Computers & Security 150 (2025) 104205

Contents lists available at ScienceDirect

Computers & Security

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

A cyber-resilient open architecture for drone control

Nicola d'Ambrosio, Gaetano Perrone, Simon Pietro Romano *, Alberto Urraro

University of Napoli Federico II, Department of Electrical Engineering and Information Technology, Via Claudio 21, 80125 Napoli, Italy

ARTICLE INFO

Dataset link: https://github.com/NS-unina/GCAP_Mod

Keywords:
Resilience
Security
Safety
STAMP/STPA-sec
Open system architecture
Microservices
Unmanned aerial vehicles

ABSTRACT

Unmanned Aerial Vehicles (UAVs) are becoming important tools in both military and civilian sectors. However, the prevalent use of monolithic architectures in contemporary platforms limits the swift integration of new features and significantly hampers the adaptability of UAVs to an ever-changing operational environment. Furthermore, this constantly evolving landscape highlights the inherent complexity of assessing drone safety and security since this process requires managing multiple and rapidly changing variables. Therefore, it is imperative to adopt an open system approach that relies on microservices and virtualization in order to overcome the limits of traditional drone architectures. This study presents a new method that involves breaking down the UAV monolithic system into a network of separate and virtualized components, each holding a single responsibility and designed according to the Open System Architecture (OSA) principle. Moreover, this work proposes a novel cyber-resilience model to determine cyber threats and assess their impact on the system. This approach leverages NIST 800-53, MITRE ATT&CK, STPA-Sec, and Attack Graph in order to identify the sequence of malicious actions that can lead to a specific hazardous scenario. Lastly, we demonstrate the effectiveness of this novel architectural paradigm by developing a software-in-the-loop simulation testbed for fast prototyping new features and validating the results of the cyber-resilience model.

1. Introduction

The extensive use of drones has led to significant transformations in several sectors, such as surveillance, research, rescue operations, and the military sector. Unmanned Aerial Vehicles (UAVs) have shown their adaptability as tools capable of operating in regions that would otherwise be unreachable or dangerous for human involvement. As a result, researchers are currently exploring methods to enhance the performance of these airborne devices in many specific scenarios (Shakhatreh et al., 2019). These improvements primarily focus on enhancing critical components of UAVs, such as autopilot systems, sensor data gathering, and network protocols. These upgrades are necessary for effectively operating drones and ensuring the successful fulfillment of their assigned tasks. However, integrating these innovations into practical and reliable solutions can become cumbersome. One of the primary concerns surrounding UAV firmware is its monolithic architecture, which creates significant challenges related to scalability, extendibility, and testing.

With the advancement of UAV systems' complexity and utilization, cybersecurity has become an essential requirement. Monolithic architecture limitations have become critical, as (i) it is not possible to model vulnerabilities and attacks without comprehensive knowledge of system internals, (ii) it is not possible to integrate security controls to address identified risks, and (iii) it is not possible to practically verify the modeled vulnerabilities through attacks in the system.

To tackle these challenges, we propose an architectural development methodology based on microservices, open standards, and virtualization, breaking down the monolithic system into a network of independent and autonomous services, each responsible for a specific capability. This technique is in accordance with the concepts specified in the Open System Architecture Framework (OSA) (Sage, 1988), which establishes a reference model and a set of standards for constructing interoperable systems employing microservices. By adhering to these principles, we can ensure that our architecture allows for the integration of security controls, remains adaptable to emerging technologies and standards, and enables testing capabilities for assessing the cybersecurity of the developed UAV system. We evaluate these capabilities by implementing several attack scenarios through both Software-in-the-Loop (SITL) and Hardware-in-the-Loop (HITL) simulations.

Even though open systems are increasingly being used across various domains, there is a significant lack of studies on security assessment approaches for these systems. Notably, state-of-the-art methodologies like STAMP (Leveson, 2004) and STPA (Ishimatsu et al., 2010; Young

* Corresponding author.
E-mail addresses: nicola.dambrosio2@unina.it (N. d'Ambrosio), gaetano.perrone@unina.it (G. Perrone), spromano@unina.it (S.P. Romano), al.urraro@studenti.unina.it (A. Urraro).

https://doi.org/10.1016/j.cose.2024.104205
Received 22 July 2024; Received in revised form 23 October 2024; Accepted 7 November 2024
Available online 23 November 2024

and Leveson, 2013), integrated with STRIDE (De Souza et al., 2020), have been proposed as suitable frameworks for addressing these issues. However, these frameworks often fall short in modeling vulnerabilities and evaluating their specific impacts. To overcome these challenges, we propose integrating NIST 800-53 (NIST, 2024) and the MITRE ATT&CK framework (MITRE ATT&CK, 2024) alongside the aforementioned methodologies. To summarize, this work brings the following contributions:

- We propose an approach for transitioning from a monolithic system to a microservices-based architecture for UAV firmware in order to align with principles expressed by the *Open System Architecture Framework* (OSA).
- We introduce a novel cyber-resilience model based on STPA, STRIDE, MITRE ATT&CK, and NIST 800-53 standards. This model identifies hazardous scenarios and uncovers the sequences of malicious actions that can trigger them.
- We implement an experimental Software-in-the-Loop (SITL) simulation environment using Docker, along with a Hardware-in-the-Loop (HITL) simulation that integrates a commercial drone into the system. This demonstrates the benefits of OSA principles and validates the outcomes obtained from applying our cyber-resilience model.

All the code necessary to replicate the experiments mentioned above is available in the project's GitHub repository (NS-UniNa, 2024).

The remaining sections of the paper are structured as follows. Section 2 analyzes relevant literature on the application of OSA principles to UAV design, the implementation of safety and security models, and the identification of common UAV attacks. Section 3 presents a methodology for deconstructing the UAV monolithic system into separate virtualized components, assessing both safety and security. Section 4 provides insights into the development process used to build our prototype drone system. Section 5 describes the application of the discussed modeling methodologies within our domain of interest. Section 6 details the process of extracting attack paths that lead to the hazardous scenarios identified during the safety analysis. Section 7 applies our enhanced safety-security model to extract attack paths from our experimental setup, identifying three specific attack paths. Section 8 emulates these attacks to validate their effectiveness and impact. Section 9 analyzes the benefits and drawbacks associated with our approach. Section 10 presents the conclusions.

2. Related works and background

Current UAV monolithic architectures have significant limitations in integrating new features, adapting to evolving technologies, and managing risk. Their rigid structure hampers flexibility, scalability, and responsiveness, making it difficult to introduce new functionality or adopt modern technologies like microservices and virtualization. Consequently, monolithic systems face obsolescence and slow down innovation, necessitating a shift toward more adaptable architectural frameworks.

Safety and security models are essential for identifying potential hazards and maintaining system reliability. These models can be categorized into linear, epidemiological, and systems accident models (Zhang et al., 2022). Systems Theoretic Accident Model and Process (STAMP) (Leveson, 2004) does represent a key model, offering broader safety recommendations compared to alternatives like AcciMap (Rasmussen, 1997). STAMP's practical implementation, Systems Theoretic Process Analysis (STPA) (Ishimatsu et al., 2010), assesses risks by examining control structures and identifying unsafe actions. The extended STPA-Sec (Young and Leveson, 2013, 2014) addresses security but lacks protocols for intentional attack scenarios and overlooks factors outside the control model (Schmittner et al., 2016).

Enhanced approaches like STPA-SafeSec (Friedberg et al., 2017) add generic component diagrams to include cyber threats in causal factors, yet still do not account for the sequence of malicious actions. Frameworks such as STRIDE (Khalil et al., 2023; Haider et al., 2019; Ahn et al., 2021) help identify threats but fail to integrate safety risks (Sahay et al., 2023), making their combination with safety models (De Souza et al., 2020) necessary for linking hazards and cyber attacks.

Attack Graphs (AG) (Zenitani, 2023) map out detailed attack paths, while integrated approaches like Cassandra (Castiglione and Lupu, 2023) merge STPA-Sec, STRIDE, and formal validation to analyze potential attack sequences. Despite versatility, these methods lack formal descriptions of malicious actions. Hybrid testbeds (Perrone et al., 2024) that integrate AG and STRIDE aid in simulating real-world scenarios but do not formally map malicious actions to a standard dictionary.

The analysis of the state of the art allows us to identify the key challenges for monolithic UAV architectures, such as inflexibility, limited scalability, and risk of obsolescence. It also unveils the current limitations of cyber-resilience models, underscoring the need for a comprehensive framework to address these issues effectively.

In the depicted scenario, two main focus areas can be identified. The first area concerns the *Open System Architecture* (OSA) framework and its applications in the world of UAVs. The second area examines potential weaknesses and vulnerabilities in UAV systems.

2.1. Open system architecture

The current development cycle for drone firmware and simulators often employs a monolithic architecture and tightly coupled components. Various components such as the physics engine, sensor models, terrain generator, visualization, and autopilot interface are all bundled together within a single integrated application. Although this technique may seem easy to put into action at first, it carries significant drawbacks. Integrating novel features requires substantial modifications to the source code and meticulous testing since changes in one element may impact other system components (Boniol and Wiels, 2014). Consequently, the maintenance of the platform code can be cumbersome.

Researchers are actively exploring the use of microservices-based architectures to address these difficulties (Irizarry et al., 2022; Ates et al., 2009; Matlekovic et al., 2022). The decoupling technique recommended by the microservices development paradigm allows for integrating several components developed by different teams. This leads to decreased development time and cost, improved solution testability, and expedited safety and security validation (Immanuel and Johnson, 2002).

Within this context, the *Open System Architecture* (OSA) framework offers a great opportunity to move beyond the constraints of monolithic systems toward more flexible and interoperable platforms. OSA specifies the interfaces and separation between applications and components that manage those interfaces. Additionally, OSA provides details on the construction of systems. The main promoter of this approach is undoubtedly the *United States Department of Defense* (DoD), which has fostered several initiatives to propose OSA standards for use in military avionics systems. Notable among these initiatives are the *Unmanned Systems* (UxS) *Control Segment* (UCS) *Architecture*, the *Open Mission Systems* (OMS) initiative, and the *Open Group Future Airborne Capability Environment* (FACE) Approach (Tokar, 2017). The UCS introduces an open standard message block for machine-to-machine communication with the goal of promoting easy integration of new services, as well as components reuse across different programs. The OMS architecture employs *Service-Oriented Architecture* (SOA) concepts (The Open Group, 2024) to design the *Avionics Service Bus* (ASB) protocols, which enable the integration of various components into a cohesive military system. The FACE open architecture is defined by a collection of standards to address business and conformance concerns in parallel with technical guidelines. These standards provide a strong foundation for creating compatible and reusable components for avionics systems. Similar goals are also pursued by initiatives like the ROS-Military project (Towler

and Bries, 2018), which promotes the adoption of open architectures and modular design principles for military robotics applications.

In essence, these methodologies propose a novel approach to decomposing avionics systems into smaller, cohesive, and loosely coupled architectural components. The interfaces that enable communication between these components are open and adhere to standardized protocols.

2.1.1. OSA-based UAV architectures

Pastor et al. (2009) propose a highly flexible and reusable hardware/software architecture aimed at facilitating the development of UAV-based remote sensing applications. Central to their design is a distributed embedded architecture that leverages a service-oriented approach to effectively manage and integrate the various sensors and components of a UAV system. To standardize the interaction between these components, they introduced the UAV Service Abstraction Layer (USAL), a service-oriented framework that supports the development and execution of remote sensing applications. USAL provides a suite of pre-defined services that can be easily customized to meet the specific requirements of different applications. Furthermore, the architecture allows for the seamless introduction of additional services to extend functionality while also enabling the reuse of existing services. The system's middleware is specifically designed to facilitate the inter-service communication necessary for USAL, ensuring efficient coordination and data exchange between system components.

Aerostack (Sanchez-Lopez et al., 2017) is an open-source software framework developed for the construction and management of aerial robotic systems. This open-source architecture provides flexibility through the use of container virtualization and supports both hardware-in-the-loop (HITL) and software-in-the-loop (SITL) simulations, suitable for single or multi-robot solutions. In terms of communication, Aerostack leverages the Robot Operating System (ROS) middleware, which enables standardized communication protocols. This setup facilitates the deployment of software components across onboard computers or ground stations connected via either a local area network (LAN) or a wireless local area network (WLAN). Communication between these components is facilitated through predefined ROS topics and messages tailored for aerial robotics. It is important to highlight that considerations of the security pertaining to this platform were not factored into their findings.

Both works leverage container-based virtualization to implement an open-system architecture. However, they primarily focus on describing the architecture itself. In contrast, the key contribution of our work lies in demonstrating the SITL and HITL simulation capabilities of the OSA architecture, with a particular emphasis on cybersecurity through the introduction of a cyber-resilience model that can be applied within the presented OSA system.

2.2. UAV attacks

Like any other system, *Unmanned Aerial Vehicles* (UAVs) face a range of potential threats that can compromise their safety, security, and privacy. These threats may occur in the physical domain, the cyber domain, or the cyber–physical domain Yu et al. (2023). Physical threats encompass the act of intentionally damaging or manipulating hardware components. Cyber threats involve attacks that compromise the integrity, confidentiality, and availability of communication networks. Cyber–physical threats refer to attacks that directly impact the drone's interaction with the physical environment.

Physical attacks. These attacks aim to completely destroy the target drone or one of its core components. Typically, these attacks involve intentional physical damage to the UAV using firearms or striking the drone with other objects. Another form of physical attack is interception, where the drone is captured mid-flight using nets or lasers (Arthur, 2018). Additionally, this category includes attacks that target remote controls, such as Android or iPhone devices, by infecting them with malware to compromise the UAV operation (Mansfield et al., 2013).

Cyber–physical attacks. The objective of these attacks is to alter the drone's perception of its physical environment by compromising the functionality of its sensors through the use of spoofing and jamming techniques. Signal jamming disrupts communication signals between the drone and its remote controller or GPS satellites using jamming devices. Signal spoofing, on the other hand, involves manipulating broadcast communication signals to provide false data to the intended recipients. The most significant example of this is GPS spoofing, where false GPS signals deceive the UAV by providing incorrect location data (Kerns et al., 2014).

Cyber attacks. These attacks aim to disrupt network communication between components by employing several strategies, including *Man in the Middle-based attacks* (MitM), *False Data Injection/False Command Injection* (FDI/FCI), Replay attacks, and Tampering attacks. A MitM attack involves intercepting and manipulating communications between two entities that are believed to be interacting directly with each other (Yaacoub et al., 2020). Several approaches are taken to perpetrate MitM attacks, and one of the most common methods is the use of ARP (Address Resolution Protocol) poisoning. FDI/FCI attacks introduce false data or commands into systems by exploiting the absence of authentication features in communication protocols such as MAVLink (Xu et al., 2021). Replay attacks capture legitimate transmissions and replay them to force the drone to exhibit unintended behavior (Sánchez et al., 2019). Tampering attacks maliciously alter the integrity of transmitted or stored information, potentially compromising crucial systems for aviation safety, such as the Traffic Collision Avoidance System (TCAS) (Salgado and de Sousa, 2021).

3. Methodology

This section aims to present the methodology applied in this work, with Fig. 1 providing an overview of the process. This methodology comprises three stages, each of which has distinct objectives: (i) developing a new architectural approach for constructing UAVs leveraging OSA principles; (ii) creating a cyber-resilience model for identifying hazardous scenarios and the sequence of malicious actions that can trigger them; (iii) validating the designed model by simulating attacks against a realistic, ad hoc constructed, experimental setup. Notably, each stage takes specific inputs and produces outputs that are utilized in the subsequent stage.

3.1. Legacy to OSA

In the first stage, which is divided into three subphases, we perform a thorough analysis of the existing UAV architecture to identify methods for decoupling its components and transitioning from a monolithic, tightly coupled system to a microservices-based architecture. Furthermore, we assess the applicability of these concepts within the contexts of Software-in-the-Loop (SITL) and Hardware-in-the-Loop (HITL) simulations. In the second subphase, we choose the components for this architecture based on the analysis of results from the previous subphases. The final subphase involves integrating the selected components and establishing the experimental setup.

3.2. Cyber-resilience model construction

In the second stage, we first apply the four steps of the STAMP/STPA-Sec procedure to the designed architecture, then identify and validate the sequence of malicious actions that can lead to specific accidents. Starting from a high-level description of the domain of interest, we first analyze the UAV system to identify its components and interconnections, as the STAMP methodology outlines. Sequentially, we: (i) utilize the STPA-Sec process to identify accidents, hazards, and losses; (ii) define the safety control structure; (iii) identify hazardous control actions (Section 5). Then, we identify and validate the sequence

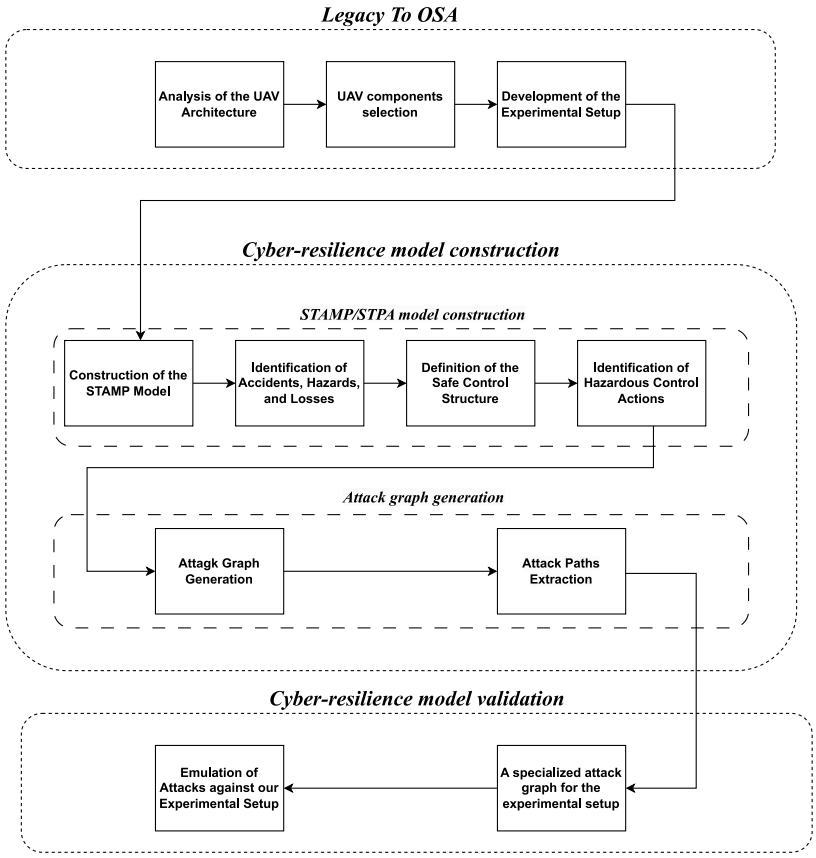


Fig. 1. Methodology overview.

of malicious actions that can lead to specific accidents among those determined in the previous stage (Section 6).

In detail, we employ NIST 800-53 to model weaknesses and use a combination of STRIDE and the hazardous control actions identified in the previous stage to determine attack goals (Section 6.1). Secondly, we generate an Attack Graph using MulVal (Ou et al., 2005) and extract the desired attack paths using Dijkstra's algorithm (Section 6.2). The malicious actions, which specify all the steps to reach the attacker's goals, are modeled using the MITRE ATT&CK framework. This characteristic offers a notable improvement compared to other works that rely solely on the STRIDE threat model.

3.3. Cyber-resilience model validation

In the final stage, we evaluate the constructed cyber-resilience model by developing a specialized attack graph tailored to the experimental setup defined in the first phase. This attack graph is used to systematically analyze potential attack vectors and vulnerabilities. Subsequently, we emulate the malicious actions outlined in the identified attack paths against the setup to test the effectiveness and robustness of the cyber-resilience model. This evaluation aims to validate the model's ability to withstand and mitigate the specified attacks, while also assessing its practical implications for enhancing the overall security posture of the UAV system under study (Section 8).

4. Legacy to OSA

This section provides insights into the development and validation process employed for building our drone system. The system design follows the principles of the *Open System Architecture* (OSA). Implementation relies on both *Software-in-the-Loop* (SITL) and *Hardware-in-the-Loop* (HITL) simulations.

In our evaluation of flight controllers, we analyzed in-depth the PX4 Flight Controller (px4, 2024) and the INAV Flight Controller (iNavFlight, 2024), to ascertain the most suitable option for our specific research context. PX4 is widely recognized in UAV systems for its comprehensive list of functions and initially appeared to offer substantial advantages for our work. Nevertheless, we had difficulties regarding its compatibility with a virtualized and containerized environment. Indeed, the complexity of simulated hardware dependencies and high interdependence with the underlying operating system made it challenging to establish adequate isolation within the containers. This lack of isolation posed a significant obstacle in creating a highly modular and configurable simulation environment. In order to overcome these issues, we moved our focus to the INAV Flight Controller. This controller is renowned for its versatility and its capability to adapt to a diverse range of drones and mini-drones.

Regarding communication protocols, we utilized the *Multiwii Serial Protocol* (MSP) and *Transmission Control Protocol* (TCP) in our simulation environment. MSP, a lightweight serial protocol, serves as the primary communication method between UAV components and the flight controller. TCP, on the other hand, is used to transport MSP data over IP networks, facilitating seamless integration within a virtualized and containerized setup. This approach enables the simulator to run inside a container while simultaneously communicating with multiple sensors and devices. Consequently, the combined use of MSP and TCP contributes to an efficient and interoperable simulation setup.

4.1. Experimental setup

The Flight Controller executes the INAV autopilot software and manages flight control logic. The GPS Module generates simulated GPS data and transmits it to the flight controller. Lastly, the Radio Controller Module generates simulated radio controller data and transmits it to

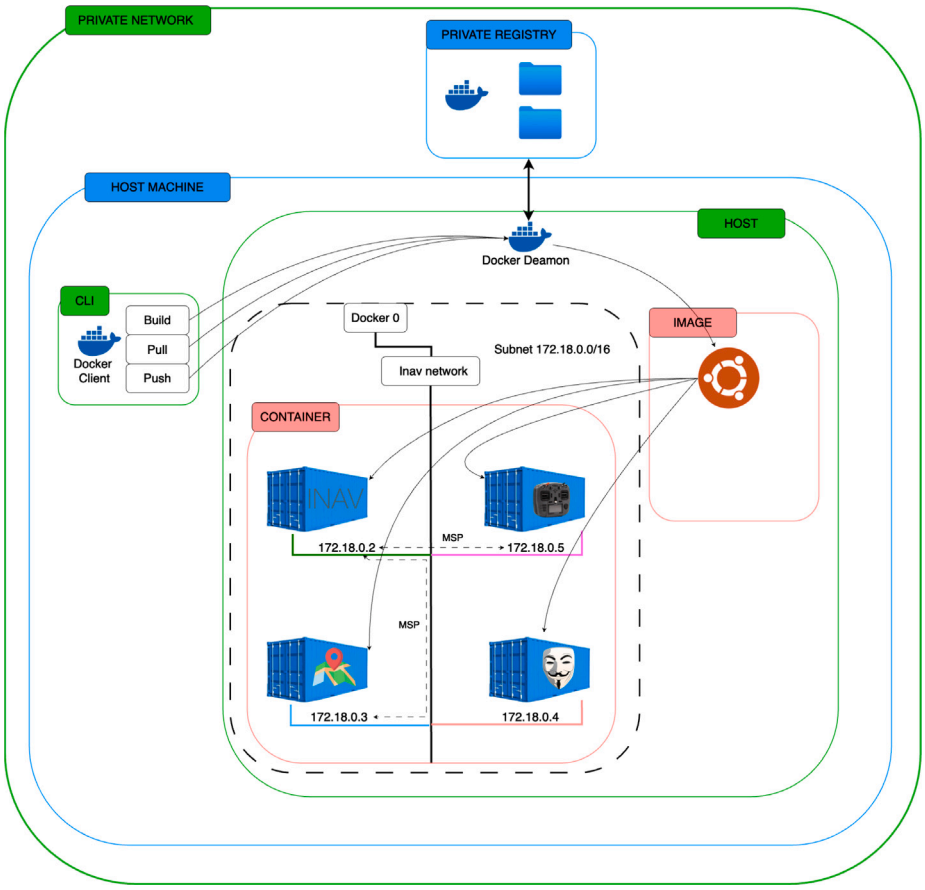


Fig. 2. Overview of the experimental setup.

the flight controller. In addition to the aforementioned components, the framework encompasses two auxiliary components: the Attacker Node and the INAV Configurator. These nodes are not accountable for ensuring the reliable simulation of the drone's flight or managing its control logic. Instead, their purpose is to enhance testing and security evaluation activities. The Attacker Node is instrumental in emulating attack activities, as it will be discussed in Section 8. The INAV Configurator provides a graphical user interface on the host computer for configuring the Flight Controller (see Fig. 2).

We employ the Docker engine on the host machine to ensure isolation between the various components of our modular *Software-in-the-Loop* (SITL) environment. Docker is particularly useful in this architecture because it not only manages the lifecycle of each container but also provides the network infrastructure interconnecting them.

Another key feature of this architecture is the use of a container image registry. This registry acts as a central repository for docker images, simplifying the process of distributing and updating containerized applications. Indeed, developers can create their components, independently assess safety and security requirements, and share them with the community through these databases. Therefore, this approach allows others to integrate these components by simply verifying adherence to specified interfaces and performing the necessary integration tests. This streamlined process ensures that components can be easily shared and tested. It is worth noting that this design architecture also enables the integration of real sensors by employing a "serial to IP" converter. This feature allows sensor data to be seamlessly transmitted over the network, enabling real-time interaction between physical sensors and the virtualized components within the Software-in-the-Loop (SITL) environment.

5. STAMP and STPA models construction

The STAMP model is used to assess the safety of drones by analyzing different system components and their interconnections in order to discover all control loops. The control chains in the STAMP model are established by a series of interactions, where each link is defined by the responsibilities assigned to the components in the system. The components include the human pilot, autopilot, sensors, and communications. Through the analysis of the interdependencies among these elements, it is feasible to detect potential risks and determine the necessary protective measures. The diagram generated from this research is shown in Fig. 3. As we will better explain in the following, the figure reports *Control Actions* (CA) with red lines and *Feedback* (F) with blue lines, over either digital (*solid*) or physical (*dashed*) channels.

To enhance clarity and comprehension, in the upcoming stages of our analysis we will delve deeper into the navigation component and a subset of sensors. This subsystem, depicted in Fig. 3 and highlighted by the dashed black shape, was selected for focus due to its critical role in reducing the risk of collisions with obstacles, other drones, or structures. This aspect is essential for preventing accidents and damage, thereby ensuring safe and hazard-free flights.

The STPA model construction is based on the STAMP/STPA-Sec methodology, which aims to individuate the cause and consequences of a specific hazardous scenario through a detailed analysis of the identified control chains. This process consists of three steps, each of which will be explained in the following paragraphs: (i) identification of accidents, hazards, and losses; (ii) definition of the control structure; (iii) identification of hazardous control actions.

Identification of accidents, hazards, and lossses. A *hazard* refers to a system state that can lead to an accident in conjunction with a worst-case set of environmental conditions. An *accident* is an undesired event

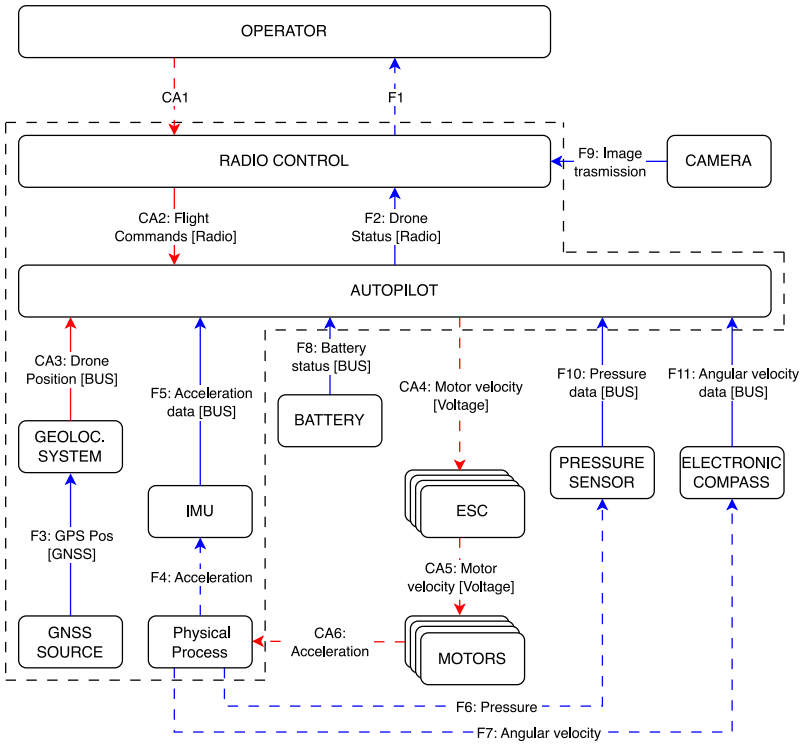


Fig. 3. STAMP model of the UAV system. (For interpretation of the references to color in this figure legend, the reader is referred to the web version of this article.)

Table 1
Mapping of identified Accidents to corresponding Losses in the drone system.

Accident	Description	Loss
A_1	Communication problems between radio control and autopilot system	L_2
A_2	Autopilot misunderstands command received	L_1 , L_2
A_3	Error in position calculation by the GPS	L_2
A_4	Loss of drone control	L_1, L_2
A_5	Spoofing of telemetry feedback signal	L_3
A_6	Collision with obstacles or terrain	L_1, L_2

Table 2
Identified Hazards and their associations with Accidents in the drone system.

Hazard	Description	Accident
H_1	Uncontrolled Emergency Landing	A_4
H_2	Unintentional entry into unauthorized areas	A_1
H_3	Loss of GPS signal	A_3
H_4	Violation of the operational altitude ceiling	A_2, A_6
H_5	Loss of planned route due to imprecise GPS data	A_4, A_6
H_6	Loss of radio control signal	A_2, A_4
H_7	Loss of telemetry feedback signal	A_5

that results in a *loss* (e.g., human life or injury, property damage, financial loss) (Williams, 2024). Therefore, this step involves a comprehensive system analysis to ensure all potential risks are considered and documented.

In accordance with Sayers et al. (2020), our analysis considers three scenarios of losses: Loss of lives (L_1), Loss of drone (L_2), and Loss of sensitive information (L_3). Moreover, we consider the following subset of Accidents: Communication problems between radio control and autopilot (A_1), Autopilot misunderstands command received (A_2), Error in position calculation by the GPS (A_3), Loss of drone control (A_4), Spoofing of telemetry feedback signal (A_5), and Collision with obstacles or terrain (A_6). The results of this analysis and the interrelationships between accidents and losses are systematically described and presented in Table 1.

Associated with these accidents, we identified the following list of hazard conditions: Uncontrolled Emergency Landing (H_1), Unintentional entry into unauthorized areas (H_2), Loss of GPS signal (H_3), Violation of the operational altitude ceiling (H_4), Loss of planned route due to imprecise GPS data (H_5), Loss of radio control signal (H_6), and Loss of telemetry feedback signal (H_7). The hazards and their association with the corresponding accidents are detailed in Table 2.

Definition of the control structure. The second step of this procedure regards the definition of the *Safe Control Structure* (SCS). In this specific case, the SCS can be described by combining components such as controllers, sensors, actuators, and physical processes, along with their respective control and feedback actions. To be concise, we conduct an in-depth analysis of only the subsystem highlighted within the dashed black shape in Fig. 3. The results of this detailed analysis are presented in Table 3.

Table 3
Overview of Control Actions (CAx) and Feedback (Fx).

Control action/ feedback	Description
CA_1	Send mission start/stop commands, send mission coordinates
CA_2	Send flight commands to the autopilot
CA_3	Calculate drone position
F_1	Visual and instrumental feedback from radio control to operator
F_2	Receive telemetry/sensor status (from Autopilot to radio control)
F_3	Receive coordinates information (from satellites to GPS)
F_4	Measure Acceleration
F_5	Receive data (from IMU – Inertial Measurement Unit – to Autopilot)

Identification of Hazardous control actions. The third and final step of this methodology involves associating hazards with their respective control and feedback actions, referred to as *Hazardous Control Actions* (HCAs).

By establishing this association, we can recursively identify which parts of the control system are affected by specific hazards and, consequently, determine the potential accidents they can lead to. In our architecture, we identify the following five Hazardous Control Actions (HCAs):

- HCA_1: CA_2 is *applied* with incorrect parameters sent from the *Radio Controller* $\rightarrow$ (H_1, H_2, H_4);
- HCA_2: F_2 is *applied* with wrong parameters provided by the *Autopilot* $\rightarrow$ (H_1, H_2, H_4, H_6);
- HCA_3: CA_3 is *applied* with wrong position calculation provided by the *Localization System* $\rightarrow$ (H_3, H_5);
- HCA_4: F_3 is *applied* when incorrect data is received from the *GNSS Module* $\rightarrow$ (H_3, H_5);
- HCA_5: F_5 is *applied* with incorrect acceleration data sent from the *IMU* $\rightarrow$ (H_1, H_4);

6. From hazard scenarios to attack paths extraction

After the realization of the STAMP and STPA models, we aim to integrate their insights with the *Attack Paths* generation process. Specifically, this step enables us to identify the sequence of malicious actions that lead to a specific hazard scenario. The methodology is structured into three main steps: (i) Architecture Description, (ii) Attack Graph Generation, and (iii) Attack Paths Extraction. In the first phase, we describe the system architecture and its corresponding vulnerabilities using frameworks such as STRIDE, MITRE ATT&CK, and NIST 800-53. The second stage involves defining and generating the Attack Graph, which visualizes the attack paths and system vulnerabilities. In the third and final stage, we analyze the generated Attack Graph to discover the attack paths that lead a malicious actor to a specific hazard. We employ MulVal, a logic-based network security analyzer, to implement this process, ensuring a structured approach to understanding and mitigating potential threats.

6.1. Architecture description

Our approach begins with an analysis aimed at identifying the impact of a specific attack by establishing a correlation between threats and *Hazardous Control Actions* (HCAs). This operation will lead to the definition of a list of attack goals that a malicious actor might aim to achieve, represented as a tuple $\langle t, e, v \rangle$ where:

- t denotes the type of threat, identified using the elements of the STRIDE model: *spoof, tamper, repudiate, leak information, and deny*;
- e indicates the targeted Control or Feedback Actions, which are depicted in Table 3;
- v refers to the value injected, typically indicating a False Data Injection (indicated with $\overline{v}$) or a Command Injection (indicated with *cmd*).

We identify a comprehensive list of attack goals within the architecture specified in Section 4.1. These attack goals are outlined in Table 4 and are derived from an in-depth examination of potential threats and interdependencies within the system interconnection.

In detail, attack steps a_1, a_2, a_3, and a_5 specifically involve unauthorized alterations, commonly classified as tampering, of command (CA_2, CA_3) or feedback (F_2, F_5) signals exchanged between system components. This malicious activity typically manifests through modifications to a command option (denoted as *cmd*) or the alteration of data values (denoted as $\overline{v}$) that are required for control actions or feedback. On the other hand, attack step a_4 involves the manipulation of feedback data (F_3) through a spoofing attack. This type of attack can deceive the system into accepting false information as legitimate, potentially leading to inappropriate actions based on incorrect data.

Table 4
Mapping of attack goals to Hazardous Control Actions (HCAs).

Attack step	Description	HCAs
a_1	(*tampering*, CA_2, *cmd*)	1
a_2	(*tampering*, CA_3, *cmd*)	3
a_3	(*tampering*, F_2, $\overline{v}$)	2
a_4	(*spoofing*, F_3, $\overline{v}$)	4
a_5	(*tampering*, F_5, $\overline{v}$)	5

Table 5
Relationships between Control Actions (CA), Feedback (F), and Network protocols.

CA/F	Network protocol
CA_2, F_2, CA_3	MSP over TCP
F_3	NMEA
F_5	I2C

Table 6
Security compliance of protocols and components against NIST 800-53 controls.

Protocol/component	Description
MSP	**DS-2**: Data-in-transit is protected
NMEA	**AC-14**: Permitted Actions Without Identification or Authentication
Geolocation system	**DS-6**: Integrity checking mechanisms are used to protect firmware integrity

The second step of this stage involves a thorough examination of each protocol to understand its implementation characteristics and security weaknesses. This is accomplished by mapping each protocol to its respective control or feedback action and subsequently linking the protocol to its associated security flaws. The outcomes of these activities are presented in Table 5.

The signals CA_2, F_2, and CA_3 utilize the MSP over TCP protocol for message exchange, which is explained in Section 4.1. The F_3 signal, on the other hand, employs the NMEA (National Marine Electronics Association) protocol for transmitting GPS localization data between the GNSS source and the geolocation module. This protocol is widely adopted in UAV systems due to its standardized formats for GPS data transmission, including location coordinates and navigation parameters. Lastly, F_5 uses the I2C protocol to enable the exchange of information between the IMU and the autopilot, with the aim of facilitating the transfer of UAV's acceleration-related information. Notably, I2C is well-suited for short-distance communication within the UAV's onboard systems.

The final step of this procedure involves identifying, within the designed and constructed architecture, any missing security controls among those outlined in NIST 800-53. The NIST 800-53 framework was selected for three primary reasons. First, the absence of a security control implementation indicates a potential vulnerability that an attacker might exploit to achieve their malicious objectives. Second, this framework enables the identification of specific remediation measures necessary to address the identified security gaps. Last, this architectural model supports the seamless identification of malicious actions required to achieve specific attack goals by employing MITRE ATT&CK techniques. Indeed, the authors of the NIST framework have meticulously aligned its underlying model with NIST 800-53 security controls, hence providing a comprehensive approach to understanding and mitigating the impact of vulnerabilities. Table 6 links protocols and components in our experimental setup to the specific security checks from NIST 800-53 that they fail to pass.

Specifically, we have identified three security flaws in our architecture related to the NMEA and MSP protocols, as well as the Geolocation

Module. The NMEA protocol does not incorporate authentication and encryption, leading to non-compliance with security control AC-14. Similarly, MSP, as an unencrypted communication protocol, does not fulfill control requirement DS-2. Last, the Geolocation System does not comply with control DS-6. This failed check indicates that integrity-checking mechanisms for firmware are not implemented, which might lead to compromised firmware integrity.

6.2. Attack graph generation

After the definition of malicious goals, we aim to determine the sequence of attack actions necessary to reach such goals. We employ *Attack Graphs* (AG) to individuate these attack sequences. Indeed, AG provides a standard representation to describe all possible attack steps needed to achieve a specific objective (Schneier, 1999).

It is important to keep in mind that this representation relies on the principle of monotonicity (Muñoz-González et al., 2019; Ammann et al., 2002), which guarantees that a successful execution of an offensive action will never invalidate the precondition of another one. This cardinal principle enables us to eliminate redundant pathways, which leads to a structure called a *Directed Acyclic Graph* (DAG). A DAG is a graph where directed transitions connect nodes in a specific direction without cycles or loops. Consequently, Dijkstra's algorithm is the best choice for identifying the shortest path to reach a specific attack goal. Indeed, all attack paths must begin from the same starting point and the algorithm's complexity is $O((|V| + |E|) \times \log |V|)$, where V and E represent, respectively, the set of vertices and the set of edges in the original graph.

6.3. Attack paths extraction

In order to identify Attack Paths, we leveraged a logic-based network security analyzer called MulVal (Ou et al., 2005, 2006). This tool takes the system's architecture, a list of component vulnerabilities, and a set of attack goals as input to generate attack paths that lead to the desired objective. MulVal uses facts and rules to infer the attacker's progression throughout the network architecture. Facts are logical predicates that represent security attributes such as privileges, vulnerabilities, and network connectivity. Meanwhile, rules describe the relationships among these facts and are expressed through Horn clauses (Khan et al., 2017). In our specific context, these rules express either attack behaviors modeled through the MITRE ATT&CK framework, or environmental rules required to map the STPA outcomes onto the architecture description. The attack graph representation generated by MulVal is depicted in Fig. 4. This output highlights the following elements:

- *Fact nodes* (rectangles), also called *primitive nodes*, represent the asset state, configuration, or network condition that must exist for the attack to exploit the vulnerability.
- *Privilege nodes* (diamonds), also called *derived nodes*, represent the impact of the attack, such as information or resources obtained by an attacker.
- *Action nodes* (circles), also called *derivation* or *exploit nodes*, represent the actions an attacker would have to perform to obtain certain privileges.

To tailor MulVal to our specific needs, we modified the interaction rules file originally provided by the author of the tool. This file comprises two types of rules: primitive and interaction rules. Primitive rules are used to express facts in the *Attack Graph* (AG) produced by MulVal.

Interaction rules, on the other hand, define the logical relationships and conditions under which certain facts can lead to specific transitions in the attack graph. These rules are essential for modeling the dynamic behavior of potential attacks and the resulting system states.

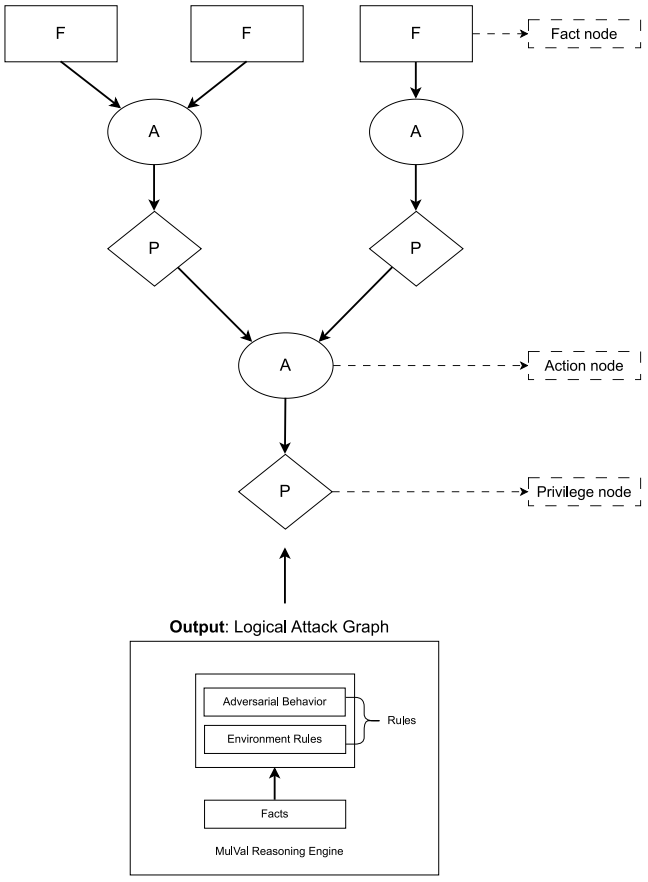


Fig. 4. MulVal logic elements and interaction.

7. A specialized attack graph for the OSA experimental setup

This section elucidates the process of generating a specific attack graph for the OSA architecture described in Section 4.1 by utilizing the "Attack Generation" methodology detailed in Section 6. This process involves two distinct stages. Initially, we define our architecture in an architecture file, which acts as the input for MulVAL. Subsequently, we extract three attack paths by applying the strategy outlined in Section 6.2. These attack paths pertain to the spoofing of F_4 and the tampering of CA_2 and CA_3, respectively. Each of these attack paths will be detailed in the following subsections. Moreover, to demonstrate the validity of the generated attack graph (and related attack paths), we will execute the attacks described by these graphs in Section 8.

7.1. Architecture description

This section provides a high-level analysis of the architecture file description for our experimental setup utilized by MulVAL.[1]

The file in question first specifies the attacker's goals of triggering a desired Hazardous Control Action. Specifically, we aim to tamper with CA2 and CA3, and spoof F3, leading to HCA1, HCA3, and HCA4, respectively (see Table 4). Then, it defines the attacker's locations, encompassing every possible position within our setup (e.g., having full physical access to the drone, or being in close proximity to the drone and being capable of interacting with it using radio frequencies). Feedback and control actions involved in the experimental setup are

[1] The complete source file associated with the architecture description can be found on the project's github repository (NS-UniNa, 2024).

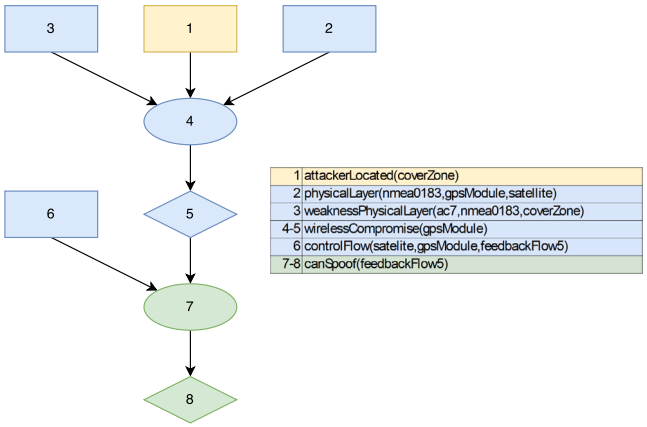


Fig. 5. GPS spoofing attack.

also defined. Furthermore, we specify that all components within the experimental setup can observe traffic exchanged between parties, even if they are not directly involved in the message exchange associated with the targeted Control Action or Feedback. Finally, the file lists all vulnerabilities related to the components and protocols employed within the architecture. These security flaws are enumerated in Table 6.

7.2. GPS spoofing

In this attack path, the attacker spoofs the F_4 signal received by the Geolocation module to trigger *Hazardous Control Action 4* (HCA_4). This malicious action deceives the Geolocation system by broadcasting false GPS signals, causing the drone to receive incorrect positional data and lose the planned route (Hazards H_3 and H_5 in Table 2). Such an incident can result in collisions with obstacles or terrain (Accident A_6 in Table 1), posing a risk of injury to individuals and/or drone destruction (L_1 and L_2).

In detail, as depicted in Fig. 5, the attacker takes advantage of their proximity (*node 1*) to the drone and the absence of mechanisms to authenticate information provided by satellite constellations to inject a fake position into the geolocation system. Indeed, the NMEA protocol used to share this data between GNSS sources and the geolocation system (*node 2*) does not provide any signing mechanism to prevent the alteration or replication of transmitted messages. This weakness, modeled using NIST security control AC-14 (*node 3*), creates the preconditions for attacks described by the Wireless Compromise technique in the MITRE ATT&CK ICS Matrix (*node 4-5*). This technique involves gaining unauthorized access to a wireless network, which allows the capture and replay of transmitted signals. Consequently, the privileges acquired through this initial access technique are sufficient to spoof signal F_4 (*node 6-8*).

7.3. False data injection

This scenario demonstrates a multi-stage attack against the Autopilot, aimed at tampering with the flow associated with Control Action 3, hence triggering Hazardous Control Action 3 (HCA_3). This objective is achieved by performing a *False Data Injection* (FDI) attack, which leads to hazards and impacts similar to the GPS spoofing attack described in Section 7.2. Specifically, the first stage involves gaining execution privileges on the Geolocation module, the second stage involves conducting network sniffing to gather information about the network, and the last stage involves tampering with CA3's flow. These stages are depicted in red, blue, and orange in the attack path in Fig. 6.

In the first step, the attacker alters the firmware of the localization system (*nodes 3–4*). This malicious activity is made possible by having direct access to the UAV drone system (*node 1*), as well as by the

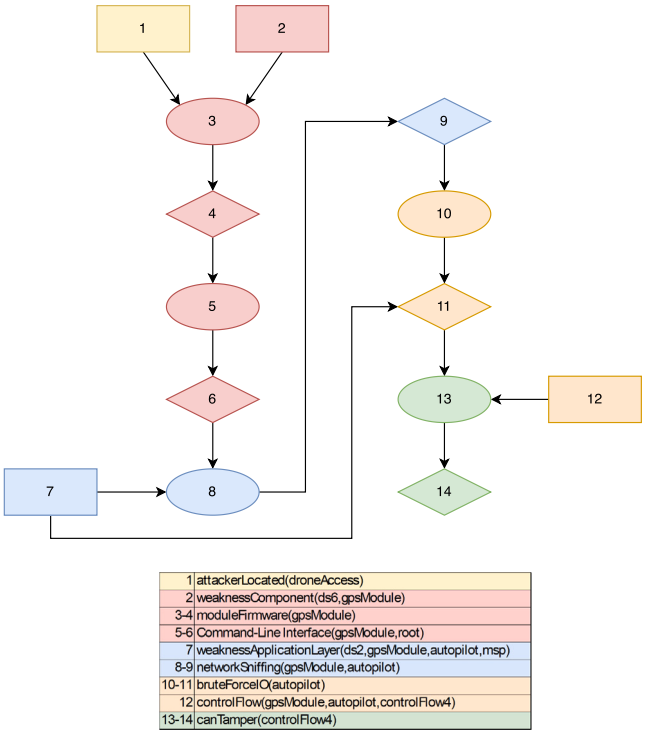


Fig. 6. False Command Injection in the geolocation module. (For interpretation of the references to color in this figure legend, the reader is referred to the web version of this article.)

lack of integrity checks on the UAV firmware, as indicated by NIST DS-2 Security Control (*node 2*). It is noteworthy that this modification also enables the execution of the system with root privileges (*nodes 5–6*). Leveraging these privileges, the attacker sniffs the network traffic to discover the components that communicate with the geolocation system, specifically the Autopilot (*nodes 9–10*). These privileges are modeled using the module firmware and Command Line Interface techniques. Subsequently, the attacker changes I/O value (*node 11–12*) exchanges between the Geolocation module and the Autopilot to tamper with Control Action 3 (*node 13–14*). Network spoofing and tampering are enabled by the lack of encryption in communication protocols (*node 7*), allowing attackers to intercept and manipulate data without detection.

7.4. False command injection

This attack path demonstrates how an attacker, masquerading as the Radio Controller, can tamper with Control Flow CA_2 and send false commands to the Autopilot. This malicious behavior can trigger *Hazard Control Action 2* (HCA_2), resulting in the loss of drone control and potentially leading to accidents (A_4) that endanger both the drone and human lives (L_1 and L_2). This sequence involves three distinct stages, as visually depicted in Fig. 7 and delineated in red, blue, and orange.

This attack path exhibits similarities to the one outlined in Section 7.3 and will therefore not be discussed in depth. Indeed, The sequence of malicious actions is identical and follows the same order. The only difference lies in *node 7*, which specifies that the geolocation node affected by the attacker, once compromised, has visibility of the traffic flowing between the autopilot and the Radio Controller. This indicates that the communication channel is not segregated but rather shared between these parties.

8. Attacks emulation

This section delves into the approach used to manipulate GPS coordinates and radio control commands of a target drone. The

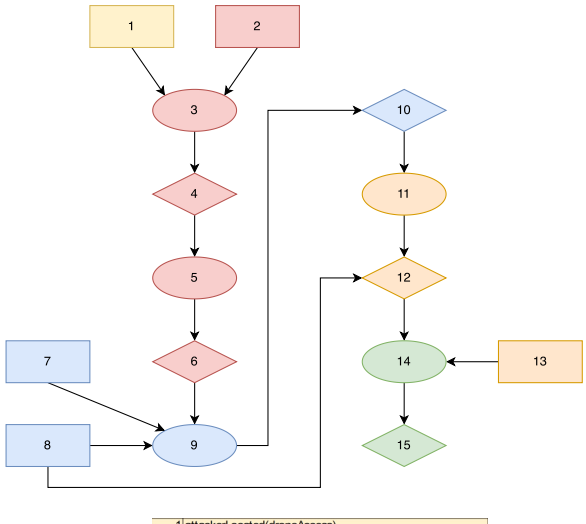


Fig. 7. False Command Injection targeting the radio controller. (For interpretation of the references to color in this figure legend, the reader is referred to the web version of this article.)

attack emulation has two primary goals: (i) demonstrate the testing capabilities of the developed OSA architecture by integrating two different simulation approaches, namely, Software-in-the-Loop (SITL) and Hardware-in-the-Loop (HITL); and (ii) validate the effectiveness and impact of the attacks discussed in the previous section.

The attacks are executed through the combined application of False Data Injection (FDI) for GPS data and False Command Injection (FCI) for radio control commands. In two distinct scenarios, the attacker manipulates control actions to send false data to the Autopilot System. In a third scenario, the attacker spoofs a physical communication channel to alter the positional data calculated by the geolocation system. The execution of these attacks precisely follows the paths outlined in the attack graph generated using MulVAL, as illustrated in Section 7. To further facilitate the execution of the discussed attacks, we have included an Attacker node in our testbed that provides utilities enabling others to replicate the malicious actions described in this work.

8.1. False data injection and false command injection

This attack aims to take control of the drone by manipulating either the GPS signal supplied to the autopilot from the Geolocation module (CA_3) or the radio controller signal (CA_2) transmitted by the radio controller to the autopilot. The execution of this attack, which consists of three distinct stages, adheres to the specifications outlined in the attack path detailed in Sections 7.3 and 7.4. For clarity, we associate each emulated malicious action using the following notation: (*node X - FDI*) for the attack path in Section 7.3, (*node X - FCI*) for the attack path in Section 7.4, and (*node X*) for actions common to both.

In the initial phase of this attack, we assume that the attacker already possesses the capability to execute commands on the Localization System (*node 6*). This assumption is grounded in the hypothesis that the attacker has direct physical access to the drone and can modify the firmware of the geolocation system (*node 4*). Consequently, the attacker can rely upon direct shell access to the Docker container dedicated to GPS. Subsequently, reconnaissance activities of the UAV network are conducted to identify active devices, open ports, and running services. The network analysis tool Nmap (nmap, 2024) is employed for this purpose.

The scan reveals the presence of three active hosts within the network, each serving a specific function: one host associated with the controller, another dedicated to the GPS, and a third host related to the remote control.

In the subsequent phase, we leverage Packet Sniffing techniques to gather sensitive information and conduct an in-depth network traffic analysis (*node 8–9 FDI* and *node 9–10 FCI*). Specifically, we utilize Wireshark, a network protocol analyzer, to sniff all network traffic running on the geolocation system and extract sensitive information from it.

This phase is crucial for understanding the communication patterns and identifying potential vulnerabilities within the UAV network.

Traffic analysis reports packets exchanged between the autopilot (i.e., Flight Controller) and the geolocation system (i.e., GPS), as well as between the autopilot and the radio controller. An in-depth inspection of this traffic reveals that the data is easily readable due to the use of unencrypted TCP sessions for transmitting data. Specifically, it is observed that the data exchanged between these components within the drone's network uses MSP over TCP for the data exchange (*node 7 FDI* and *node 8 FCI*).

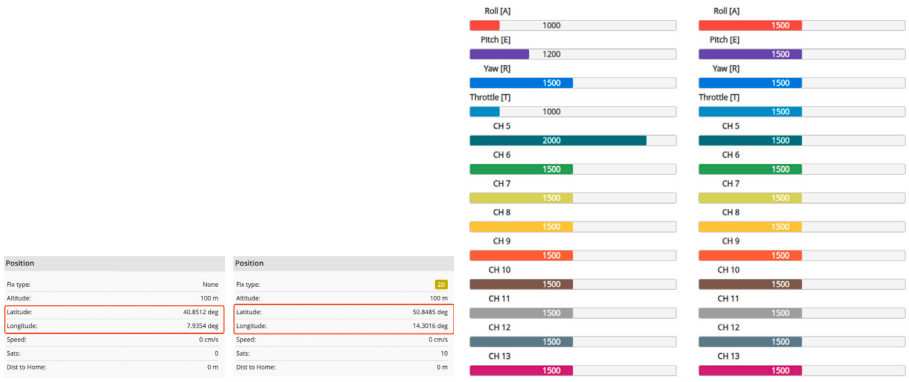
In the final phase, the attacker analyzes the MSP payload structure to determine how to manipulate data associated with CA2 and CA3 (*node 13–14 FDI* and *node 14–15 FCI*). Specifically, the attacker alters each packet with type 201 to falsify the GPS-reported position sent from the geolocation system and forge new packets with type 200 to guide the drone to a specified location, as the radio controller sends these packets (*node 10–11 FDI* and *node 11–12 FCI*). Notably, these changes also require the recalculation of the CRC before messages are injected into the network and sent to the target host. The checksum is calculated using XOR operations that involve the payload length, type, and data. The outcomes of these actions are illustrated in Fig. 8, which clearly shows how both data and commands have been successfully tampered with by the attacker.

8.2. GPS spoofing

In this subsection we explore physical attacks as a further important category of threats. Specifically, our attention is directed toward a form of physical attack that challenges trust in navigation systems, known as GPS Spoofing. This attack is particularly concerning in scenarios where location accuracy is crucial, such as drone operations. The attack path of this malicious activity is depicted in Section 7.2. For clarity, we use the notation (*node X*) to highlight the correlation between the attack demonstration and the attack path.

Before describing the attack sequence, we will investigate the details of the hardware and software tools required to exploit the vulnerability mentioned. First, due to the nature of this attack, it is essential to implement a real GPS in our experimental setup. This task can be accomplished by following the procedure described in Section 4.1. Alternatively, a commercial drone can be used to achieve similar effects. Moreover, we employ the HackRF One (Great Scott Gadgets, 2024) device to spoof the GNSS signal in close proximity to the UAV (*node 3*). In our case, a DJI Mini commercial drone is used to demonstrate the effects of the attack. The HackRF One device is connected to a laptop via a USB cable to generate spoofed GNSS signals near the UAV, disrupting its navigation. This setup can be employed to alter the UAV's perceived position in real-time, hence allowing to take control over the UAV's trajectory.

For the attack's implementation, we commence by procuring RINEX navigation files containing GPS data for injection. These data are collected from the latest files on the NASA website (NASA CDDIS GNSS Data, 2024). The downloaded RINEX file undergoes processing using a specialized application, namely gps-sdr-sim (2024), to generate GPS baseband signal data streams congruent with the HackRF One device.



(a) FDI - Before (b) FDI - After (c) FCI - Before (d) FCI - After

Fig. 8. False Data Injection (FDI) and False Command Injection (FCI) outputs. (a) and (b) show the effects of FDI before and after the attack, where GPS coordinates and position data are falsified. (c) and (d) depict FCI before and after, where command values like throttle and roll are altered, misleading the system's control inputs and outputs. FDI corrupts sensor data, while FCI manipulates control signals, both leading to incorrect system behavior.

Finally, the output is relayed by HackRF One to inject the falsified signals into the UAV's navigation system (*node 6–8*). The necessary parameters for this process include the name of the input file containing the previously generated data, the center frequency of the signal, the sampling rate, and the amplitude level of the signal. The attack reaches its goal, that is manipulating the drone's GPS signals to make it believe it is located in a different geographic location than its actual position.

9. Discussion

In this section, we first discuss how we have addressed some of the limitations of state-of-the-art cyber-resilience models. Then, we provide a comparative analysis of our microservices-based architecture against traditional monolithic architectures in terms of security threat mitigation and components substitution effort. Lastly, we offer insights into how the proposed experimental setup can be extended in order to integrate more advanced security controls in the architecture.

9.1. Addressing cyber-resilience models limitations

The approach we have proposed extends the one introduced by Castiglione and Lupu (2023), by integrating two powerful frameworks: the MITRE ATT&CK matrix and NIST 800-53. The MITRE ATT&CK matrix plays a crucial role in identifying and categorizing the potential tactics, techniques, and procedures (TTPs) used by adversaries in real-world cyberattacks, which enables a more detailed and precise mapping of vulnerabilities within the UAV system. Meanwhile, the integration of NIST 800-53 introduces a standardized framework for identifying security controls and managing security risks. This provides us with a dual advantage: (i) it allows for the systematic identification of weaknesses and the potential exploits linked to them, and (ii) it helps in determining which security control checks should be applied to safeguard the architecture.

By leveraging these two frameworks, our approach not only enhances the capability to define specific vulnerabilities and their exploit paths but also ensures that there is a structured and comprehensive set of security measures in place to mitigate the identified risks.

In evaluating the effectiveness and comprehensiveness of various cyber-security resilience models, it is indeed essential to examine how they integrate different security features and methodologies. Table 7 provides a detailed comparative analysis of our proposed approach in relation to other prominent models in the field. This table highlights several key dimensions: whether the approach incorporates intentional Attack Scenarios (AS), detects Attack Paths (AP), addresses Safety (S), and includes Formal Verification (FV). Furthermore, it specifies whether the models utilize the MITRE ATT&CK matrix (MITRE) and the NIST 800-53 checklist (NIST). By presenting this comparative overview, the table underscores the strengths of our approach, particularly its integration of diverse methodologies and frameworks, which collectively enhance its ability to provide robust and comprehensive cyber-resilience. This detailed comparison not only showcases the unique attributes of our model but also illustrates how it builds upon and extends the capabilities of existing approaches in addressing a wider range of security challenges.

Table 7
Comparison with other cyber-security resilience models.

Model	AS	AP	S	FV	MITRE	NIST
Young and Leveson (2013)				✓		
Friedberg et al. (2017)	✓		✓			
Zenitani (2023)	✓	✓				
Castiglione et al. (2023)	✓	✓	✓	✓		
Our work	✓	✓	✓	✓	✓	✓

9.2. Monolithic and microservices-based architectures: a comparative analysis

This subsection evaluates the effectiveness of our approach with regard to two main functions: (i) security threat mitigation; (ii) effectiveness of the components substitution process. The analysis is carried out by benchmarking our microservices-based architecture against traditional monolithic solutions.

From a security standpoint, we utilized the MITRE ATT&CK SPARTA tactics (SPARTA, 2024) to highlight the advantages of adopting a microservices architecture. Our findings indicate significant improvements in mitigating threats across five out of nine analyzed categories when employing a microservices-based approach:

- *Execution*: microservices allow for the isolation of execution within individual services, minimizing the risk of a widespread compromise.
- *Persistence*: the containerization inherent in microservices makes it more challenging for attackers to establish persistence. They can only access the compromised container, lacking visibility or control over the broader system.
- *Defense Evasion*: the microservices-based architecture facilitates the integration of security controls, enabling a more effective implementation of cyber defense mechanisms, as discussed in Section 9.3.
- *Lateral Movement*: microservices are designed to restrict lateral movement across services, effectively containing potential threats.

Table 8
Comparison of change impact and testing efforts in sensor integration.

Task	Monolithic architecture	Microservices architecture
Change impact analysis	Add the new driver (4 files) and modify high-level libraries (acceleration.c, gyro.c)	Modify only the sensor-related microservice Dockerfile (1 file)
Testing	Regression testing is complex due to tight coupling (Variables/functions referenced in 112 files)	Modify only the sensor-related microservice Dockerfile (1 file)

- *Impact*: an attack on one service is less likely to affect other parts of the data flow within the system, reducing the overall impact of a breach.

Despite the advantages of a microservices architecture in mitigating various security threats, our analysis reveals no significant differences between microservices and monolithic architectures for the remaining tactics (*Reconnaissance*, *Resource Development*, *Initial Access*, and *Exfiltration*). For these, risks exist similarly in both architectures, with no significant differences in their mitigation capabilities.

To evaluate the effort involved in switching from an originally supported component within the INAV flight controller to a new component, we conduct a change impact analysis. This analysis compares the implementation and testing processes of integrating a new IMU (Inertial Measurement Unit) sensor into the flight controller under two different architectural frameworks: a traditional monolithic architecture and a microservices-based architecture.

We assess the impact of these modifications using metrics similar to those in existing literature (Misirli et al., 2016; Mondal et al., 2018), which analyze the distribution of modified code across files. The deployment process entails two distinct steps: (i) *Driver Integration*: the developer must add the new driver for the IMU sensor, which in the analyzed case requires the creation of four new files, and (ii) *Library Modifications*: the high-level libraries for the existing acceleration and gyroscope sensors must be updated to enable the system to utilize data from the new IMU sensor.

In traditional architectures, regression testing presents significant challenges due to the tightly coupled nature of the system. In the sample case we have used as a benchmark, the variables and functions referenced in these new files are invoked in more than 100 other files, complicating the integration and testing processes. In contrast, our microservices-based architecture mitigates these challenges by allowing isolated updates to individual components. In this scenario, only the specific sensor-related microservice (i.e., a single Dockerfile) requires modification, provided it adheres to the established open interface between the sensor and the flight controller. This architectural decoupling significantly simplifies the implementation and testing processes by reducing the scope of code changes, thus streamlining the overall development effort. Table 8 summarizes the above findings.

9.3. Integrating security controls in the architecture

In this work, we have identified several vulnerabilities through the application of our cyber-resilience model. However, we do not yet address these vulnerabilities by directly integrating security controls into the system's infrastructure. The modular nature of the proposed container-based virtualization framework significantly simplifies the process of adding security controls by enabling the seamless integration of additional containers. With appropriate configurations, it becomes feasible to incorporate a range of security capabilities into the system, such as a lightweight firewall or an Intrusion Detection System (IDS). This further enhances the protection of the UAV system against potential threats.

A lightweight firewall can be implemented by introducing a router container as a gateway to the experimental setup architecture, as illustrated in Fig. 9. The firewall could enforce simple allow/deny rules based on services or ports, filtering traffic at a basic level, but it could also be configured with more advanced features, such as deep packet inspection, to protect against application-level attacks.

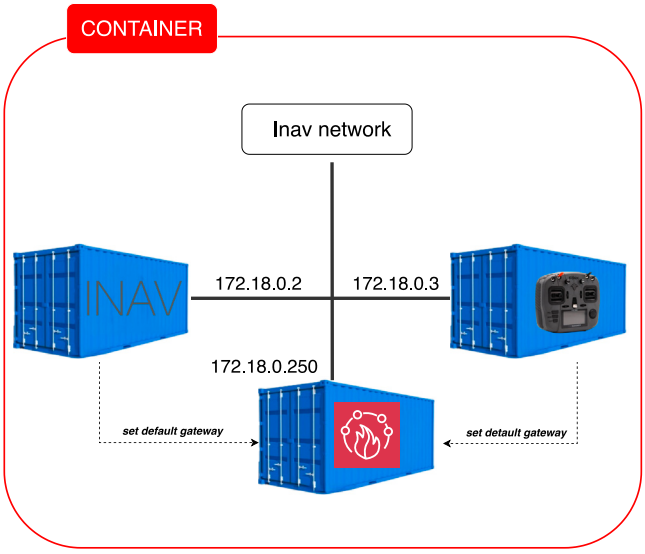


Fig. 9. Integrating a firewall into the experimental setup.

Docker's built-in capabilities greatly simplify the integration of security controls designed to analyze traffic in transit, such as Intrusion Detection Systems (IDS). By using Docker, it is possible to attach a monitoring container directly to a specific container's network interface, enabling transparent traffic analysis without disrupting the system's normal operation. This method, as demonstrated in the "netshoot" documentation,[2] offers a straightforward approach to monitoring.

In traditional networking environments, this process would involve setting up a so-called 'tap' adapter to mirror traffic to a dedicated Ethernet interface. However, Docker's flexible networking configuration allows for this setup to be achieved effortlessly, giving security testers the ability to integrate external security modules such as IDS or anomaly detection systems directly into the system with minimal effort.

10. Conclusions

This study delivers insights into the vulnerabilities of traditional monolithic UAV architectures and demonstrates the potential of transitioning to a microservices-based system aligned with Open System Architecture (OSA) principles. Key findings from our work include:

- Identification of Vulnerabilities: we successfully exposed the weaknesses in conventional UAV systems through a detailed threat modeling process, revealing how compromised components can affect the UAV's flight path via malicious data and commands;
- Validation of the OSA Approach: our experiments with a Software -in-the-Loop (SITL) and Hardware-in-the-Loop (HITL) testbed validate the effectiveness of the OSA-based architecture in enhancing system flexibility, scalability, and resilience to cyber threats;
- Novel Cyber-Resilience Model: by integrating state-of-the-art frameworks such as NIST 800-53, MITRE ATT&CK, and STPA-Sec, we proposed a comprehensive model for identifying and

[2] Netshoot (nicolaka, 2016) is a renowned network troubleshooting "Swiss-army-knife" container, widely recognized for its versatility in diagnosing and resolving network issues within containerized environments.

analyzing attack paths and vulnerabilities, offering a significant advancement over existing methods.

Future research will build upon these findings with three specific directions. Firstly, we aim to broaden the testbed to include additional critical subsystems, such as advanced communication, navigation, and sensor systems, to achieve a more comprehensive assessment of UAV security and performance. Secondly, we will focus on identifying and modeling new attack vectors to refine our cyber-resilience model. This includes exploring advanced threat scenarios and enhancing our approach to address a wider range of potential cyberattacks. Finally, further improvements will be made to the microservices-based design to optimize security, performance, and scalability. This involves iterative testing and integration of additional security controls and technologies to meet evolving operational demands.

In summary, this work provides a foundational framework for developing more secure and adaptable UAV systems through the application of microservices and open standards. By addressing both the limitations of monolithic architectures and enhancing cyber-resilience, we pave the way for future advancements in UAV technology that can effectively respond to new challenges and threats.

CRediT authorship contribution statement

Nicola d'Ambrosio: Writing – original draft, Visualization, Validation, Software, Methodology, Investigation, Formal analysis, Data curation, Conceptualization. **Gaetano Perrone:** Writing – original draft, Validation, Software, Resources, Methodology, Investigation, Formal analysis, Data curation, Conceptualization. **Simon Pietro Romano:** Writing – review & editing, Writing – original draft, Validation, Supervision, Project administration, Methodology, Investigation, Formal analysis, Data curation, Conceptualization. **Alberto Urraro:** Validation, Software, Resources, Methodology, Investigation, Formal analysis, Data curation, Conceptualization.

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.

Acknowledgments

This work has been funded by the project Azione IV.4 - Dottorati e contratti di ricerca su tematiche dell'innovazione, Italy - DM 1061 del 10 agosto 2021 (CUP: E65F21003690003).

Data availability

All of our code and data are publicly available in the paper's public github repository, which is also reference number NS-UniNa (2024) in the manuscript: https://github.com/NS-unina/GCAP_Mod.

References

Ahn, B., Kim, T., Smith, S.C., Youn, Y.-W., Ryu, M.-H., 2021. Security threat modeling for power transformers in cyber-physical environments. In: 2021 IEEE Power & Energy Society Innovative Smart Grid Technologies Conference. ISGT, IEEE, pp. 1–5.

Ammann, P., Wijesekera, D., Kaushik, S., 2002. Scalable, graph-based network vulnerability analysis. In: Proceedings of the 9th ACM Conference on Computer and Communications Security. CCS '02, Association for Computing Machinery, New York, NY, USA, pp. 217–224, [Online]. Available: https://doi.org/10.1145/586110.586140.

Arthur, H.M., 2018. "Cunter-drone systems",center for the study of the drone, bard college. [Online]. Available: https://dronecenter.bard.edu/files/2019/12/CSD-CUAS-2nd-Edition-Web.pdf.

Ates, S., Bayezit, I., Inalhan, G., 2009. Design and hardware-in-the-loop integration of a uav microavionics system in a manned–unmanned joint airspace flight network simulator. J. Intell. Robot. Syst. 54, 359–386.

Boniol, F., Wiels, V., 2014. Towards modular and certified avionics for uav aerial robotics. Aerosp. Lab (8), 1–8.

Castiglione, L.M., Lupu, E.C., 2023. Which attacks lead to hazards? Combining safety and security analysis for cyber-physical systems. IEEE Trans. Dependable Secure Comput. 1–16.

Castiglione, L.M., Stassen, P., Perner, C., Pereira, D.P., de Carvalho Bertoli, G., Lupu, E.C., 2023. Don't panic! analysing the impact of attacks on the safety of flight management systems. In: 2023 IEEE/AIAA 42nd Digital Avionics Systems Conference. DASC, pp. 1–10.

De Souza, N.P., César, C.d.C., de Melo Bezerra, J., Hirata, C.M., 2020. Extending STPA with STRIDE to identify cybersecurity loss scenarios. J. Inf. Secur. Appl. 55, 102620.

Friedberg, I., McLaughlin, K., Smith, P., Laverty, D., Sezer, S., 2017. STPA-SafeSec: Safety and security analysis for cyber-physical systems. J. Inf. Secur. Appl. 34, 183–196.

gps-sdr-sim, 2024. Osqzss. [Online]. Available: https://github.com/osqzss/gps-sdr-sim. (Accessed 28 May 2024).

Great Scott Gadgets, 2024. HackRF one. [Online]. Available: https://greatscottgadgets.com/hackrf/one/. (Accessed 28 May 2024).

Haider, M.H., Saleem, S.B., Rafaqat, J., Sabahat, N., 2019. Threat modeling of wireless attacks on advanced metering infrastructure. In: 2019 13th International Conference on Mathematics, Actuarial Science, Computer Science and Statistics. MACS, IEEE, pp. 1–6.

Immanuel, G.-Y., Johnson, E., 2002. New architectures for UAV flight control avionics. In: Proceedings. the 21st Digital Avionics Systems Conference. Vol. 2, p. 8B4.

iNavFlight, 2024. INAV-flight controller. [Online]. Available: https://github.com/iNavFlight/inav. (Accessed 28 May 2024).

Irizarry, K.A., Zhang, Z., Stewart, C., Boubin, J., 2022. Scalable distributed microservices for autonomous uav swarms. In: Proceedings of the 23rd International Middleware Conference Demos and Posters. pp. 1–2.

Ishimatsu, T., Leveson, N.G., Thomas, J., Katahira, M., Miyamoto, Y., Nakao, H., 2010. Modeling and Hazard Analysis Using STPA. International Association for the Advancement of Space Safety (IAASS).

Kerns, A.J., Shepard, D.P., Bhatti, J.A., Humphreys, T.E., 2014. Unmanned aircraft capture and control via GPS spoofing. J. Field Robotics 31 (4), 617–636.

Khalil, S.M., Bahsi, H., Ochieng'Dola, H., Korõtko, T., McLaughlin, K., Kotkas, V., 2023. Threat modeling of cyber-physical systems-a case study of a microgrid system. Comput. Secur. 124, 102950.

Khan, R., McLaughlin, K., Laverty, D., Sezer, S., 2017. STRIDE-based threat modeling for cyber-physical systems. In: 2017 IEEE PES Innovative Smart Grid Technologies Conference Europe. ISGT-Europe, pp. 1–6.

Leveson, N., 2004. A new accident model for engineering safer systems. Saf. Sci. 42 (4), 237–270.

Mansfield, K., Eveleigh, T., Holzer, T.H., Sarkani, S., 2013. Unmanned aerial vehicle smart device ground control station cyber security threat model. In: 2013 IEEE International Conference on Technologies for Homeland Security. HST, IEEE, pp. 722–728.

Matlekovic, L., Juric, F., Schneider-Kamp, P., 2022. Microservices for autonomous UAV inspection with UAV simulation as a service. Simul. Model. Pract. Theory 119, 102548.

Misirli, A.T., Shihab, E., Kamei, Y., 2016. Studying high impact fix-inducing changes. Empir. Softw. Eng. 21, 605–641.

MITRE ATT&CK, 2024. MITRE ATT&CK. [Online]. Available: https://attack.mitre.org/. (Accessed 28 May 2024).

Mondal, M., Rahman, M.S., Roy, C.K., Schneider, K.A., 2018. Is cloned code really stable? Empir. Softw. Eng. 23, 693–770.

Muñoz-González, L., Sgandurra, D., Barrère, M., Lupu, E.C., 2019. Exact inference techniques for the analysis of Bayesian attack graphs. IEEE Trans. Dependable Secure Comput. 16 (2), 231–244.

NASA CDDIS GNSS Data, 2024. NASA. [Online]. Available: https://cddis.nasa.gov/archive/gnss/data/daily/. (Accessed 28 May 2024).

nicolaka, 2016. Netshoot. Online; https://github.com/nicolaka/netshoot. 10 September 2024.

NIST, 2024. Security and privacy controls for information systems and organizations. [Online]. Available: https://www.docker.com/. (Accessed 28 May 2024).

nmap, 2024. Nmap - User guide. [Online]. Available: https://nmap.org/man/it/index.html. (Accessed 28 May 2024).

NS-UniNa, 2024. GCAP-MOD repo. [Online]. Available: https://github.com/NS-unina/GCAP_Mod. (Accessed 28 May 2024).

Ou, X., Boyer, W.F., McQueen, M.A., 2006. A scalable approach to attack graph generation. In: Proceedings of the 13th ACM Conference on Computer and Communications Security. pp. 336–345.

Ou, X., Govindavajhala, S., Appel, A.W., et al., 2005. MulVAL: A logic-based network security analyzer. In: USENIX Security Symposium. Vol. 8, Baltimore, MD, pp. 113–128.

Pastor, E., Barrado, C., Royo, P., Lopez, J., Santamari, E., 2009. An open architecture for the integration of UAV civil applications. In: Aerial Vehicles. InTech, [Online]. Available: http://dx.doi.org/10.5772/6488.

Perrone, G., d'Ambrosio, N., Capodagli, G., Romano, S.P., 2024. Scass: Breaking into scada systems security. Available at SSRN 4750612.
px4, 2024. px4 autopilot. [Online]. Available: https://px4.io/. (Accessed 28 May 2024).
Rasmussen, J., 1997. Risk management in a dynamic society: a modelling problem. Saf. Sci. 27 (2–3), 183–213.
Sage, A.P., 1988. Systems Engineering. Wiley.
Sahay, R., Estay, D.S., Meng, W., Jensen, C.D., Barfod, M.B., 2023. A comparative risk analysis on CyberShip system with STPA-Sec, STRIDE and CORAS. Comput. Secur. 128, 103179.
Salgado, M.L., de Sousa, M.S., 2021. Cybersecurity in aviation: the STPA-sec method applied to the TCAS security. In: 2021 10th Latin-American Symposium on Dependable Computing. LADC, pp. 1–10.
Sánchez, H.S., Rotondo, D., Vidal, M.L., Quevedo, J., 2019. Frequency-based detection of replay attacks: application to a quadrotor UAV. In: 2019 8th International Conference on Systems and Control. ICSC, pp. 289–294.
Sanchez-Lopez, J.L., Molina, M., Bavle, H., Sampedro, C., Suárez Fernández, R.A., Campoy, P., 2017. A multi-layered component-based approach for the development of aerial robotic systems: The aerostack framework. J. Intell. Robot. Syst. 88, 683–709.
Sayers, J.M., Feighery, B.E., Span, M.T., 2020. A STPA-sec case study: Eliciting early security requirements for a small unmanned aerial system. In: 2020 IEEE Systems Security Symposium. SSS, pp. 1–8.
Schmittner, C., Ma, Z., Puschner, P., 2016. Limitation and improvement of STPA-sec for safety and security co-analysis. In: Computer Safety, Reliability, and Security: SAFECOMP 2016 Workshops, ASSURE, DECSoS, SASSUR, and TIPS, Trondheim, Norway, September 20, 2016, Proceedings 35. Springer, pp. 195–209.
Schneier, [Online]. Available: https://www.schneier.com/academic/archives/1999/12/attack_trees.html.
Shakhatreh, H., Sawalmeh, A.H., Al-Fuqaha, A., Dou, Z., Almaita, E., Khalil, I., Othman, N.S., Khreishah, A., Guizani, M., 2019. Unmanned Aerial Vehicles (UAVs): A survey on civil applications and key research challenges. IEEE Access 7, 48572–48634.
SPARTA, 2024. SPARTA tactics. Online https://sparta.aerospace.org/tactic/SPARTA. 10 September 2024.
The Open Group, 2024. Service-oriented architecture. [Online]. Available: https://collaboration.opengroup.org/projects/soa-book/. (Accessed 28 May 2024).
Tokar, J.L., 2017. A comparison of avionics open system architectures. ACM SIGAda Ada Lett. 36 (2), 22–26.
Towler, J., Bries, M., 2018. ROS-military: Progress and promise. In: Proceedings of the 2018 Ground Vehicle Systems Engineering and Technology Symposium. GVSETS.
Williams, A.D., 2024. System theoretic process analysis (STPA). [Online]. Available: https://www.osti.gov/servlets/purl/1645411. (Accessed 28 May 2024).
Xu, H., Zhang, H., Sun, J., Xu, W., Wang, W., Li, H., Zhang, J., 2021. Experimental analysis of MAVLink protocol vulnerability on UAVs security experiment platform. In: 2021 3rd International Conference on Industrial Artificial Intelligence. IAI, pp. 1–6.
Yaacoub, J.-P., Noura, H., Salman, O., Chehab, A., 2020. Security analysis of drones systems: Attacks, limitations, and recommendations. Internet Things 11, 100218.
Young, W., Leveson, N., 2013. Systems thinking for safety and security. In: Proceedings of the 29th Annual Computer Security Applications Conference. pp. 1–8.
Young, W., Leveson, N.G., 2014. An integrated approach to safety and security based on systems theory. Commun. ACM 57 (2), 31–35.
Yu, Z., Wang, Z., Yu, J., Liu, D., Song, H., Li, Z., 2023. Cybersecurity of unmanned aerial vehicles: A survey. IEEE Aerosp. Electr. Syst. Mag..
Zenitani, K., 2023. Attack graph analysis: An explanatory guide. Comput. Secur. 126, 103081, [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0167404822004734.
Zhang, Y., Dong, C., Guo, W., Dai, J., Zhao, Z., 2022. Systems theoretic accident model and process (STAMP): A literature review. Saf. Sci. 152, 105596.

Nicola d'Ambrosio is currently an Ph.D. student at the Department of Electrical Engineering and Information Technology, University of Napoli Federico II. He has worked as a free-lance infrastructural penetration tester with SecSI, a startup and university spin-off that focuses on network security. He has also been a tutor at the Cyber HackAdemy, a professional cybersecurity training course in collaboration between the University of Napoli Federico II and Accenture.

Gaetano Perrone is currently CTO at SECurity Solutions for Innovation (SECSI), a company based in Naples focused on researching innovative solutions in the network security domain. Passionate in any area related to network security, cloud computing, and software development, he received a Ph.D. degree in computer engineering and systems from the University of Napoli Federico II in 2022. He continues conducting research in fuzzing, knowledge graphs, and reinforcement learning domains to automate, formalize and increase the effectiveness of security testing approaches.

Simon Pietro Romano is a Full Professor in the Department of Electrical Engineering and Information Technology at the University of Napoli Federico II. He lectures on computer networks, computer architectures, network security, and Web and Real-Time Communication Systems. Additionally, he is the co-founder of Meetecho, a startup and university spin-off specializing in scalable video streaming and WebRTC-based unified collaboration, as well as SECSI (SECurity Solutions for Innovation), which focuses on network security. Active in IETF (Internet Engineering Task Force) standardization activities, particularly in the Applications and Real-Time (ART) area, Simon Pietro Romano has been involved in networking research since 1998. He has made significant contributions to real-time multimedia applications, particularly in the design, implementation, and standardization of scalable, distributed architectures for conferencing, media control, and telepresence. Additionally, he has conducted research in network security, with a focus on distributed intrusion detection, critical infrastructure protection, and AI-based automation of Penetration Testing techniques. His research interests also include issues related to the distribution of control in 5G-compliant, container-based, virtualized architectures. Serving as the Director of the CINI laboratories in Naples, Simon Pietro Romano has extensive experience participating in R&D projects at both the national and international levels. He recently served as the Principal Investigator in the SHINE (Secure Hybrid In Network caching Environment) project funded by the European Space Agency (ESA) and currently leads the CINI research unit in three ESA-funded projects. Simon Pietro Romano is also the Scientific Director of the Accenture Cyber HackAdemy (which focuses on cyber training), Managing Director of the Apple Developer Academy, and Director of the ITEM National Laboratory at CINI (National Interuniversity Consortium for Informatics).

Alberto Urraro obtained a master's degree in computer engineering from the University of Napoli Federico II in 2023. He is passionate about security, with a focus on Critical Infrastructure Protection and Risk Assessment.