This means that they can be employed as any of the three infrastructure elements of a 1553 network, depending on the situation. This allows for flexibility and cost-effectiveness. RTs can be subdivided into two subcategories: embedded RTs and stand-alone RTs. The former consist of interface circuitry and functional elements being integrated inside a sensor or subsystem, which is directly connected to the 1553 data BUS. Generally, such devices are not designed to serve as a BC. If the sensor has sufficient processing power itself, however, it might serve as back-up BC if no other device is available in the network. The latter are devices designed to provide a 1553 network interface to one or multiple systems which do not have 1553 communication capabilities. As such, they are the only devices which serve as a multiplexing/demultiplexing units in the network infrastructure.

Fig. 1 illustrates a typical 1553 BUS network configuration.

2.1. *Network architecture*

The simplest 1553 network topology is obtained by interconnecting a BC, a BM and a certain number of RT's to a BUS infrastructure. The interconnection of the terminal interfaces with the copper wiring is realized through couplers in order to adapt the impedance. This is the most commonly used architecture, as devices are all directly connected to a single level BUS. Such interconnection scheme can provide redundancy by implementing multiple wiring. Each wiring is identified as a *channel*. At a network level, the topology remains unvaried.

A more complex architecture can provide for multiple level distribution of elements, extending the concept of the single level BUS. Alternatively, a hierarchical organization of systems can be devised. In such schemes, multiple BUSes are present; each of them interconnects a cluster of devices. This is a practical solution for achieving functional partitioning, or for isolating devices which require stringent levels of BUS loading to operate correctly. Data transfers and relationships among BUSes are achieved through control schemes. Two schemes are generally adopted: *equivalent levels of control* and *hierarchical levels of control*. The former implies that the interconnected BUSes are formally independent and coordination is required only when data is to be transferred from one domain to the other. All other operations - i.e. error handling, recovery schemes, etc. - remain prerogative of the individual BUS. The latter control scheme, on the other hand, presumes inequality of the interconnected BUSes. As a consequence, a hierarchical relationship is established among the networks. The controlling BUS system is generally referred to as the *global BUS*, whereas the subordinate BUS systems are referred to as *local BUSes*. This configuration is usually adopted to control a weapon BUS from the avionic BUS, or to have a single system,

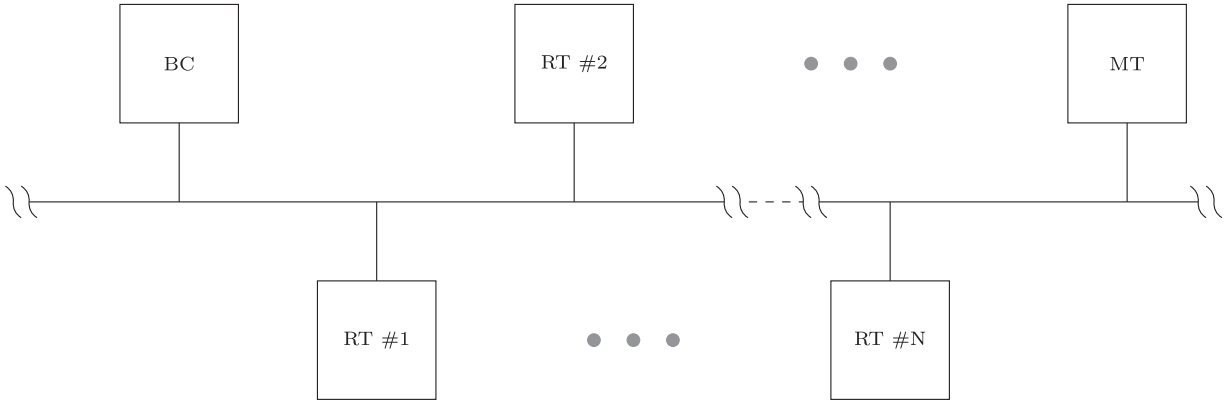

Fig. 1 – MIL-STD-1553 network example.

composed of several subsystems, interface an internal 1553 network.

2.2. *Word formats*

The 1553 standard adopts asynchronous TDM (Time Division Multiplexing). That is, each device utilizes its own clock source for transmission; the receiving device will extract the clock from the received signal for demodulation. The bit streams are modulated through PCM (Pulse Code Modulation). The encoding scheme is Manchester II biphase, which favors steady clock extraction from the signal.

Data to be transferred on the BUS are organized in 16-bit-long words. These are the building blocks of communication on 1553 networks. The 16 bits are preceded by a 3-bit-long synchronization signal and followed by a single parity bit, totalling 20 bits per word. The standard defines three word types: *command word, status word* and *data word*.

Synchronization is achieved through the *sync* signal. It is required that the *sync* waveform be a an invalid Manchester encoded PCM, in order to have a unique pattern which cannot be obtained through particular sequences in the body of the word. As it has been mentioned, the length is equivalent to the duration of three bits. The waveform is identical for *command words* and *status words*, whereas it is different for *data words*. In the former case, the first half is *high* and the second half is *low*. The transition occurs at 1.5-bit time from the beginning of the *sync* signal. For *data words*, the waveform is opposite: *low* in the first half and *high* in the second half. By identifying either of the two synchronization signals, the receiver is able to determine the beginning of each word and begin demodulating the message.

The *command word* format is used to control and manage the transfer of information on the BUS. As previously mentioned, communication on 1553 network is established and initiated only by the BC. *Command words* are the means through which the BC performs such tasks. *Data words* are used for transmitting data from any terminal to any other terminal, as commanded by the BC. *Status words* are transmitted by RTs in response to every *command word* received to provide the BC the internal state of the RT. At every reception of a valid command word, the status word will be reset.

Mode codes can be used to perform a wide variety of actions on the bus hardware (synchronization, dynamic bus control, transmission, transmitter shutdown, etc.).

2.3. *Timing and framing*

The transmission of message sequences identifies frames on the BUS. These are determined by the periodicity of information exchanges. In fact, certain devices require constant communication with other devices in order to operate. That is the case, for example, of any device that requires inertial platform data to perform its task. The latter will require information such as altitude, attitude, position, etc. at a precise frequency - i.e 50 Hz. Therefore, the message format associated with such exchange will have to be reproduced periodically on the BUS. Other types of exchanges, on the other hand, are intrinsically aperiodic. That is the case, for example, of target coordinates being issued from the main computer of the aircraft to a store (weapon or sensor).

Therefore, the BC has to be aware of which messages need to be transmitted periodically and which is their required time interval. In addition, it has to reserve BUS resources for the transmission of asynchronous messages. It is thus required to perform scheduling, which is achieved through structures such as *minor frames* and *major frames*.

Frames are sets of message slots, assigned to periodic messages or reserved for aperiodic ones. A *major frame* identifies the overall periodic structure of patterns on the BUS. A certain period is established upon setup and it identifies the *major frame* repetition time. Usually this corresponds to the period of the transaction which has the lowest frequency. As an example, spacecraft BepiColombo, which employs 1553 data bus communication, is set on 1s*major frame* repetition time.

Each *major frame* is subdivided into a certain number of *minor frames*. The repetition time of the latter is directly correlated to the period of the highest frequency transaction occurring on the BUS. In fact, the period of *minor frames* on a BUS has to be less than the minimum period among cyclic mes-

sages. If such condition is not met, then frame overflow[2] will occur systematically.

The transmission of messages on the BUS is characterized by gaps. In fact, multiple words are sent contiguously by a terminal. A certain amount of time, however, occurs before responses to *command words* are sent back. Such interval is called *response gap*. For this reason, generally 1553 devices adopt a timeout scheme to prevent deadlocks: if a terminal does not respond within timeout, the recipient will not wait further and frame execution will continue. Common values for response timeout are of the order of 16 μs.

In addition to *response gaps*, the time interval between the transmission of adjacent message formats is defined *intermessage gap*. This has a minimum value of 4.0 μs and is formally identified as the time interval between the mid-bit zero crossing of the last bit of the preceding message to the mid-zero crossing of the next *command word sync* waveform.

3. Cyber attacks on MIL-STD-1553

Given the specificity of the protocol, a rogue device connected to a 1553 network has the possibility to perform a significant, though limited, plethora of attacks. In fact, the multiplexing and scheduling scheme, which delegates all control to the BC, creates a single point of failure which can be targeted and exploited. If a device is able to inject *command words* in the network, other 1553 devices are not able to distinguish such messages from legitimate ones. Such action can have severe consequences, which range from preventing the exchange of information among systems to providing them with fictitious or malicious data. Depending upon higher level vulnerabilities, the attack might propagate to other domains through tactical networks. In fact, modern day weapon systems are performing more and more as interconnected data processing units and the information exchange is vital for the conduction of the mission. Therefore, a viable logic-level cyber attack that is intrinsically based on the MIL-STD-1553 protocol specification would exploit the centralized multiplexing scheme by statistically learning the frame scheduling and executing malicious actions at convenient bus times.

The types of attack that are relevant to a 1553 infrastructure, and thus will be the subject of further consideration, are the following:

- Direct Denial of service (through band consumption);
- Exploitation (through spoofing and message injection).

We will discuss the opportunity of performing each of these attacks on a 1553 architecture and what consequences a cyber attack delivered through each technique might have on the overall system.

3.1. *Denial of service (DoS)*

This cyber attack aims at blocking communication on the BUS. It makes use of low level jamming to consume BUS resources and prevent data exchange among subsystems. The possible consequences are evident. As an example, a guided weapon which requires the coordinates of the ground target in order to be fired, could be isolated from the system. In this case if communication cannot be established, the weapon system will not be released. For a 1553 network, communications among terminals can be blocked by randomly injecting messages which will cause collisions on the BUS. If the cyber device has physical access to the BUS, the implementation is straightforward. Indeed, the most basic implementation would be to transmit words in correspondence to standard messages that occur on the bus, according to *major frame* and *minor frame* patterns. Such behavior will cause interference and noise and induce errors in decoding. The BC might notice the impossibility of communicating on the bus and take action to try and restore the link, such as switching transmission on a redundant channel. However, if the cyber device has access to each channel, the attack can be performed simultaneously on each of them. The BC might report the situation to the operator - i.e. the pilot, or the navigator. One may therefore be interested in achieving a DoS through a *minimal* injection of noise, in order to reduce the chances of being detected.

Furthermore, DoS can be performed in a wide variety of ways. More subtle and "elegant" methodologies can be identified, which enable the attacker to remain undetected. One of these techniques is to have the cyber device cause a collision with the *command word* being periodically transmitted by the BC, which will thus be never decoded by the RT. Simultaneously, it might also respond on behalf of such RT, providing false information. Indeed, if the *command word* collides on the BUS, it will fail message validation performed by the RT and the latter will not provide any response. As a consequence, if the cyber device provides the expected response, the BC will be misled into thinking that the communication has been successful. This solution implies an *a priori* knowledge of the target message characteristics, so that the cyber device can properly compose the status word expected by the BC.

3.2. *Spoofing*

This attack consists of exploiting faults and vulnerabilities in the logic of interconnected subsystems. Indeed, the 1553 data bus protocol has been engineered to be a high-reliability means of communications. Security has not been addressed as a major player in the process of communication and therefore devices can be found vulnerable to exploitation. As an example, a cyber device with physical access to the 1553 bus can detect an idle time on the bus (i.e. when neither the BC, nor the RT's are transmitting data) and initiate a message exchange in that interval. Every other device will assume that communication on the bus has been initiated by the legitimate BC, as the protocol does not provide any means for identifying a terminal operating as BC.[3] Therefore, the cyber device would be able

[2] Frame overflow occurs when the update timer for a certain information to be sent on the BUS triggers before the *minor frame* has ended.

[3] One means for establishing the identity of a terminal would be to analyze the spectral signature of the signals it transmits at

to provide terminals with malicious information. Considering for example the aforementioned case of a guided weapon, the attacker might provide the weapon with coordinates which do not belong to the designated target of the mission, overriding any previous data. No device would know that the coordinates stored in the weapon system are counterfeit, while the BC would simply ignore such message exchange. The consequences of such attack can be devastating: they might range from the weapon not being fired, to the device guiding itself not towards a target-free area.

The main issue to face when performing this type of attack is to determine the idle time on the BUS when to transmit. This translates into predicting intermessage gaps or unused aperiodic message slots. Indeed, a cyber device would have to foresee with a certain level of assurance whether at a certain time interval the network will be free from legitimate communication. To perform such analysis, it has to implement algorithms based on signal processing and on pattern recognition, which are able to evaluate the underlying traffic characteristics and perform adequate predictions.

The implementation of both of these attacks will be the objective of the following sections. We will identify a set of basic features that will be implemented on the Cyber Device. The next section will explore the modeling process of the device, focusing on the architectural design and implementation choices, while Section 5 will illustrate its software implementation.

4. Design

In order to design a Cyber Device capable of performing attacks on a 1553 network, we have embraced so-called *model-based systems engineering* techniques. In particular, we have adopted the ARCADIA (Architecture Analysis & Design Integrated Approach) methodology (Voirin, 2017). ARCADIA approaches development by focusing on functional analysis, system design, justification of architectural choices and continuous verification. It has been standardized in 2007 as an AFNOR (Association Française de Normalization) published norm. The methodology is structured upon five perspectives at each modeling level: (i) Operational Analysis; (ii) System Analysis; (iii) Logical Architecture; (iv) Physical Architecture; (v) Product Building Strategy. In the following, we will briefly touch upon all of the above mentioned phases, with the exception of the product building strategy, that goes beyond the research goals we want to focus on in this paper.

4.1. *Operational analysis*

At an operational level, the starting point of modeling has been the identification of a realistic scenario.

Since the operational needs have prevalent influence on the implementation of the system, sufficient concern was put at this stage. The scenario, therefore, has been identified as follows:

> A well-deployed intelligence network has provided technical information regarding systems of an aircraft employing a 1553 avionic infrastructure. Such information is utilized by a hacker to configure a Cyber Device capable of performing a series of cyber attacks on the network. The configured device is handed over to a defector, who - having physical access to the target aircraft - proceeds with the deployment, circumventing security policies of the military installations. When the asset is on an operative mission, the Cyber Device starts monitoring traffic on the BUS and performs analysis in order to identify the most convenient instant in which to perform the predetermined cyber attack. A trigger condition causes the device to start the message injection, hampering the functions of the aircraft by targeting a specific system or the entire network. When the system is being targeted by the Cyber Device, the aircraft has limited functionalities - i.e. unavailable weapons, unavailable EW[4] countermeasures, etc. In this scenario, we suppose that the EW countermeasures are rendered unusable by the device. At this point a physical menace - i.e. IR-guided[5] missile - is shot against the aircraft, whose hindered systems are unable to detect and deceive. The asset is therefore hit and shot down.

The scenario is based upon the following assumptions:

- A hacker receives enough intelligence information to be able to configure a Cyber Device designed to perform attacks on an aircraft avionic network. The information includes technical data such as message exchange and ICDs (Interface Control Documents) and the network topology of the interconnected systems;
- A defector is granted physical access to the target aircraft and is therefore able to physically deploy the Cyber Device, connecting its interface to the 1553 avionic BUS.

The Cyber Device, upon deployment, is able to both listen and send messages on the 1553 BUS. Upon these hypotheses, the identified *actors* and *entities* of the scenario are therefore the following: *Hacker, Defector, Cyber Device, Military Security Services, Asset (Aircraft & Pilot), Command & Control Chain, Kinetic Menace*.

The identified Operational Capabilities describe what the system has to achieve at its highest level of abstraction. According to the scenario, they have been identified as the following: *Configuration, Deployment* and *Execution*.

Fig. 2 illustrates the involvement of *actors* and *entities* in each *Operational Capability*.

At this stage, the following operational activities have been identified: *Intelligence Information acquisition, Cyber Device configuration, Infiltration, Security Policy empowering, Cyber Device deployment, ATO (Air Tasking Order) Acquisition, Mission planning, Mission performing, Command & Control, Kinetic Attack performing*.

Fig. 3 illustrates the attribution of such functions to each of the entities and actors previously mentioned.

a physical level. Such fingerprinting procedure is not included in any release of the standard.

[4] **E**lectronic **W**arfare
[5] **I**nfrared-guided.

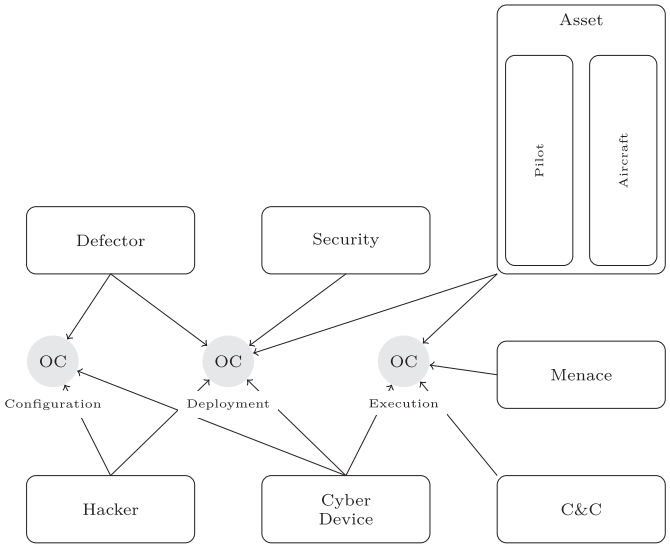

Fig. 2 – Operational capabilities.

The design process clearly indicates what are the constraints that the subsequent design phases will have to comply with. The modeling activity proceeds with *System Analysis*, described in the following section.

4.2. *System analysis*

Transitioning from *Operational Analysis*, this step aims at refining the Operational Activities into System Functions. These are then attributed either to the Cyber Device or to external actors. Such analysis, therefore, allows for a definition of the functional boundaries of the device. The methodology dictates a definition of functional exchanges describing the interaction among external actors and between each actor and the Cyber Device. These exchanges, at a lower level, will be realized by data exchange or physical interactions.

The aforementioned Operational Activity refinement leads to a breakdown of functions, as well as to a mapping phase. With respect to this last point, some activities have been mapped one-to-one with System Functions, while others have been subdivided into multiple System Functions.

The final step in the process has been function attribution, which was performed as shown in Fig. 4.

As it can be observed, the following functions have been attributed to the Cyber Device:

- BUS Interfacing;
- Environment data acquisition;
- Cyber attack configuration;
- Cyber attack execution;

BUS interfacing This function collects all tasks regarding BUS interfacing. Every duty regarding either writing or reading messages to and from the BUS is taken into account at this level. Isolating it from the remaining tasks, we are able to achieve for a higher level of abstraction and simplification at the final stages of implementation.

Environment data acquisition In order to properly perform message injection on the 1553 infrastructure, the Cyber Device must be aware of the traffic characteristics on the network. Therefore, an acquisition utility is needed in order to adequately process data that are to be used when executing *cyber* attack sequences. The contribution of this system function is crucial to determine timing and logic of any attack.

Cyber attack configuration This system function constitutes the core of the Cyber Device. It takes as inputs the user configuration and the environment library and generates attack sequences to be performed at run-time.

Cyber attack execution This function is demanded to execute the live attack sequences on the network. This operation requires exact timings and adequate logic, in order to cause faults in the aircraft system. It provides *BUS interfacing* with the message formats to be injected and receives potential feedback from the 1553 infrastructure.

Having defined which are the system-level functions that device is responsible of and what are its interactions with external actors, the modeling process evolves through *Logical Analysis*.

4.3. *Logical analysis*

At this point, the design phase focuses on the architectural design of the software. We have already defined every aspect which is to be taken into account in architectural design. In particular, transitioning from *system functions*, the *Logical Analysis* aims at identifying components and sub-components which will constitute the software elements.

Further functional breakdown is performed in order to properly define sub-component boundaries and interactions among them. Such analysis leads to the architectural design illustrated in Fig. 5.

Configuration component This sub-component provides a configuration utility for the user. It consists of a UI (User Interface) that accepts parameters related to the attack configura-

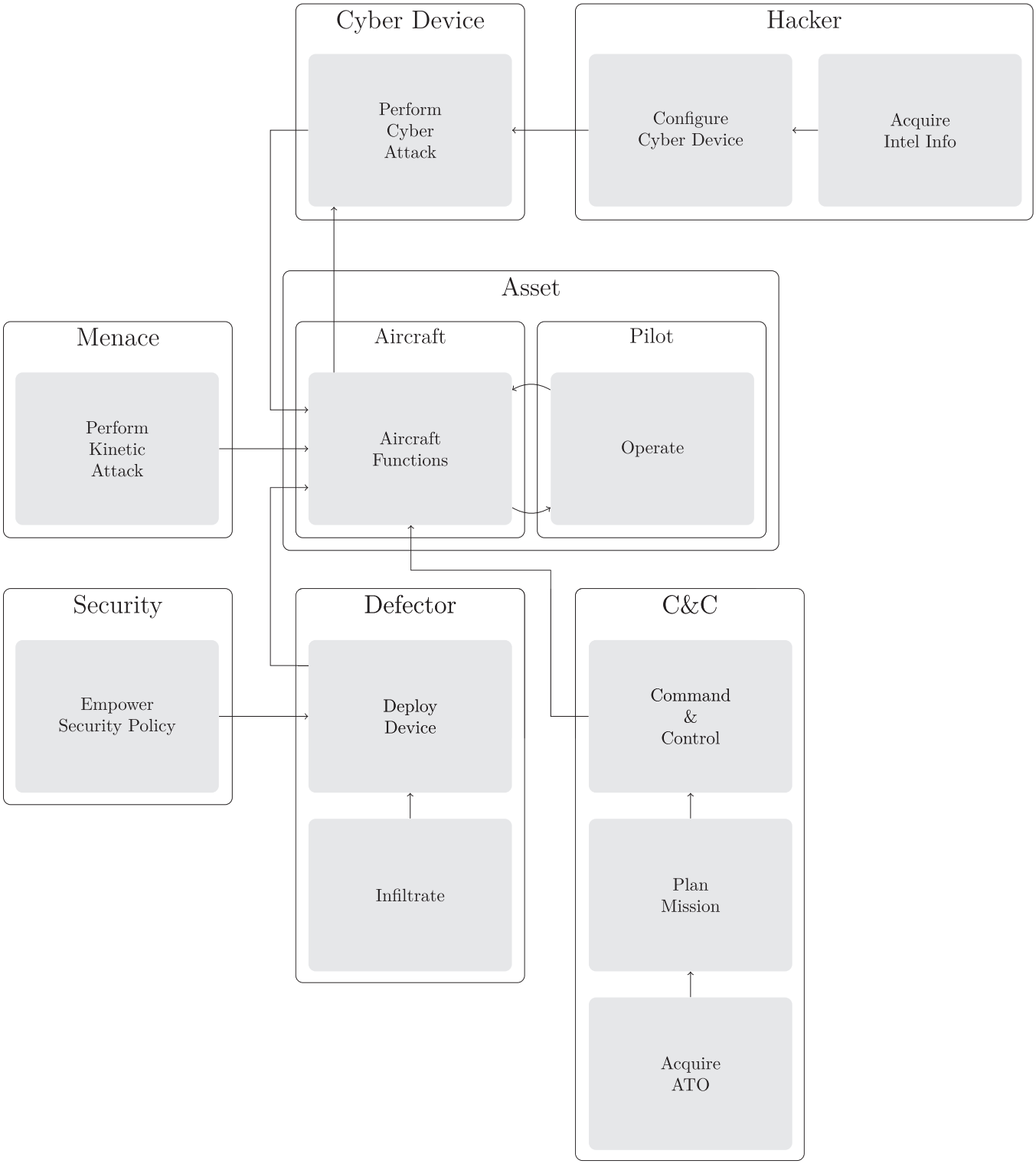

Fig. 3 – Operational context.

tion (i.e., type of attack, target, triggering information, etc.). It must generate a configuration file that other sub-components may access at run-time to gather necessary information to carry out the attack.

Sensing component This sub-component implements the algorithms necessary for the processing of traffic data. By passively listening to messages on the BUS, this element must be able to perform a thorough analysis and to extract information such as frame period, message repetition frequency, probability distributions, etc. Such information is then organized and stored into an *environment library* file that will be used at run-time in order to perform the attack.

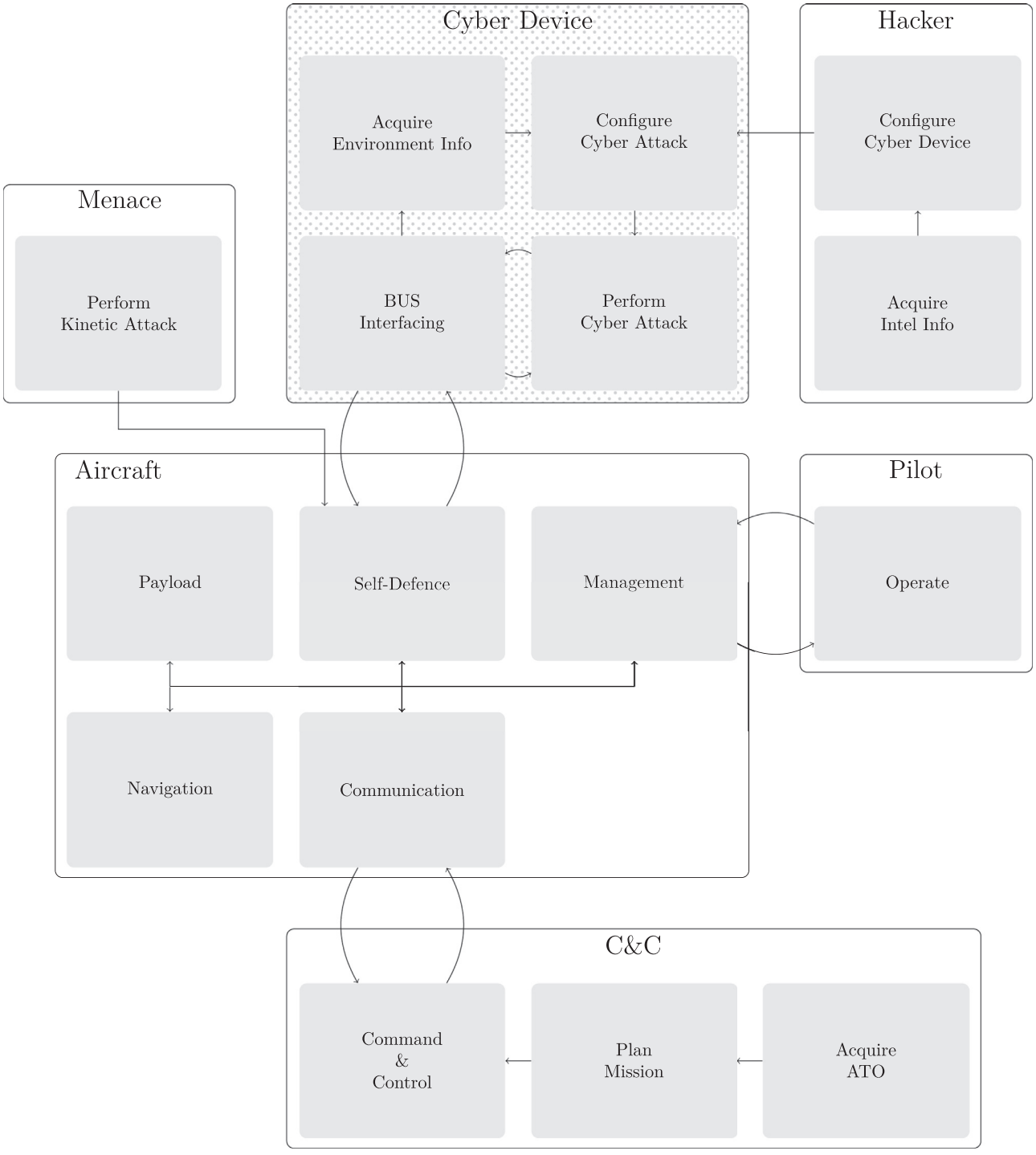

Fig. 4 – System architecture.

Bus interface component This sub-component incorporates both the physical interconnection of the device to the 1553 infrastructure and the firmware libraries which provide data-link layer services. The firmware will interface the software layer through provided API (Application Programming Interface). The lowest level BUS interfacing will be demanded to COTS (Commercial Off The Shelf) boards and pertaining software.

Cyber attack component This sub-component is required to perform attack sequence execution and coordination. In order to achieve this task, it takes advantage of configuration file and of the *environment library* generated by the other sub-components. It interacts with the *BUS Interface Component* to perform message injection and to receive feedback from the 1553 infrastructure.

Having unfolded and performed logical analysis, software implementation is now within reach. The last step is to trans-

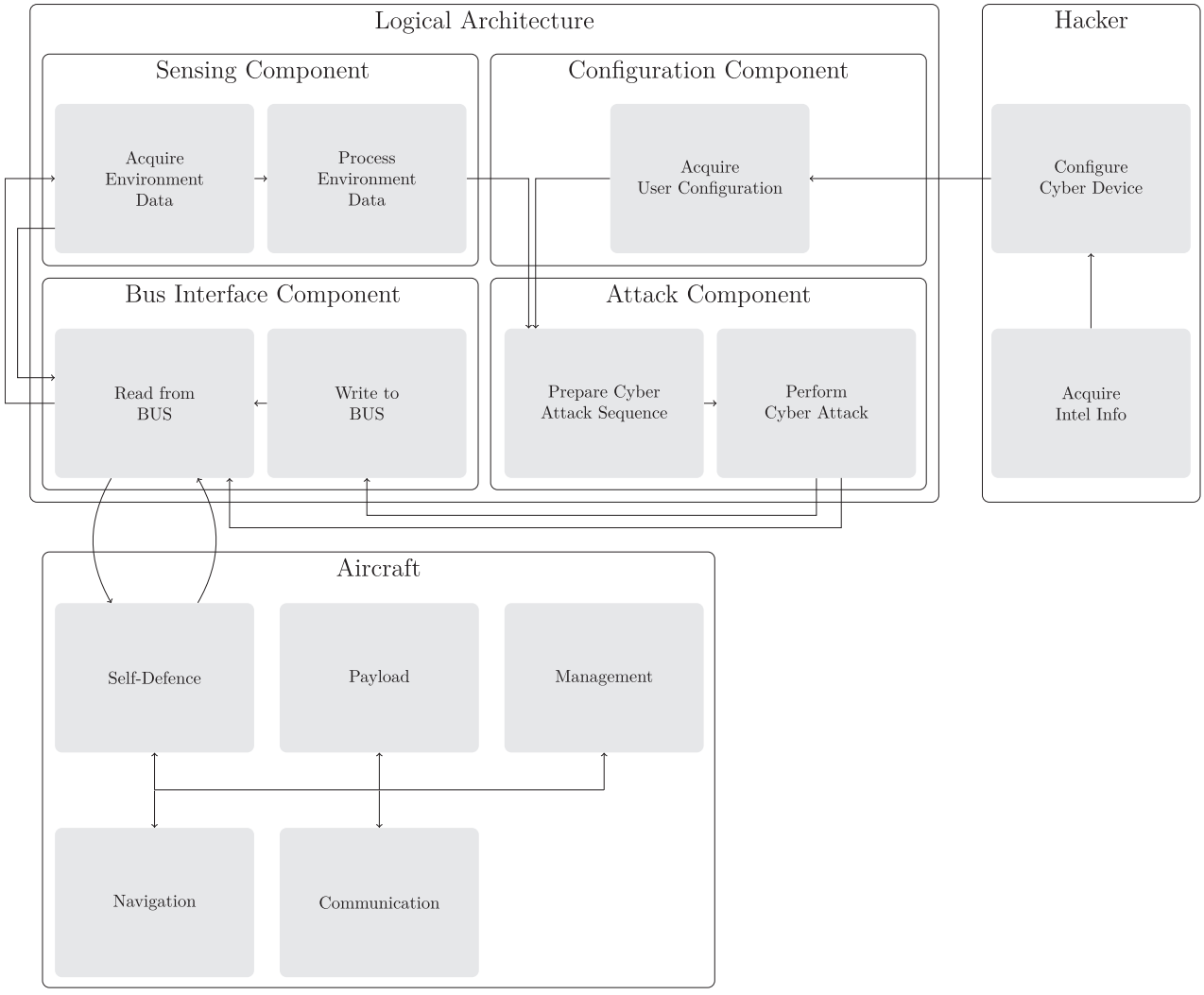

Fig. 5 – Logical system architecture.

late the designed system into a functioning device, taking into account peculiarities and details which pertain to a software-level abstraction. In the following section we will therefore describe the transition from architectural analysis to hardware and software implementation.

5. Hardware and software implementation

This section will cover all aspects regarding the transition from the architectural analysis to the software implementation. The development of a tool that is required to interface a network at a low level of the newtorking stack must necessarily take into account existing hardware constraints. Therefore, before introducing the software, we will describe what hardware has been acquired and used within our study.

5.1. *Hardware*

A MIL-STD-1553 interface is required to take care of real time BUS interfacing and physical level protocol execution. A variety of boards, rigs and transceivers are commercially accessible and are provided with detailed documentation and supporting libraries. These are industry-level products characterized by high-end technical specifications. The most well known suppliers include Sital Technologies, United Electronic Industries, Data Device Corporation and Abaco Systems, among the others. For our project, we leveraged Data Device Corporation boards, such as the BU-67103U and the BU-67106K. Such boards are provided with a rich and thorough API, namely AceXstreme SDK (Software Development Kit) and all related documentation. Such a library is multi-platform[6] and is written in the C/C++ programming language.

In the following section we provide a brief overview of the features of the boards. The reader is referred to Data Device Corporation (2010) for further information on the AceXtreme SDK.

[6] AceXstreme SDK supports the following OSs: Windows, Linux, VxWorks, Integrity.

Fig. 6 – DDC BU-67103U (left) and BU-67106K (right) MIL-STD-1553 boards.

5.1.1. *DDC boards and AceXtreme SDK*

Data Device Corporation provides a broad selection of MIL-STD-1553 computers, boards and components. Their solutions are relevant for numerous applications: military, aerospace, industrial, etc. Our application is intended to make use of 1553 board interfaces. As anticipated, we have used the following boards: (i) BU-67103U; (ii) BU-67106K (Fig. 6).

Both of them provide interfaces for single or dual-redundant 1553 networks and can be programmed to operate as BC, Multi-RT[7], or Concurrent-BM.[8] The difference between the two devices lies in the fact that the former provides a USB interface, while the latter a PCI-e interface.

5.2. *Software*

The software development process has taken advantage of industry-level solutions which allowed for reliable and user-friendly coding. We have deliberately decided to make our code as multi-platform as possible, allowing easy portability from Windows to Linux.

The initial problem to be managed at this stage was to consider the extent of the AceXtreme SDK. In fact, one of the requirements that we set for our device was that it has to be as multi-platform as possible. Therefore, we have explored solutions that allow for isolation of most of the code from the usage of a specific API. In this case, the architecture design described in Section 4 came to the rescue. In fact, we had already attributed one of the four *logical components* constituting the Cyber Device to be tasked with all IO and interfacing issues. At this stage, such component has been implemented as a *class* - namely *IoClass*. This is meant to "wrap" the AceXtreme SDK functions and provide a common interface for accessing the 1553 network to the remaining software components. Indeed, such choice allows for other APIs to be used in the future, while keeping the same interface. As a consequence, there will be no further code changes to be done in other parts of software. It is a fairly basic principle of code reuse, but it is a powerful solution nevertheless.

The second problem to be tackled was the design and implementation of basic data structures to support the implementation of the protocol. In fact, the AceXtreme SDK is provided with pre-configured data structures, but they are limited to some *structs* and functions and are not as easily manageable. Therefore we realized a custom set of data structures and grouped them into an ad hoc defined *stdCyberDev namespace*.

The third problem was to identify the cyber attack features to be implemented on the Cyber Device. The identified attack threats were *Denial of Service* and *Exploitation*. As already stated, the former is to be achieved through physical jamming of the BUS infrastructure and can be performed either against the whole system or against a specific message. On the other hand, we also observed how the latter attack can be accomplished through message injection.

Taking this into account, we have identified the following features to be implemented: (i) Jamming; (ii) Injection; (iii) Selective Injection.

Jamming This basic attack consists of injecting massive amounts of messages in the 1553 network in order to generate physical noise and prevent communications. As stated in Section 4, the attacker may be interested in obtaining an occupation percentage lower than 100%- ideally around 30%. In fact, lower levels of BUS loading allow for power management, lower probability of being detected, and cost-effectiveness. Therefore, the configuration phase of such attack will allow the user to specify the desired BUS loading to be obtained.

Injection The most powerful feature of the Cyber Device will allow to inject messages in the 1553 network, minimizing collisions with underlying traffic. To achieve such result, the device needs to perform proper analysis in order to predict the time instant in the major frame period where an "optimal" injection would take place. The degree of "optimality" will be asserted based on the collision probability. In principle, the injection needs to be performed on a forecast basis: the software needs to exploit the periodicity of the traffic to anticipate the injection instant, as the 1553 timing constraints do not permit complex calculations to be performed at run-time. This is paramount, as the 1553 time scale is about three orders of magnitude less than that of software execution. In addition, OS scheduling might cause jitter that could prevent the correct execution of such attack. The impact of scheduling will be assessed in Section 6.

Selective jamming It is a powerful implementation of the Cyber Device software. It is meant to create continuous collisions with a specific message in order to prevent data from being exchanged among two or more specific systems on the network. The impact of this feature is relevant if it is included in a more complex cyber attack chain which aims - for example - at replacing the payload data of a periodic message.

[7] Multi-RT mode allows the board to execute up to 31 Remote Terminals simultaneously.

[8] Concurrent-BM mode allows the board to perform bus monitoring operations while executing RT or BC actions.

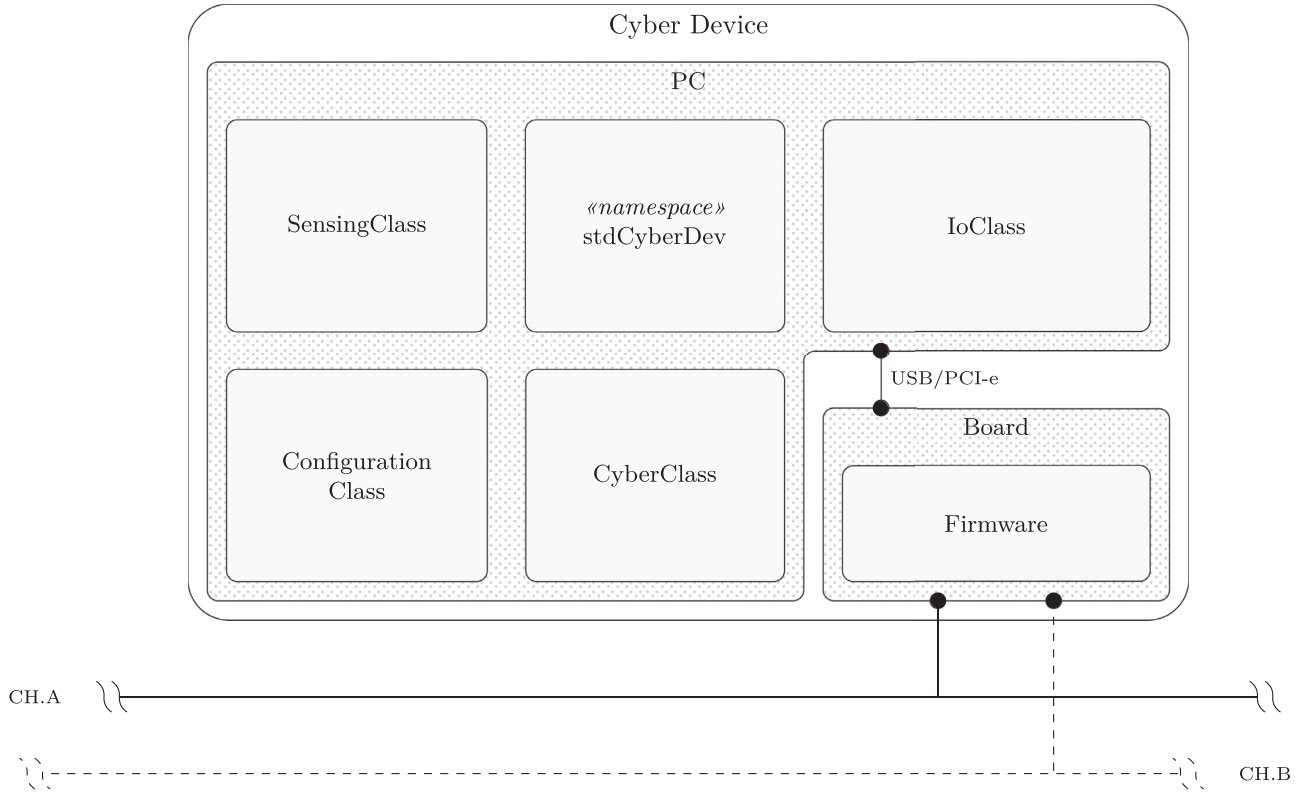

Fig. 7 – Cyber device architecture.

Suppose, for instance, that an intruder wants to provide a system with fictitious platform data in order to cause faults or malfunctioning. Since platform data are generally provided on a periodic basis - i.e. 50 Hz - the cyber attack must be composed of two simultaneous actions:

- The injection of fictitious platform data messages on the bus at the required rate, whilst minimizing collisions;
- The prevention of legitimate platform data from reaching the destination. This could be achieved through *Selective Jamming*.

In addition, the attacker may be interested in preventing specific Status Words from being received by the BC. This can have diverse consequences, depending on BC and RT logic. For example, the BC - upon several Status Word reception failures - might consider a certain RT as being offline and interrupt all communications with it. Another effect of interest might be the BC diverting such traffic on the redundant BUS, thus reducing BUS loading on the main line. As always, the desired effect needs to be tailored for the specific system.

Intrinsically, realizing a *Selective Jamming* requires precise timings, with a granularity as fine as 10 µs–20 µs. Such is the duration of a single 1553 word. Demanding such precision to an OS is essentially naif. A *firmware* execution is inescapable for achieving acceptable results. In our case, we attempted to exploit DDC boards *firmware* functionality to prove the feasibility of the attack. In particular, the steps to be performed by the Cyber Device are the following:

1. Identify the target message with respect to its position within the major frame period;
2. Schedule an appropriate frame hierarchy, where messages to be transmitted are placed in correspondence of the target message in the major frame period;
3. Activate the major frame execution, assuring the synchronization of the latter with the underlying traffic.

Since major frame execution is a *firmware* level task of the DDC boards, the timing deltas among scheduled messages will be assured with fine granularity. Nevertheless, the software will be demanded to manage frame synchronization. That is, to assure that major frame execution began in sync with the underlying traffic. This is obviously not guaranteed by DDC boards and no function of the SDK can be exploited to achieve such result using DDC hardware.

The remaining components of the code are one-to-one mappings of the *logical components*. That is, the *Cyber Attack Component* has been implemented in *CyberClass*, the *Sensing Component* in a *SensingClass* and the *Configuration Component* in a *ConfigurationClass*. The execution of the program is orchestrated by a *main* function. Fig. 7 provides an overall picture of the Cyber Device software architecture.

In the next paragraphs we will provide a thorough analysis of every software component, beginning from the *stdCyberDev namespace*, which constitutes the foundations of the architecture, and subsequently building up. We will give deep understanding of how the single objects interact with one another, not forgetting to map the relationships with the modeling process that has been described in Section 4.

5.3. *The* stdCyberDev *namespace*

In this section we are going to explore the *stdCyberDev* package, which contains all data structures and elements supporting the software. We will now briefly examine each one of them.

ConcurrentQueue This data structure is a thread-safe queue. It is necessary for multi-threading operations, such as reading messages from the BUS and simultaneously save them to a local file. Concurrent threads that either produce or consume items from the list will lock on a *mutex* prior to any operation.

MsgClass, frame, MinorFrame, MajorFrame These data structures implement the equivalent MIL-STD-1553 definitions in a OOP framework.

In particular, *MsgClass* provides an abstraction of all message exchanges that can take place in a 1553 infrastructure. The AceXtreme SDK does not provide full integration of data structures for messages received from BM and messages to be sent by BC. Such distinction is particularly evident in the API function calls. The Cyber Device software, taking advantage of *MsgClass*, achieves a full integration of the two cases. For this reason the class provides two distinct parametric constructors: the former takes as input arguments the atomic details of the transaction (RT addresses, sub-addresses, word count, etc.), while the latter requires a single argument, that is a pointer to an IRIG (Inter Range Instrumentation Group) *Chapter10* packet. IRIG Chapter10 is a specification for avionic BUS monitoring. In particular it standardizes the output of the BM, providing a specific packet format for sending monitor data to other devices. For more details about IRIG Chapter10, see Range Commanders Council (2007).

Finally, it is important to observe that an orthogonality concept has been introduced in order to handle message sequences as signals. This will be particularly useful when implementing statistical analysis functions. Such orthogonality is assessed through a scalar-like product between messages, whose definition is the following:[9]

Definition 5.1. Let *A*and *B*be two distinct messages.

If $A = B$,then:

$$A \cdot B = 1$$

Alternatively, if $A \neq B$,then:

$$A \cdot B = 0$$

Equality between messages is assessed based on the command word: if two messages are characterized by the same command word then they are assumed to be equal.

The same rationale applies to the *MinorFrame* and *MajorFrame* data structures, which are polymorphic *classes* inheriting from a common *Frame* interface. These objects are essentially enhanced containers that store either *MsgClass* objects or *MinorFrame* objects, respectively.

Ch10DataPacket This data structure is required in order to provide a level of abstraction to the IRIG Chapter10 packets acquisition process. The AceXtreme SDK requires that a memory buffer be allocated every time a packet is received. Such memory buffer must be of sufficient size to contain the whole packet, which might be of variable size. At the end of the acquisition process, the buffer must be deallocated to prevent memory leakage. Ch10DataPacket implements "Resource Acquisition Is Initialization" (RAII) technique, which makes memory management transparent to the user. The memory buffer is allocated at object initialization and deallocated when its destructor is called.

Timeline, PeriodExtractor, BusEnvironment, synchronizer

These data structures have been designed to implement the traffic analysis algorithms. The output of the process is a library to be used at run-time for cyber attack execution. These *classes* make use of the data structures already introduced and will now be explored individually.

The *Timeline* object, as the name indicates, provides an abstraction for a generic timeline. Essentially, it is a container that associates a numerical value of type *Float64* with a time instant of type *Uint64*. The data structure provides methods to access elements either by their corresponding time instant or by their progressive index, as a typical array would.

The *PeriodExtractor* is a *functor* object.[10] Its task is to estimate the major frame period of underlying traffic. To achieve such goal it makes use of *MsgClass timeTag* attribute, which represents the time instant at which each message was seen on the BUS. The way in which the period estimation is performed is peculiar. In fact, in order to exploit conventional knowledge about signal period estimation, we had to make some considerations and preliminary steps.

We assumed the message acquisition sequence as if it were a signal of elements in a finite space: that of 1553 command words. Each of the 65, 536elements of the space is defined orthogonal to each of the remaining ones. The concept of orthogonality illustrated in Definition 5.1 makes us of this concept: two messages are considered *equal* if they share the same command word.

The signal extracted from the sequence is stored in a *Timeline* object. However, it cannot be considered uniformly sampled as messages are received in a non-uniform manner. Indeed, *MsgClass timeTag* values are not evenly spaced. In order to take advantage of signal processing tools, we are required to manipulate the data in order to achieve uniform sampling. Indeed, we are artificially padding the signal with *null*,[11]equally spaced messages.

The sampling rate is chosen as the reciprocal of the minimum time distance between two messages in the original sequence. All these operations are performed by method *padSignal()*.

[9] It is important to observe that such definition is not rigorous. The scalar product does not fulfill all requirements to be considered as such. For example, it does not have the linearity property. This is why it is scalar-like and not strictly scalar.

[10] Functors, or function objects, are constructs that allow an object to be invoked with the same syntax as a function call. The advantage of using functors is that, being objects instances to all effects, they possess a state and behave in all respects as an object would.

[11] *Null* messages are those having command word equal to 0×0000.

The subsequent steps are straightforward: having a uniformly sampled signal, the *functor* computes its autocorrelation function. Autocorrelation provides a measure of similarity of a signal to a shifted version of itself. Maximum values are always obtained at zero-shift. For a periodic signal, however, maxima are also observed at period-shift. The algorithm exploits these characteristics to estimate the periodicity of the message sequence. Indeed, the latter rarely displays a strictly periodic structure, due to the presence of asynchronous messages and jitter. Nonetheless, these factors can be considered as additive noise disturbing an underlying strictly periodic signal. By extracting the maximum value of the autocorrelation - excluding zero-shift - we obtain an estimate of major frame period. The user can tune the length of the sequence to be acquired for period extraction: the longer the sequence, the more accurate the estimate. Too long sequences, however, may result in excessive delay, since the time-consuming autocorrelation computation has a n^2computational complexity.[12]

BusEnvironment constitutes the output object of the traffic analysis process and will be used by the device upon performing the cyber attack. It contains all information needed for the software to predict traffic on the network. Further details will be provided in Section 5.5, where the algorithm for generating the library is explained.

AttackClass This *class* gathers all the information required to perform a cyber attack on the BUS infrastructure. As previously reasoned, the *class* implements the following attacks: (i) Jamming; (ii) Injection; (iii) Selective Jamming.

Depending on the attack type, some further information is required. In particular, for *Jamming*, the *class* requires the BUS loading percentage to be achieved. For *Injection*, on the other hand, the user is required to specify the message that needs to be injected. Additionally, he/she may specify if the injection is to be triggered by the transmission of another message on the BUS. Lastly, *Selective Jamming* allows to specify the message to be targeted.

5.4. *IoClass*

This *class* has already been introduced. It provides abstraction and simplification of the AceXtreme SDK, allowing the rest of the software to interface with the 1553 BUS in a seamless manner. Future developments may provide the implementation of *IoClass* for additional APIs. The *class* grants methods to configure the hardware, as well as to start and stop monitoring and writing execution independently. In addition, a simple syntax has been introduced to facilitate major frame and asynchronous message setup.

5.5. *SensingClass*

This *class* is in charge of the execution of all traffic analysis algorithms. It is also responsible for library generation. It is a *static class*, that is, all of its functions are *static functions* and not *methods*. In fact, its purpose is intrinsically static, since it is meant to generate an independent object, a *BusEnvironment* instance. The library generation is performed through method *analyzeLive()*, which implements multi-threading techniques to improve performance.

The latter exploits the functionality of *PeriodExtractor* to acquire an estimate of the major frame period. The following step involves the computation of probability distributions of all message types observed on the BUS. In particular, *SensingClass* acquires a set of monitored messages and, according to their *timeTag* value, estimates the probability $p(t)$of observing each of them in each instant within the aforementioned period. The estimator in this case is defined as follows:

$$\hat{p}(t) = \frac{\sum_{n=0}^{N} v_n(t)}{N}$$

where $\hat{p}(t)$is the estimate of $p(t)$,$N$is the number of observed major frames and $v_n(t)$is a function returning 1 if the message has been observed at time t,0 otherwise.

As an implementation choice we have decided not to perform the normalization over $N$and maintain the value of $\hat{p}(t)$as an integer.

For each message, the values of $\hat{p}(t)$are stored into a *Timeline* inside a *BusEnvironment* data structure. Each of these is univocally associated with the corresponding message. In addition, *BusEnvironment* provides a *busLoading* attribute which returns the probability of occupation of the BUS at each instant within the major frame period. Such attribute is computed based on the various $\hat{p}(t)$.

In addition to performing traffic analysis, the class permits to save data to a local file and load it when necessary. The *methods* that implement these features are respectively *saveTrafficDataToFile()* and *loadTrafficFromFile()*. Finally, method *loadTrafficLive()* allows to load live traffic from the BUS interface.

5.6. *ConfigurationClass*

This *class* constitutes the configuration input of a cyber attack sequence. It provides methods for setting up and sequentially popping a cyber attack chain. Internally, the attacks are stored in a queue container.

5.7. *CyberClass*

CyberClass is the core of the Cyber Device. It is demanded to load and execute cyber attack sequences. In order to do so, it requires an instance of *ConfigurationClass*, which contains the *cyber* attack information, and an instance of *BusEnvironment*, which provides the necessary traffic data. The latter is essentially used for *Injection* and *Selective Jamming* attack types. In order to avoid collisions on the BUS, *CyberClass* makes use of the *busLoading* attribute of *BusEnvironment*.

6. Experimental tests

This section describes the experimentations conducted using *hardware-in-the-loop* implementations of the MIL-STD-1553 standard. In particular, the functionality of the software has been tested against both dummy traffic and real-system traffic.

[12] We cannot use (yet) FFT algorithm for autocorrelation computation as the Fourier Transform is not defined for the finite space in question.

Fig. 8 – Test #1—Architecture setup.

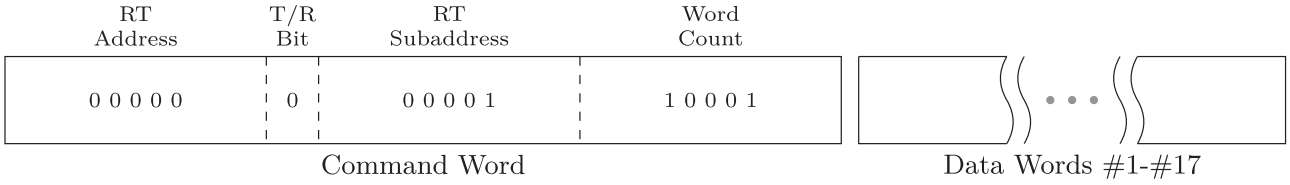

Fig. 9 – Test #1—Transmitted message.

A preliminary setup procedure has been followed in each test, whose steps are the following:

1. Cable all 1553 physical connections among the DDC boards to realize the physical BUS infrastructure;
2. Connect DDC boards to the PC's;
3. Setup the DDC board *device number* using the DDC Card Manager application;
4. Setup each terminal, according to the test configuration;
5. Launch the Cyber Device application according to the test configuration;
6. Gather data and assess the test results.

6.1. *Test #1: bus loading*

The first test was aimed at verifying the basic 1553 functionality, in particular the message writing procedure. In addition, the *Jamming* attack was tested with regards to the obtained BUS loading levels. As previously mentioned, the software lets the user specify a desired BUS loading percentage to be achieved when performing a *Jamming* attack. Therefore, it is important to verify that such desired percentage is effectively reached.

To perform this test, a simple architecture setup was used (Fig. 8).

Thus, the only traffic-generating terminal on the BUS was the Cyber Device. The BM was used in this test - as well as the other tests - for assessing the Cyber Device effects. The message that was being transmitted is the following (Fig. 9):

The results of this first experimentation are positive. The Cyber Device is able to transmit messages and to achieve the desired bus loading percentage with a MSE (Mean Square Error) of 5.31%. The following Table 1compares the obtained loading percentage to the desired one.

Table 1 – Test #1 bus loading results.

Desired bus loading percentage	Obtained bus loading percentage
20%	20.2%
40%	45.3%
60%	60.1%
90%	90.3%

6.2. *Test #2: injection on dummy traffic*

The objective of the second test was to prove the full injection procedure, from the statistical analysis to the message transmission. The test setup is shown in Fig. 10.

In this case we have several terminals connected to the BUS network. The BC and the RTs simulate message exchanges on the BUS, according to the pattern shown in Fig. 11. Such traffic scheduling produces a 25%BUS loading percentage, which is consistent with what is generally observed on real systems.

The Cyber Device's objective was to inject the message represented in Fig. 12 without causing collisions with the underlying traffic. As described in Section 5, the software employs an algorithm based on autocorrelation and classification in order to identify the time slot when the collision probability is minimum.

It is critical to consider the fact that the execution of software on a general purpose Operating System is subject to latency, and that the system's scheduling policy introduces jitter. If latency can in part be compensated by software tweaks, unpredictable jitter cannot. In addition, latency compensation introduces other jitter components. Hence, part of this test was to assess the impact of OS scheduling on the message

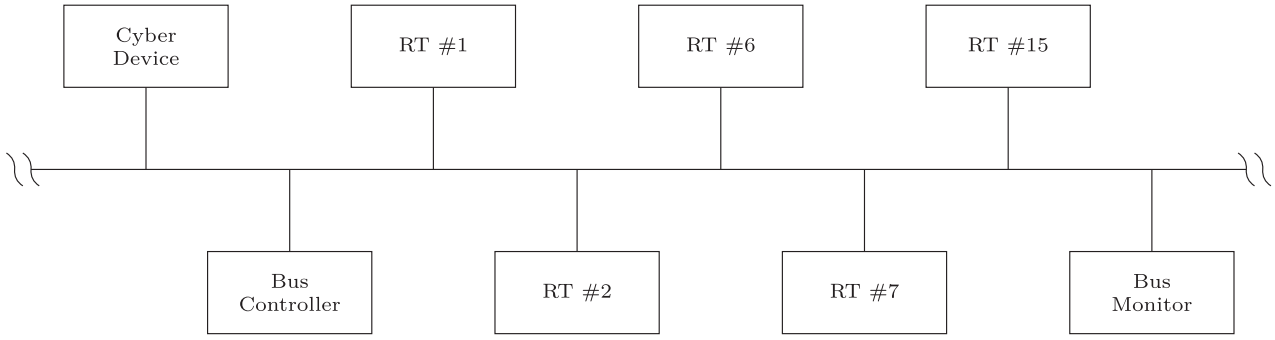

Fig. 10 – Test #2—Architecture setup.

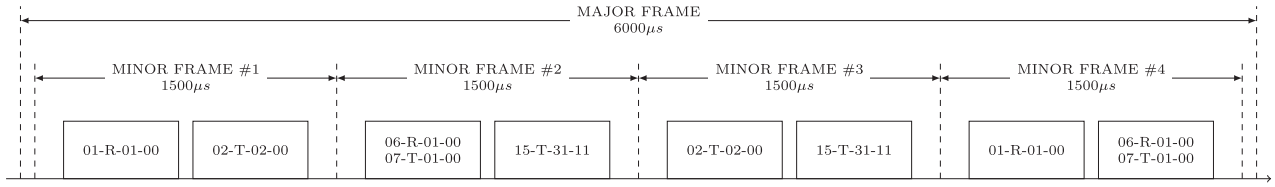

Fig. 11 – Test #2—Frame pattern.

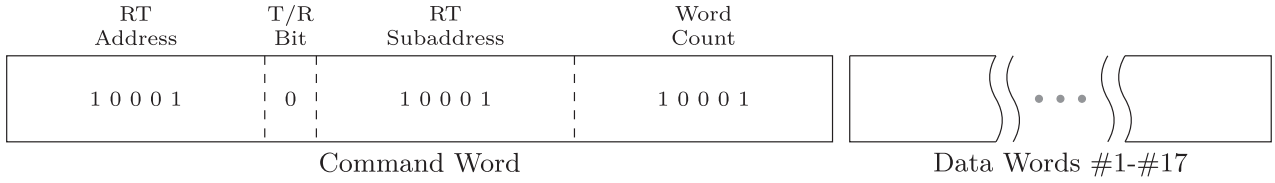

Fig. 12 – Test #2—Injected message.

Table 2 – Test #2 injection results.

Attempt no.	Cyber dev. inj.		Random inj.
	Collision	Jitter	Collision
1	NO	86 μs	YES
2	NO	286 μs	YES
3	NO	382 μs	NO
4	NO	106 μs	YES
5	NO	172 μs	YES
6	NO	126 μs	YES
7	NO	134 μs	NO
8	YES	588 μs	NO
9	NO	238 μs	YES
10	YES	564 μs	YES
Success rate %	80%		30%

transmission. To provide some figures: the word duration of 1553 is 20 μs, which means that the longest message that can be transmitted - RT-to-RT, 36 words - will be 720 μs long. Depending on the specific OS, execution timings may be of up to several milliseconds.

The test was executed having the Cyber Device run on a Windows platform - a non-real-time OS. The following Table 2 summarizes the injection results, with regards to the number of collisions occurred and the observed jitter.

Evidently, the jitter values are high compared to the standard 1553 timing scale. This outcome indicates that testing on a real-time OS, such as VxWorks or RT-Linux (a Linux kernel distribution that implements preemption and real-time features on Linux systems) would be advisable. Nevertheless, the injection has proved to be successful in 80%of cases. On the other hand, injecting a message in a random manner on the same traffic pattern proved to be successful in only 30%of cases. Indeed, the 50%improvement is to be attributed to the Cyber Device algorithm, despite the OS introducing noise.

6.3. *Test #3: injection on a real system*

This third test was analogous to the previous one, the difference being in the setup of the architecture. In this case a real system was used. Specific details of the equipment are classified and will be omitted. We will nevertheless provide a general description of the system's network topology and message framing. The target is composed of two terminals connected to a 1553 BUS network, which we will call *System BUS*. The first of them - namely the EWS (*Electronic Warfare Suite*) Manager - serves as BC, while the other - namely the *sensor* - serves as the sole RT of the network.

The EWS is demanded to perform coordination and control tasks in support of EW subsystems and sensors. In this case, the only sensor present on the network is the recipient

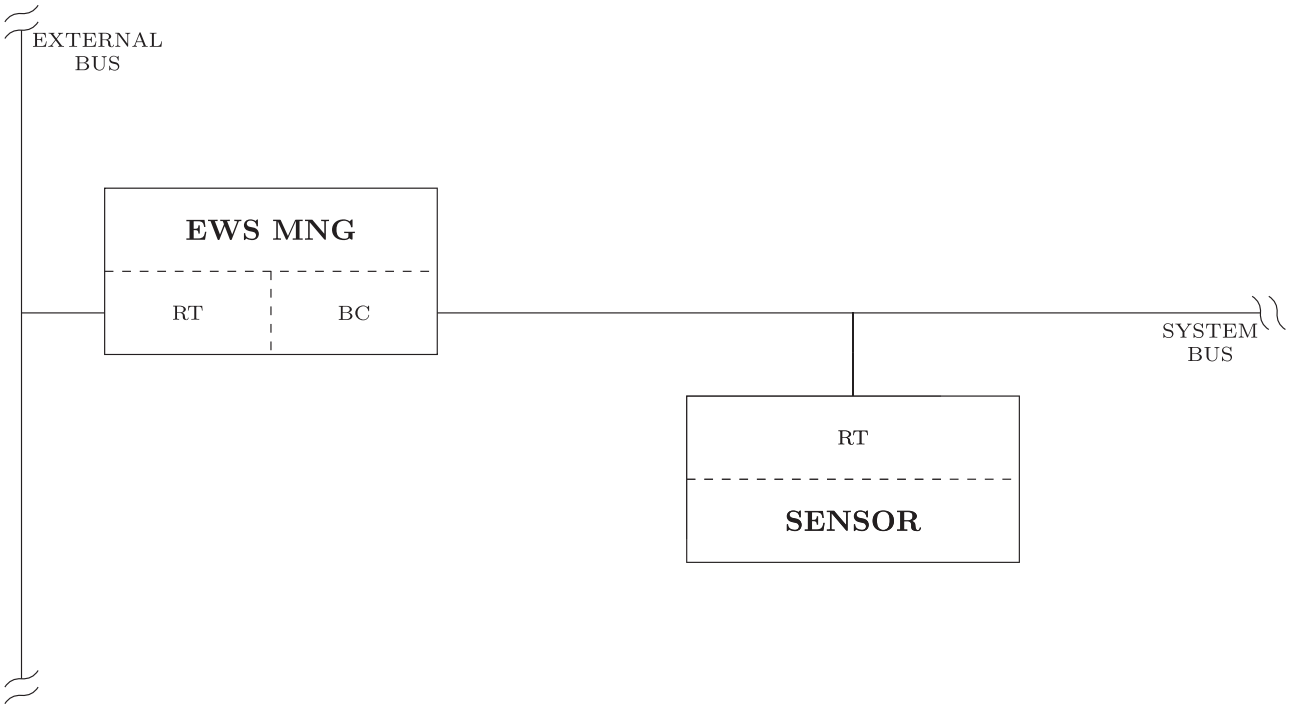

Fig. 13 – Test #3—Real system architecture.

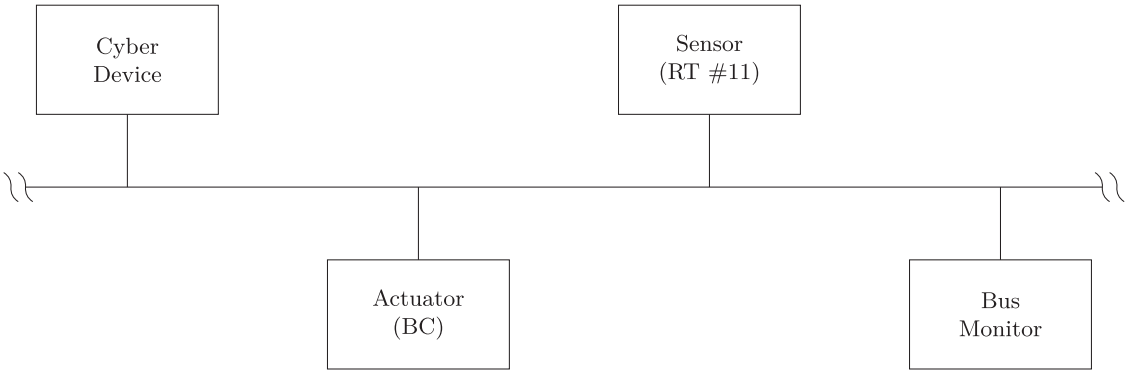

Fig. 14 – Test #3—Architecture setup.

of all the EWS Manager commands. The latter is responsible of performing EW tasks such as threat detection, identification, jamming, etc.

The system is engineered to be integrated on avionic platforms and to receive external data that are needed for operation. In particular, these data are received by the *EWS Manager* via another 1553 network, which for the purposes of this work will be called *External BUS*. Therefore, the *EWS Manager* acts as a generic RT on the *External BUS*. Fig. 13 illustrates such a concept.

For the purposes of this test, the Cyber Device has been connected to the System BUS, as shown in Fig. 14.

The injection has been conducted in the same manner as test #2. The difference lies in the nature of the underlying traffic, which in this case is not trivial but being generated by the real system. The message to be injected was the same as in the previous test (shown in Fig. 12), while the underlying traffic pattern is shown in Fig. 15. Such traffic scheduling corresponds to a measured BUS loading of 5.6%,which is considerably lower than the one adopted in the previous test. We can expect, therefore, remarkably higher success rates both for the random injection and for the Cyber Device injection. In fact, the outcomes highlighted extremely high percentages in either case, with the Cyber Device injection totalling a 100%success rate against the 80%of random execution. The test results are summarized in Table 3.

6.4. *Test #4: selective jamming on dummy traffic*

As described in Section 5, the main concern in the *Selective Jamming* case lies in the regularity of the injection times. In particular, the accuracy of injection needs to be of the order of µs. Such property can be guaranteed by *firmware*-level or *hardware*-level (i.e., FPGA (Field Programmable Gate Array) ex-

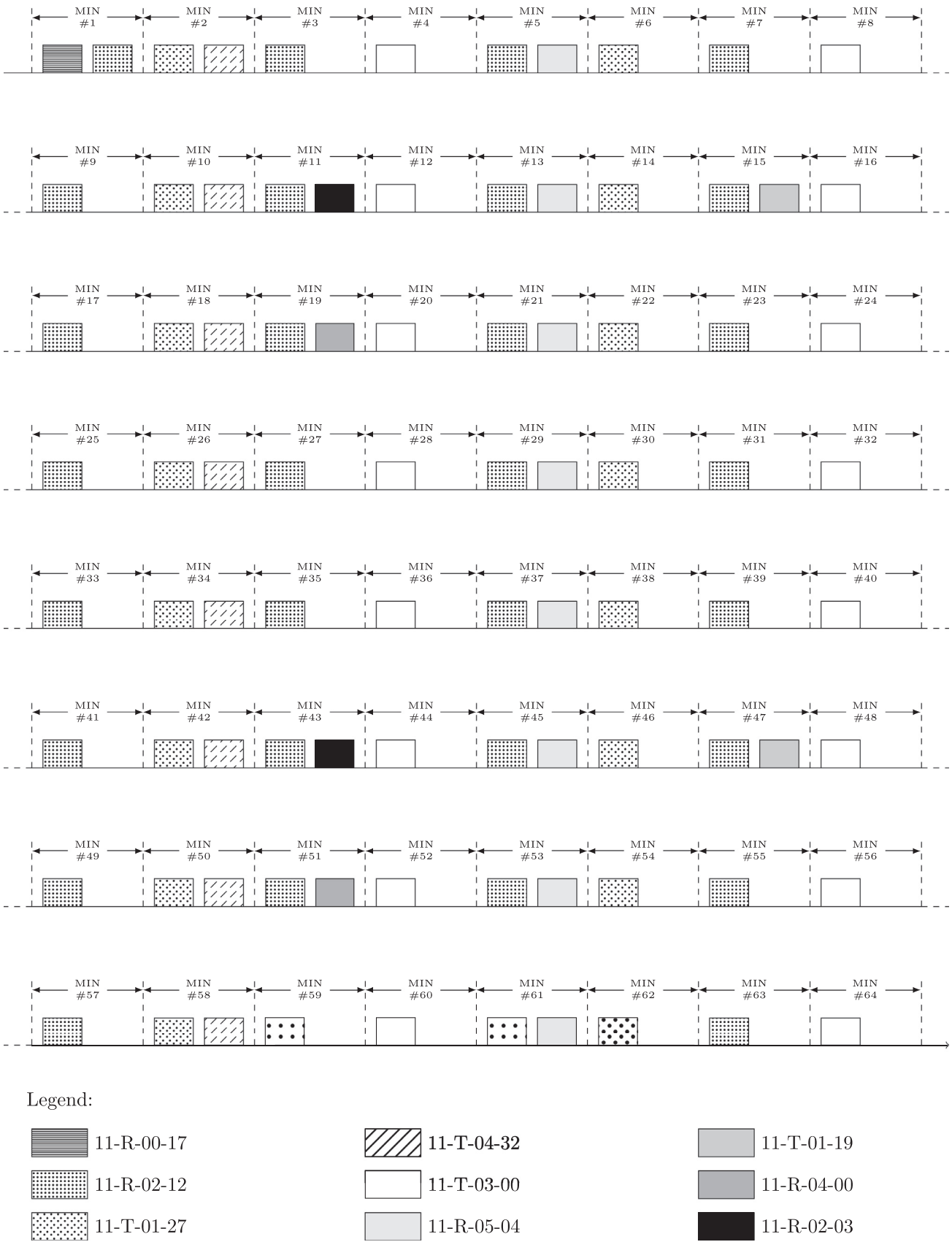

Fig. 15 – Test #3 frame pattern.

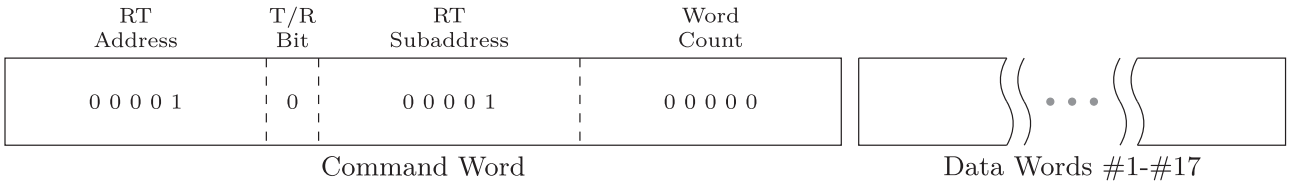

Fig. 16 – Test #4 target message.

Table 3 – Test #3 injection results.

Attempt no.	Cyber dev. inj. Collision	Random inj. Collision
1	NO	NO
2	NO	NO
3	NO	NO
4	NO	NO
5	NO	NO
6	NO	NO
7	NO	NO
8	NO	NO
9	NO	YES
10	NO	YES
Success rate %	100%	80%

Table 4 – Test #4 results.

Attempt no.	Latency	Success
1	114 μs	NO
2	230 μs	NO
3	358 μs	NO
4	91 μs	NO
5	211 μs	NO
6	208 μs	NO
7	348 μs	NO
8	496 μs	NO
9	299 μs	NO
10	107 μs	NO

ecution. Software-level execution, as previously stated, is subject to latency and OS scheduling jitter which are - generally - of the order of ms, three orders of magnitude greater than required.

This last test aimed at assessing the feasibility of *Selective Jamming* in the way it has been implemented in the Cyber Device software. The underlying traffic is the same as test #2, and is illustrated in Fig. 11. Of the four messages in such scheduling, the designated target of the attack is shown in Fig. 16.

Results highlight that the major issue lies in the synchronization of the Cyber Device frame, as it had been predicted. In fact, in every test case the attack proved unsuccessful. Nevertheless, monitoring data, as shown in Table 4, demonstrates that the cause of failure is due to software latency and jitter. In essence, the time instant at which the board firmware begins its major frame execution depends mostly on OS scheduling. We attempted to reduce latency as much as possible by anticipating the command. This however introduces additive jitter components. Overall, the mean value of obtained jitter is 246.2 μs. This is comparable to the values obtained in test #2 (see Table 2). In that case, however, the success rate depended on BUS loading levels and a latency of $\sim$ 200 μs proved to be acceptable. In this case, on the other hand, the same value is unacceptable for the timing characteristics of the 1553 protocol. A possible solution to this obstacle, which we propose for further work, would be to implement the full algorithm at a firmware level, or - better - at a hardware level through the use of FPGA technology.

7. Conclusions

The study presented in this paper has analyzed the vulnerabilities of a common avionic BUS protocol and realized a software tool capable of performing cyber attacks based on the aforementioned analysis. Such effort is grafted into the wider context of CEMA (Cyber Electro Magnetic Activities), which has become paramount in modern day conflicts and battlefields. Indeed, weapons systems are embedding more and more information technology in their core; as a consequence they are vulnerable to attacks mounted in the fifth dimension, namely the cyberspace. The need is therefore to introduce resilience in the mentioned domain. The broader objective of the paper is to provide a basis upon which measures can be taken in order to introduce high levels of resilience in avionic platforms.

The first step has been a thorough investigation of the selected protocol, namely MIL-STD-1553. The standard dates back to the mid 1970s, when security was not a major topic in system engineering. Such a protocol implements little or no procedures aimed at enforcing security policies. The main vulnerability lies in its use of time-division multiplexing, which is entirely being delegated to the BUS Controller. Through the use of command words, the BC initiates communication among the terminals and orchestrates timing and scheduling. As a consequence, if a rogue device is able to estimate idle times and send messages without causing collisions with underlying traffic, it might be able to spoof terminals and create effects. The feasibility depends on upper OSI layer implementations, terminal logic and network traffic conditions (i.e., BUS loading). In other situations, the attacker might be interested in voluntarily cause collisions on the BUS in order to prevent or impair communications. The target of the cyber action may be the infrastructure itself, or a specific system that interfaces the BUS. In the former case, jamming the physical layer is enough to achieve general collision and render impractical the legitimate exchange of messages. In the latter case, a deeper analysis led to the identification of a *Selective Jamming* attack that might be performed to impede a specific message from being transmitted. This is based on the intention of creating

collisions with a single message, which is generally inserted into a periodic framing pattern.

The second step has been to proceed with model-based system engineering methods in order to justify the system architecture on the basis of the definition of an operative scenario. The adopted modeling framework - namely ARCADIA - unfolds through three steps: Operational Analysis, System Analysis, Logical Analysis. The process allowed us to define four architectural blocks.

The subsequent phase has been to translate the modeling results into a functioning software. The implementation has been realized in C/C++, taking advantage of the AceXtreme SDK for interfacing to DDC MIL-STD-1553 boards.

The last step comprised testing and experimentation, both on dummy and real systems. The results pointed out the fact that the aforementioned 1553 vulnerabilities can indeed be exploited, and that the solution devised in our software is a favorable starting point for further work and investigation. In particular, we demonstrated that it is possible to statistically analyze traffic in order to inject messages without causing collisions. On the other hand, we have not been able to fully assess the feasibility of *Selective Jamming*. This is mainly due to the fact that the adoption of a non-real-time OS such as Windows introduces too much jitter, thus not allowing to achieve frame synchronization. In order to determine if such an attack is really effective, the software will have to be ported on a real-time OS, such as VxWorks or RT-Linux, or - better - be implemented for firmware or hardware execution.

Overall, the software requires further refinements, improvements and updates to be utilized either as a demonstrative tool or as a development tool. Nevertheless, it provides a solid basis for continuing the analysis of vulnerabilities on avionic platforms, and in particular those fielding the MIL-STD-1553 protocol. A feasible solution for implementing some cyber resilience into these systems should either take the form of a statistically-based intrusion detection system, as proposed in Stan et al. (2017) and Casillo et al. (2019), or happen through a savvy design of deep learning techniques, as in Wang et al. (2019). Indeed, some general principles presented in this work can be applied and adapted to other systems or protocols. We thus presume that our study will raise greater interest in cyber-avionics. Much work is needed, not only to fill the security gap of legacy technology, but mostly to face tomorrow's challenges which the introduction of complex and interconnected weapon systems will bring about.

Declaration of Competing Interest

The authors declare that they have no known competing financial interests or personal relationships that could have appeared to influence the work reported in this paper.

CRediT authorship contribution statement

D. De Santo: Conceptualization, Investigation, Methodology, Writing - review & editing, Software, Formal analysis. **C.S. Malavenda:** Conceptualization, Writing - review & editing. **S.P. Romano:** Conceptualization, Investigation, Methodology, Writing - review & editing, Supervision. **C. Vecchio:** Conceptualization, Writing - review & editing.

Acknowledgments

The authors would like to thank Daniela Pistoia from Elettronica S.p.A for her precious support and advice.

REFERENCES

Ali S, Al Balushi T, Nadir Z, Hussain OK. Cyber Security for Cyber Physical Systems. Springer; 2018.

. Cyber Warfare. In: Andress J, Winterfeld S, editors. Syngress; 2014.

Bozdal M, Samie M, Jennions I. A survey on can bus protocol: attacks, challenges, and potential solutions. In: 2018 International Conference on Computing, Electronics Communications Engineering (iCCECE); 2018. p. 201–5.

Casillo M, Coppola S, De Santo M, Pascale F, Santonicola E. Embedded intrusion detection system for detecting attacks over can-bus; 2019. p. 136–41.

Charles Billo, Chang W. In: Technical Report. CYBER WARFARE - An Analysis of the Means and Motivations of Selected Nation States; 2004.

Connell M, Vogler S. In: Centre for Naval Analysis Occasional Paper Series. Russia's approach to cyber warfare; 2017.

Data Device Corporation, 2003. MIL-STD-1553 Designer's Guide Sixth Edition.

Data Device Corporation, 2010. AceXtreme C Software Development Kit - Reference Manual.

El-Rewini Z, Sadatsharan K, Selvaraj DF, Plathottam SJ, Ranganathan P. Cybersecurity challenges in vehicular communications. Veh. Commun. 2020;23:100214. doi:10.1016/j.vehcom.2019.100214.

Gunduz MZ, Das R. Cyber-security on smart grid: threats and potential solutions. Comput. Netw. 2020;169:107094. doi:10.1016/j.comnet.2019.107094.

Hicks B. Transforming avionics architectures to support network centric warfare, 2; 2004. p. 8.E.3–81.

Koç CK. Cyber-Physical Systems Security. Springer; 2018.

Lehto M, Neittaanmäki P. Cyber Security: Analytics, Technology and Automation. Springer; 2015.

Lind WS, Nightengale K, Schmitt JF, Sutton JW, Wilson GI. The changing face of war: into the fourth generation. Marine Corps Gazette 1989;October:22–6.

Lydiate D. In: Technical Report. Air Power's Cyber Challenge: Are Cyber-Vulnerabilities a Credible Threat to a Modern Air Force?. University of Exeter; 2018.

Range Commanders Council, 2007. Chapter 10 - Digital Recording Standard.

Stan, O., Elovici, Y., Shabtai, A., Shugol, G., Tikochinski, R., Kur, S., 2017. Protecting Military Avionics Platforms from Attacks on MIL-STD-1553 Communication Bus, 1–15.

. Cyber Security and IT Infrastructure Protection. In: Vacca JR, editor. Syngress; 2014.

Voirin, J.-L., 2017. Model-based System and Architecture Engineering with the Arcadia Method.

Wang H, Ruan J, Ma Z, Zhou B, Fu X, Cao G. Deep learning aided interval state prediction for improving cyber security in energy internet. Energy 2019;174:1292–304. doi:10.1016/j.energy.2019.03.009.

Wu W, Li R, Xie G, An J, Bai Y, Zhou J, Li K. A survey of intrusion detection for in-vehicle networks. IEEE Trans. Intell. Transp. Syst. 2020;21(3):919–33.

Simon Pietro Romano is an associate professor in the "Department of Electrical Engineering and Information Technology" at the University of Napoli Federico II. He teaches computer networks, computer architectures, network security, and telematics applications. He is also the co-founder of Meetecho, a startup and university spin-off dealing with scalable video streaming and WebRTC-based unified collaboration, as well as of SECSI (SECurity Solutions for Innovation), that focuses on network security. He actively participates in IETF (Internet Engineering Task Force) standardization activities, mainly in the Applications and Real Time (ART) area. He conducts research activities in the field of networking since 1998. He has been working on the definition of advanced networking architectures capable to provide end-users with optimized quality of experience, thanks to a closed-loop approach going from SLA-based management down to policy-based configuration of the devices and autonomic monitoring. He has contributed to the field of real-time multimedia applications, with special regard to the design, implementation and standardization of scalable, distributed architectures for conferencing, media control and telepresence. He has carried out research in the field of network security, by mainly focusing on distributed intrusion detection and critical infrastructure protection. Lately, he has started investigating issues associated with distribution of control in 5G-compliant, container-based, virtualized architectures. He has a long track of participations in R&D projects, both at the national and at the international level. He is currently leading the SHINE (Secure Hybrid In Network caching Environment) project funded by the European Space Agency (ESA).