An Investigation of Byzantine Threats in Multi-Robot Systems

Gelei Deng
Nanyang Technological University
Singapore
gdeng003@e.ntu.edu.sg

Yuan Zhou*
Nanyang Technological University
Singapore
y.zhou@ntu.edu.sg

Yuan Xu
Institute of Computing Technology,
Chinese Academy of Sciences
China
xuyuan@ict.ac.cn

Tianwei Zhang
Nanyang Technological University
Singapore
tianwei.zhang@ntu.edu.sg

Yang Liu
Nanyang Technological University
Singapore
yangliu@ntu.edu.sg

ABSTRACT

Multi-Robot Systems (MRSs) show significant advantages to deal with complex tasks efficiently. However, the system complexity inevitably enlarges the attack surface and adds difficulty in guaranteeing the security and safety of MRSs. In this paper, we present an in-depth investigation about the Byzantine threats in MRSs, where some robot is untrusted. We design a practical methodology to identify potential Byzantine risks in a given MRS workload built from the Robot Operating System (ROS). It consists of three novel steps (requirement specification using signal temporal logic, attack surface determination via data-flow analysis, attack identification using requirement-driven fuzzing) to thoroughly assess MRS workloads. We use this fuzzing method to inspect five typical MRS workloads from past works and the ROS platform, and identify three novel kinds of attacks that can be launched with five attack strategies. We conduct comprehensive experiments in the Gazebo simulator and a real-world MRS with three TurtlBot3 robots to validate these attacks, which can remarkably decrease the system's performance, or even cause task failures.

CCS CONCEPTS

• **Computer systems organization → Robotics**; • **Security and privacy → Distributed systems security**.

KEYWORDS

multi-robot system, Byzantine threat, fuzzing

ACM Reference Format:
Gelei Deng, Yuan Zhou, Yuan Xu, Tianwei Zhang, and Yang Liu. 2021. An Investigation of Byzantine Threats in Multi-Robot Systems. In *24th International Symposium on Research in Attacks, Intrusions and Defenses (RAID '21), October 6–8, 2021, San Sebastian, Spain.* ACM, New York, NY, USA, 16 pages. https://doi.org/10.1145/3471621.3471867

*Corresponding author

RAID '21, October 6–8, 2021, San Sebastian, Spain

ACM ISBN 978-1-4503-9058-3/21/10…$15.00
https://doi.org/10.1145/3471621.3471867

1 INTRODUCTION

The robotics technology is becoming more popular and ubiquitous in our society. A variety of intelligent and autonomous robots were designed to significantly improve our work efficiency and quality of life. With the increased complexity of tasks and performance demands, Multi-Robot Systems (MRSs) have gained ever-growing attention. Multiple mobile robots are interconnected with each other within the same environment. They collaboratively work on an enormous task, which is hard to achieve by a single robot. Due to these benefits, MRSs have been developed for myriad scenarios and applications, such as precision agriculture [23, 24, 62, 89], minefield mapping [41, 46, 76, 85, 86], search and rescue [54, 59, 63].

The significance of MRSs calls for special attention to security, as the system complexity can increase the attack surface. Past works have demonstrated that a single robot device is vulnerable to a plethora of attacks from different components, including sensors [22, 27, 50, 81], actuators [20, 33], motion controller [68], Robot Operating System [19, 28] and applications [48, 50, 68]. These vulnerabilities enable an external adversary to easily intrude into the robot and take full control of the robot, resulting in Byzantine faults in an MRS. A Byzantine fault describes a condition of a distributed system where some components may fail and there is a lack of sufficient information to identify such failures [96]. In a distributed MRS, if one robot is compromised, it has the potential to affect other robots and even threaten the entire system, due to their close communication and collaboration. For instance, in 2021 January, a drone swarm got crashed during a light show in Chongqing, China. Up to 100 drones lost controls and hit into a building due to a small bug in the mainframe control [87].

Such Byzantine problem has been extensively studied in traditional distributed systems. However, it is relatively less explored in the context of MRSs. Prior work [10, 13, 58, 82] considered the Byzantine resilience in MRSs from a theoretical perspective. They simply treat each robot as a dot without considering the specific workloads, robots' capabilities and physical constraints. Hence, it is infeasible to apply these solutions and findings to the real-world Multi-Robot scenarios and tasks in a practical way.

We are particularly interested in two unsolved questions: *given the design or implementation of an MRS, (1) how can we identify the potential Byzantine threats and the optimal attack strategy? (2) how much damage can a Byzantine robot bring to the entire MRS?* Addressing these two questions is challenging. First, an MRS can

execute a variety of workloads with distinct characteristics and user requirements. The inter-robot communication and collaboration can be implemented with various mechanisms. So it is hard to design a unified method for vulnerability identification and assessment. Second, during the operation, a malicious robot has very high freedom to affect other robots and the entire system in unexpected ways. They exchange different types of messages, and each message has a large input space for the robot to tamper with. This makes it difficult to comprehensively search for potential attacks.

In this paper, we provide the *first* practical study towards the Byzantine threats in MRSs based on the Robot Operating System (ROS) [2]. ROS is the most popular open-source platform for robot app development. It provides thousands of packages to achieve various functions, compatible with mainstream robot devices. This platform has benefited the robotics research, as well as the development of commercial products in industry, e.g., Dji Matrice 200 drone [1], PR2 humanoid [7], and ABB manipulators [5][1].

We design a requirement-driven fuzzing methodology, which can automatically analyze a given MRS workload and identify the potential Byzantine risks. We consider a threat model where only one robot in an MRS is malicious. Our method can be easily extended to the case with multiple Byzantine robots. We assume the Byzantine robot can arbitrarily compromise any type of messages sent from it at any time. Then our fuzzing strategy has three steps to discover the potential risks. (1) *Requirement specification*: we formulate the requirements for the normal operation of an MRS with Signal Temporal Logic (STL). This includes the general requirements (safety, mechanism, performance) as well as task-specific requirements. (2) *Data-flow analysis*: we dynamically generate the data-flow diagram at the level of node operations by simulating the workload. Through analyzing this diagram, we extract the parameters and communication messages controllable by the Byzantine robot, which form the fuzzing input space. (3) *Requirement-driven fuzzing*: we mutate the messages from the identified input space and check whether the requirements are violated. Different mutation strategies (dropping, content modification, etc.) are considered for different types of messages.

We build an MRS workload suite, which incorporates standard implementations of common MRS workloads and coordination schemes from the past literature [12, 16, 17, 31, 38, 42, 45, 53, 55, 69, 71, 78, 80, 90, 94, 99, 100, 105] and ROS platform. Using our requirement-driven fuzzing methodology, we uncover three new forms of Byzantine attacks with five attack strategies in these existing workloads. (1) *Task assignment control* attack: the Byzantine robot can manipulate the messages of location, robot status or task bidding to compromise the task assignment process. (2) *Map merging poisoning* attack: the Byzantine robot can decrease the quality of generated map by falsifying the explored map messages. (3) *Task forwarding manipulation* attack: the Byzantine robot can tamper with the transmitted task information to mislead other robots to perform wrong jobs.

We perform extensive experiments using the Gazebo simulator [49] to validate the effectiveness of these attacks. They can significantly decrease the performance of the entire system, or even cause system crash. Moreover, we deploy these workloads in a real-world environment and MRS consisting of three TurtleBot3 UGVs [8], and successfully achieve the discovered attacks. In summary, we make the following contributions:

- We design a novel requirement-driven fuzzing methodology to identify Byzantine threats and the corresponding strategies for distributed MRS workloads.
- We introduce and opensource a first-of-its-kind MRS workload suite, consisting of different standard workloads and coordination schemes. They can be deployed in simulators as well as physical robots for performance evaluation, security assessment and other purposes as well.
- We discover three new forms of Byzantine attacks in existing common MRS workloads.
- We perform evaluations in both simulation and real-world environments, and the real-world experiments confirm that the consequences observed in simulated environments are realistic.

The rest of this paper is organized as follows. Section 2 introduces the background of ROS, MRS workloads and our threat model. Section 3 presents our novel fuzzing method for Byzantine threat identification. We describe our MRS workload suite in Section 4, followed by the discovered attacks in Section 5. Sections 6 and 7 demonstrate our evaluations in a simulator and physical environment, respectively. We discuss possible countermeasures and related works in Section 8, and conclude in Section 9.

2 BACKGROUND AND THREAT MODEL

2.1 Robot Operating System

ROS is the most popular robotic platform for robot research and development. It has been widely adopted in the research community, as well as industry, e.g., Dji Matrice 200 drone [1] and PR2 humanoid [7]. This platform provides full-stack open-source services to ease the development of robotic workloads. First, it offers a set of core libraries as the low-level middleware. These libraries are deployed between robot apps and hardware to support runtime execution, such as abstracting hardware, passing messages and managing devices. Second, it provides thousands of high-level packages for various functions [6]. Developers can integrate these packages to build a robot workload.

In this paper, we focus on the Multi-Robot workloads implemented from the ROS platform. Our methodology and tool can be extended to other robotic platforms and implementations as well.

2.2 Workflow of Robot Tasks

The workflow of a task running on a robot can be represented as a Directed Acyclic Graph of actions (actionDAG), where each node represents a certain action, and edges represent the dependencies of the actions in this task [101]. Figure 1 shows the structure of a standard robot task. It consists of three fundamental stages: (1) *Perception*: the robot extracts estimated states of the environment and the device from raw sensor data. It uses the `Localization` node to determine the device position, and `CostmapGen` node to model the device's surroundings. (2) *Planning*: the robot determines the long-range actions. It uses the `Path Planning` node to find the shortest path, and `Exploration` node to search for accessible regions. (3) *Control*: the robot processes the execution actions and

[1]ROS is used as a communication wrapper in ABB manipulators.

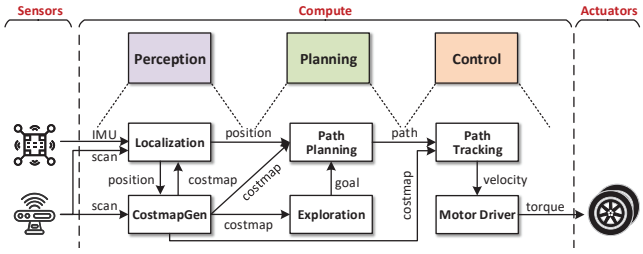


Figure 1: Application pipeline for a typical robot task.

forwards these motions to the actuators. It uses `Path Tracking` to produce velocity commands following the planned path, and `Motor Driver` to transfer the velocity command to specific actuators.

2.3 Multi-Robot Systems

In an MRS, a number of robots with the same type (homogeneous) or different types (heterogeneous) work together to complete one workload. Such collaboration mode can bring two benefits over single-robot systems. First, since a robot is mainly designed with one specific functionality, the incorporation of multiple robots can address complex tasks that can never be achieved by one robot. Second, the computation capability and power capacity of a robot are limited. Hence, an MRS can significantly increase the working efficiency and operation duration. Due to these advantages, MRSs have been practically adopted in many scenarios.

2.3.1 Multi-Robot Workloads. We present a categorization of common MRS workloads in our daily life.

Navigation. This type of workloads is a fundamental capability of mobile robot systems, widely applied in house cleaning [67], warehouse delivery [29, 51], surveillance[70] and patrolling[32, 66]. It can be abstracted as determining the robot's own position and moving towards a predefined destination. To achieve this goal, in Figure 1, the `CostmapGen` node creates a costmap of the robot's surroundings, and `Localization` estimates the robot's position. Based on such information, `Path Planning` generates an optimal collision-free path to the destination. `Path Tracking` follows this path and outputs the best action. The final velocity command is sent to the actuators. During moving, each robot needs to frequently interact with the environment, recognize surrounding objects or other robots, and possibly recalculate the path.

Exploration. In this type of workloads, the robots are expected to spread in an unknown area to achieve the maximal coverage, and collect as much information as possible. Typical examples of exploration include map building [37] and rescue [74]. Generally, the goal of each robot is to keep reaching new locations that are never touched by other robots. In Figure 1, the `localization` node executes the Simultaneous Localization and Mapping (SLAM) algorithm to infer the robot's position in absence of a map. Then, `Exploration` selects an unexplored position as the destination and sends the goal to `Path Planning`. By repeating this process of costmap update and exploration, the map of the environment will be expanded, until the entire area has been explored.

Formation. A swarm system is a special MRS which consists of a large number of simple robots with local sensing and communication capabilities. These robots interact with each other to produce complex swarm behaviors. Formation is one typical workload for swarm systems, where the robots try to maintain certain physical arrangements or patterns. There are two typical swarm behaviors in a formation task. The first type is aggregation/dispersion. Aggregation refers to the behavior where robots from different locations gather together in one spot. In contrast, dispersion is to move the robots from one spot to fully cover a certain area. The second type of swarm behaviors is pattern formation: robots need to adjust their locations to create a global shape, varying from simple geometry [11] to more complex shapes, e.g., alphabetical letters [39].

Antagonism. An MRS can also be implemented for the purpose of antagonism, e.g., robotic soccer and robot combat. For example, in a soccer game, the robots in one team are instructed to compete with the opponent team to score the goals and defending the opponent robots. Such MRSs are usually implemented in a closed monitored environment and less prone to attacks. So we do not discuss the security vulnerabilities of these workloads in this paper.

2.3.2 Communications in Multi-Robot Systems. Since robots in an MRS collaborate on the workload, they need to frequently exchange messages. In general, robots share information by either broadcasting or one-to-one communication via wireless networks. To efficiently control the entire system, there are typically two coordination schemes in modern MRSs.

Centralized scheme [69, 71]. In this design, a centralized entity is introduced to coordinate all the robots in an MRS. This entity can be a local edge gateway, a remote cloud server, or even a powerful robot inside the system. It collects information from the robots, makes decisions, and sends the instructions to different robots.

Decentralized scheme [16, 31]. This design eliminates the centralized entity, so each robot can communicate with others directly. Every robot retrieves information from the environment and other robots and makes decisions by itself. Robots exchange or broadcast messages frequently to make the entire system reach consensus. This decentralized scheme exhibits a higher level of autonomy.

It is worth noting each workload can be implemented by either the centralized or decentralized scheme. They may have different efficiency for different workloads. In Section 4, we will review and analyze the real-world MRS implementations for different workloads and coordination schemes.

2.4 Threat Model

We consider an MRS where a number of robots collaboratively work on one workload. We focus on the Byzantine threats in this system. Particularly, we assume one robot is malicious and fully controlled by the adversary, which attempts to compromise the entire MRS. There are several reasons that make this assumption realistic. (1) The ROS middleware lacks basic security mechanisms for the authentication and encryption of the communication between nodes, and thus suffers from many security issues, e.g., plaintext communication, lack of authentication or authorization [19, 25], and denial-of-service vulnerability [28, 102]. A remote adversary can easily leverage these vulnerabilities to break into the robot and control it to perform arbitrary malicious behaviors. (2) A lot of function packages in the ROS platform contain exploitable software vulnerabilities [28, 57]. According to the Robot Vulnerability Database [3, 93], 17 robot vulnerabilities and 834 bugs (e.g., no authentication,

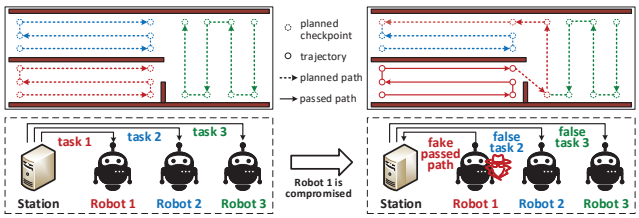


Figure 2: An example of Byzantine threats in a surveillance application.

uninitialized variables, buffer overflow) were discovered in the function packages of 51 robot components, 37 robots and 34 vendors in the ROS platform. Most of them are still unpatched. Exploitation on real-world ROS package vulnerabilities was reported [21], and red teaming strategies on ROS applications were studied in [56]. These software vulnerabilities also enable the adversary to intrude into the robot and take full control of it. (3) The ROS platform is open for everyone to upload and share their function packages [103]. Unfortunately, it does not perform any security check over the submitted code. Hence, an adversary can publish malicious function packages for other users to download. Based on the ROS2 Robotic Systems Threat Model [4]: "third-party components releasing process create additional security threats (third-party component may be compromised during their distribution)".

The Byzantine robot tries to affect the completion of the workload by sending malicious messages to other robots or the centralized controller. It can also refuse completing the tasks assigned to it. This can bring disastrous consequences due to the following two facts. First, robots are closely interconnected with each other following either the centralized or decentralized scheme. The stability and integrity of the entire MRS highly depend on the reliability of the inter-robot communications. Second, there is a severe lack of Byzantine-resilient mechanisms in existing MRS designs and implementations. Developers do not consider Byzantine defenses because it is challenging to have a satisfactory solution due to the variety and complexity of messages exchanged between robots. Deploying such defenses can also decrease the system performance as a ratio of robots are not trusted. These two facts exacerbate the severity of Byzantine threats in MRSs.

The Byzantine robot can also have other means to interfere with the system. For instance, it can alter the environmental states or perform sensor spoofing attacks to indirectly affect the decisions of other robots. It can also conduct physical damages (e.g., path blocking, collision) to compromise the system. Those attack vectors are not considered in our work, as they are less stealthy and could be easily detected by the ground monitoring systems.

Note that we do not consider the case where the benign robots can detect the existence of the Byzantine robot via monitoring the surrounding environments. The reason is that due to the limit of communication and sensing ranges, a robot cannot obtain the global information of the environment or the past behaviors of other robots. The absence of such information makes it hard for a benign robot to identify whether its neighbors are anomalous from their current behaviors and states. How to detect the Byzantine threat from benign robots will be an interesting future work.

An example of Byzantine threats. Figure 2 shows an example of Byzantine threats in a surveillance workload, where three robots perform the navigation task given by the control station in real time. Each task consists of a sequence of planned checkpoints that the robot needs to follow. We assume that robot 1 becomes the Byzantine adversary and sends falsified path data to the control station. This can cause the station to mistakenly believe some untouched checkpoints have been passed, and then replan new tasks for benign robots which will exclude these checkpoints. Then these spots will never be navigated, and the workload cannot be completed.

3 A METHODOLOGY TO CHARACTERIZE BYZANTINE RISKS IN MRS

In this section, we present our novel methodology to automatically and comprehensively identify possible Byzantine threats in MRSs.

3.1 Overview

Given an MRS workload, our goals are to (1) identify whether its implementation is Byzantine-resilient, *i.e.*, functioning well when some robot is malicious; (2) if not, produce the optimal attack strategies to compromise the system.

As described in Section 2.4, a Byzantine robot can tamper with arbitrary messages in an arbitrary way to interfere with the entire MRS. To thoroughly identify potential Byzantine risks in a workload, we propose to use the fuzzing strategy [83]. However, there are several design challenges to apply fuzzing to our scenario. First, traditional fuzzing bug-oracles are designed to mainly detect system crashes, rather than abnormal system states in our case (e.g., robots are stuck in the idle status permanently). To address this issue, we introduce a bug oracle which is aware of MRS states via Signal Temporal Logic (STL) formulas with robustness semantics [26, 60]. The STL formulas describe the requirements the MRS should satisfy during its operation. Our method constantly monitors if the formulas are violated when fuzzing the target workload. Second, traditional fuzzing techniques cannot generate mutated communication messages due to the large input space of the MRS workload, with numerous types and formats of messages. To handle this limitation, we propose to leverage dynamic data-flow analysis to extract the critical inter-robot communication messages, which can significantly reduce the input space for fuzzing.

Figure 3 shows the workflow of our methodology. We adopt the STL to specify the requirements that the MRS should follow (Section 3.2). With these requirements, our method performs dynamic data-flow analysis (Section 3.3) and requirement-driven fuzzing (Section 3.4) to extensively evaluate the workload and output the possible attacks. Below we introduce the mechanism of each step in detail.

3.2 Requirement Specifications

During the execution, an MRS should satisfy various requirements to guarantee safety and task completion. These requirements can be divided into general ones (e.g., safety) and task-specific ones (e.g., navigation coverage, map accuracy). We adopt STL to formulate these two kinds of requirements for attack identification.

We briefly describe some basic concepts of STL, while more details can be referred to [60]. Let $\Box$, $\Diamond$, and U be the temporal operators "always", "eventually", and "until", respectively. Given a

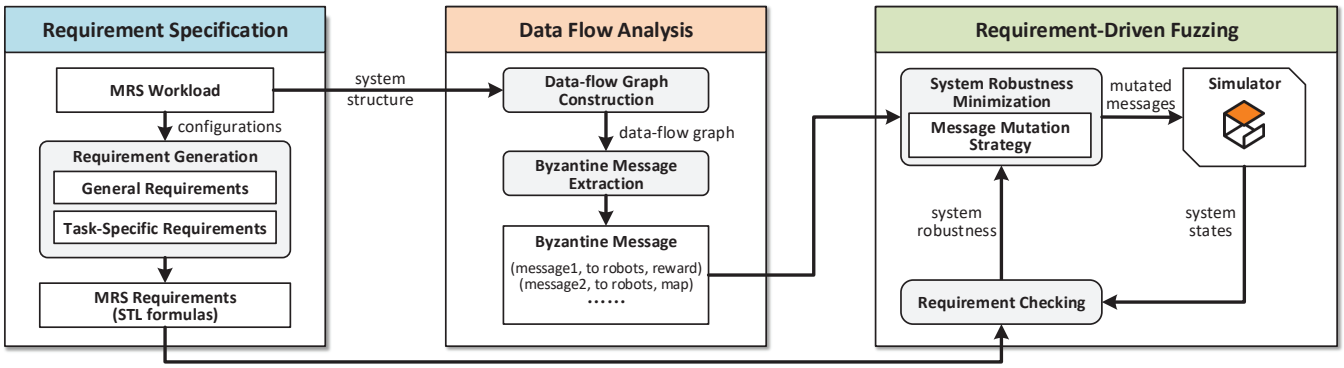


Figure 3: Workflow of the proposed methodology.

variable set X, its value at time t is denoted as $X(t)$. Then a signal w over X is a time sequence $(t_0, X(t_0)), \ldots, (t_n, X(t_n))$, where $t_0 = 0$ and $t_i < t_{i+1}$. For our case, the variable set of a robot is the position x, velocity v, acceleration a, and the set of detected obstacles O. The syntax of an STL formula φ over X can be defined as: $\varphi := \top \mid \mu \equiv f(X(t)) > 0 \mid \neg\varphi \mid \varphi_1 \vee \varphi_2 \mid \varphi_1 U_{[a,b]} \varphi_2$, where $\top$ means True, $\neg$ is the negation operator, and $\vee$ is the disjunction operator. $\mu \equiv f(X(t)) > 0$ is called an atomic STL formula, where $f : X \rightarrow R$ is a real-valued function related to a property, e.g., the distance function (e.g., the minimal distance between a robot and its surroundings) for safety consideration. We use these notations to describe some representative requirements for an MRS.

3.2.1 General Requirements. First, we consider some general requirements which are suitable for various Multi-Robot workloads.
Safety. The most important requirement is collision avoidance. At any time instance, a robot should keep a safe distance d_s from obstacles, including other robots in the same environment. Let $d(t)$ be the minimal distance between the robot and obstacles at time t, then the STL formula for safety is $\varphi_1 \equiv \square(d(t) \geq d_s)$.
Mechanics. Due to the physical limitations, the speed and acceleration of a robot cannot exceed the boundaries. Suppose the maximal speed and acceleration of a robot are $v_{\max}$ and $a_{\max}$, respectively, then the STL formula is $\varphi_2 \equiv \square(0 \leq v(t) \leq v_{\max} \ \& \ |a(t)| \leq a_{\max})$.
Energy saving. Due to the limited battery capacity, a robot is expected to reach its destination x_g before power exhaustion. Hence, suppose the battery power at time t is $E(t)$, the STL formula can be written as $\varphi_3 \equiv (\Diamond \|x(t) - x_g\|_2 \leq \epsilon) \ \& \ ((E(t) > 0) \ U \ (\|x(t) - x_g\|_2 \leq \epsilon))$, where ϵ is a predefined tolerance for task completion.
Execution time. For an arbitrary workload, each robot is expected to complete the assigned task as soon as possible within a given time budget T. Hence, the STL formula for timeliness can be written as $\varphi_4 \equiv \Diamond_{[0,T]} \|x(t) - x_g\|_2 \leq \epsilon$.

3.2.2 Task-specific Requirements. In addition to the above general requirements, there are also some specific requirements for different workloads. We describe three examples as below.
Navigation requirement. In a navigation workload, the robots in the system are required to complete a set of navigation tasks, such as going through a set of waypoints or regions. However, due to the unexpected and dynamic changes in the environment, not every task can be completed safely (e.g., some spots are occupied by accident). Hence, each system is given some tolerances for task completion. Given a set of navigation tasks $\{x_g^1, x_g^2, \ldots, x_g^K\}$ and the minimal task completion rate ω, the STL formula for the navigation requirement can be written as $\varphi_5 \equiv \Diamond(\wedge_{i \in \{i_1,\ldots,i_k\}} \|x(t) - x_g^i\|_2 \leq \epsilon)$ $U \ k/K \geq \omega$, where $\wedge$ denotes the conjunction operator.
Exploration requirement. For exploration, robots are instructed to collect as much information as possible with a shorter moving distance. To evaluate the completion of an exploration task, the MRS is required to cover $w \in [0, 1]$ of the ground truth map within a given time duration T. Hence, the STL formula for this requirement can be written as $\varphi_6 \equiv \Diamond_{[0,T]} M(t)/M \geq w$, where $M(t)$ is the area of the explored map at t.
Formation requirement. In a formation workload, robots in the system should coordinate with each other to form a predefined formation. The system first determines a set of positions for these robots to occupy. Hence, we can illustrate the following requirement: given a formation with a set of vertexes $\{p_1, p_2, \ldots, p_m\}$, the system should assign each vertex to one robot exclusively, and then each robot moves to its corresponding destination within a given time budget T. Let $\sigma(r_1, r_2, \ldots, r_m)$ be a permutation of the robots $\{r_1, r_2, \ldots, r_m\}$ and $\sigma(i)$ is required to move to p_i. The STL can be described as $\varphi_7 \equiv \Diamond_{[0,T]} \wedge_{i \in \{1,2,\ldots,m\}} \|x_{\sigma(i)}(t) - p_i\|_2 \leq \epsilon$.

3.3 Data-flow Analysis

After deriving the requirements, we need to identify the input space for fuzzing. According to our threat model, the Byzantine robot can send arbitrary malicious messages to other robots or the centralized controller. We propose to use data-flow analysis [35] to identify the critical messages that could possibly violate the requirements. This can be achieved with the following two steps automatically:

3.3.1 Data-flow graph construction. In ROS applications, the task in each robot consists of multiple computation nodes that perform different functions (Figure 1). The communication between those nodes (either inside one robot or across different robots) is implemented in a publisher-subscriber mode. Message topics are many-to-many named buses which describe the states of robots or environment. A node can subscribe to a topic if it wants to receive relevant data, or publish data to a topic. Therefore, given an MRS workload, we first construct the corresponding data-flow graph [15] to include all nodes and the types of messages flowing among them.

3.3.2 Byzantine message extraction. The next step is to identify the critical messages that can be manipulated by the Byzantine robot. We label each inter-node communication based on its source node and highlight the nodes which are controlled by the Byzantine robot. Then messages sent by these nodes can be falsified. We exclude the messages transmitted inside one robot and only consider the inter-robot communication for fuzzing.

We inspect all the Multi-Robot packages from the ROS platform [2] and discover six common types of messages as the targets of the Byzantine attacks, elaborated as below:

M1: Odometry. This type of messages typically stores the estimation of the robot's instant velocity and position in the environment. This message is important for robots to adjust their motion to avoid collision and complete motion tasks. For instance, in a navigation scenario, the centralized controller needs to collect robots' exact positions from their odometry messages and calculate the corresponding paths for them.

M2: Robot status. Robots in an MRS need to frequently broadcast their current statuses (e.g., "active", "idle") for the system to properly allocate the tasks in time. Some MRS workloads may introduce more statuses to better coordinate the robots. For instance, in a coordinated exploration task, robots can stay at the "verification" status when they are moving in the explored regions. The map information sent at this status may be used to increase the map accuracy. The exploration tasks are preferably assigned to the robots at the "idle" status, and then to those at the "verification" status. This can maximize the utilization of all the robots in the MRS.

M3: Map. Most robot workloads need the map information during the execution, whether it is known (navigation) or unknown (exploration). In ROS, a map message is generally represented as the occupancy status of each cell in the target region. A typical map message in the *nav_msgs* package contains a variable *MapMetaData*, which includes the information of the width and height of the map in terms of the number of cells, the resolution of each cell and the origin of the map, and a vector variable *data*, which describes the occupancy probability of each cell in the map. The accuracy of the exchanged map information can heavily affect the allocation and execution of subsequent tasks.

M4: Reward. In some MRS workloads, each robot calculates the reward of performing one specific task and broadcasts the value to the entire system. A new task is thus allocated to the robot with the highest reward. As a result, the reward values can significantly affect the allocation decisions, then the efficiency and completion of the entire workload.

M5: Task/Goal. This type of messages contains the current task to be completed. In a centralized system, these messages are sent from the centralized controller to each robot for task assignment. In a decentralized system, robots broadcast those messages until the task assignee receives the task information. A Byzantine robot participating in the propagation of such messages can tamper with the tasks or goals and mislead the assignee to perform wrong jobs.

M6: Path. This type of messages contains the trajectory of a robot from the current location to the destination. In some workloads (e.g., decentralized exploration [31]), each robot has the capability and responsibility of calculating its own path based on the given goal. In the applications where individual robots do not have the computation capability to generate paths independently, the paths are calculated by the centralized controller and sent to the robots.

3.4 Requirement-driven Fuzzing

Our next stage is to perform requirement-driven fuzzing over the MRS workload. We mutate all possible critical messages identified in Section 3.3.2, and monitor if they lead to any violations of the requirements specified in Section 3.2.

3.4.1 Overview. Algorithm 1 details our requirement-driven fuzzing procedure. For each requirement in terms of the STL formula, our method repeatedly conducts the following steps for each message: (1) identifying the message data type and performing mutations according to the mutation strategy designed for the data type (Line 5); (2) replacing the original message with the mutated one, executing the workload in the simulator, and recording the state sequence of the system execution (Line 6). Particularly, the adversary has the right not to perform the assigned tasks, which is also considered during the simulation; (3) computing the robustness of the recorded state sequence (Line 7); (4) if a violation is detected, storing the simulation configurations and continuing with the next message (Lines 8 - 10); otherwise, if the robustness of the mutated message is smaller, replacing the old message with the mutated one and updating the corresponding robustness (Lines 11 - 13). After all the messages in the input space are fuzzed, we summarize the mutations that can lead to requirement violations. A new round of fuzzing will start if testing time is allowed.

3.4.2 Message Mutation Strategy. For a given requirement with the STL formula φ, the corresponding robustness degree ν of the system can be calculated at the end of the workload execution. The mutation strategy aims to minimize the system robustness degree by varying the messages sent by the Byzantine robot, and ideally result in a system requirement violation. For different types of messages, we provide a couple of possible mutation strategies.

The first strategy is to drop the messages. The Byzantine robot can pretend to "forget" sending critical messages to the corresponding receiver, or broadcasting them to the entire system. This is one kind of Denial-of-Service attack in MRS.

The second strategy is to randomly change the values contained in the messages within the legal range. For the numerical type (e.g., odometry, reward), the Byzantine robot can change the message to a random value within the legal range. For the categorical type (e.g., robot status), the Byzantine robot can change it to a different category. Similarly, for the message type of task/goal, the adversary can alter the content to a random task pre-defined in the workload.

The third strategy is specifically designed for the map message. A map message is a 2-dimensional metric, which provides a much larger fuzzing input space than other types. So a fully random mutation strategy is inefficient. Instead, we consider two new mutation methods: (1) the Byzantine robot replaces the target map with an empty one or a fully-occupied one, to check the system's Byzantine-resilience in extreme cases; (2) the Byzantine robot randomly picks a region with a distance of l from its current position, where l is a pre-defined hyper-parameter based on the map size.

Algorithm 1: Requirement-driven Fuzzing

Input: A simulator SIM, the set of message types for mutation M, fuzzing space $\Pi_{msg \in M} Input(msg)$, a set of STL formulas Φ, a fuzzing time-limit τ
Output: V: requirement robustness related to each message; M_v: the corresponding messages causing violations.

1 **for** *each requirement* $\varphi \in \Phi$ **do**
2 Initialize the values of the input messages $M = \{msg_0, msg_1, \ldots, msg_n\}$, $V(msg_i) = +\infty$, $M_v = \emptyset$;
3 **while** *total_time* $< \tau$ **do**
4 **for** *each msg in* M **do**
5 $mutated\_msg$ = MUTATE(msg, $Input(msg)$);
6 $state\_seq$ = SIM.SIMULATE($mutated\_msg$);
7 $v = req\_checker(\varphi, state\_seq, 0)$; /* Compute the robustness of φ with respect to $state\_seq$ */
8 **if** $v < 0$ **then**
9 $M_v = M_v \cup \{msg\}$;
10 $M = M \setminus \{msg\}$;
11 **else if** $v < V(msg)$ **then**
12 $msg \leftarrow mutated\_msg$;
13 $V(msg) = v$;
14 **end**
15 **end**
16 **if** $M = \emptyset$ **then**
17 Generate a new set of M randomly, initialize V, and repeat Lines 4 - 15.
18 **end**
19 **end**
20 **end**

3.4.3 Requirement Checking. Give an STL formulas φ and a sequence of system states $state\_seq$, the requirement checking process $req\_checker(\varphi, state\_seq, t)$ returns the robustness degree of φ over w at time instant t, which describes how far w is from satisfying or violating φ at t [60]. The robustness can be computed as follows. First, the robustness of the atomic STL $\mu \equiv f(X(t)) > 0$ with respect to $w(X) = X(0), X(1), \ldots, X(n)$ at time instant t can be computed as $\rho(\mu, w, t) = f(w(X))[t] = f(X(t))$. Based on the syntax of STL, we have $\rho(\neg\varphi, w, t) - \rho(\varphi, w, t)$, $\rho(\varphi_1 \vee \varphi_2, w, t) = \max\{\rho(\varphi_1, w, t), \rho(\varphi_2, w, t)\}$, $\rho(\varphi_1 U_{[a,b]} \varphi_2, w, t) = \max_{t' \in [t+a, t+b]} \min\{\rho(\varphi_2, w, t'), \min_{t'' \in [t,t']} \rho(\varphi_1, w, t'')\}$. Hence, the robustness of an arbitrary STL formula can be computed by applying the above the computation recursively. More details about the robustness degree functions can be found in [60].

Based on the definition of robustness degree, $\rho(\varphi, w, t) < 0$ means that the signal violates φ at t. Hence, for each STL-based requirement $\varphi \in \Phi$ derived from the requirement specification step (Section 3.2), the system robustness degree ρ on the selected requirement is calculated with the system state sequence obtained from the simulator. If we detect an execution whose ρ is smaller than 0, i.e., a requirement violation, we store the system configurations and mutated message; otherwise, we guide the mutation to the direction that decreases the robustness. If no requirement violation is detected at the end of fuzzing, the lowest system robustness together with the corresponding mutated message and system configurations are returned as output.

4 AN MRS WORKLOAD SUITE

To extensively evaluate the effectiveness of our method and understand the Byzantine vulnerabilities in MRSs, we select five typical MRS workloads as our testbed, which cover a variety of coordination schemes and application domains discussed in Section 2.3. These workloads are identified from prior literature and existing packages in the ROS platform. Each workload can support an arbitrary number of robots running end-to-end tasks including perception, planning and motion control. They are ready to be deployed to a ROS simulator (e.g., Gazebo [49]) or physical robot devices.

To our best knowledge, this is the first-of-its-kind workload suite for Multi-Robot applications based on the ROS platform. We open-source this MRS workload suite[2], and expect it can contribute to the robotics community for other purposes as well (e.g., performance evaluation and characterization, MRS hardware and software co-design). We give detailed descriptions of these workloads in this section, followed by their security assessment in the next section.

4.1 Workload Descriptions

W1: Centralized Navigation [45, 55, 71]. The goal of this workload is to efficiently complete a surveillance task by coordinating multiple robots to navigate to different target locations. Figure 4 shows the workflow of this workload. A centralized controller server (e.g., Ground Control Station (GCS)) is introduced to manage the communication. Specifically, given a sequence of goal positions, the controller server generates the corresponding collision-free paths and sends the path information to different robots. Then each robot follows the designated path to reach its destination. During moving, a robot needs to avoid collisions with obstacles and other robots. Meanwhile, it also needs to frequently send two types of messages to the controller server: (1) odometry messages containing the robot's current location; (2) status messages denoting whether the robot has arrived at the destination, or in the "idle" status waiting for further commands. The controller server takes such information to update the paths and assign a new task to the robot which is in the "idle" status and closest to the target location.

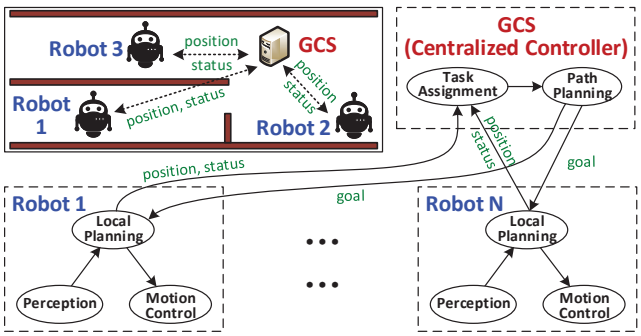


Figure 4: Workflow of W1

[2] https://github.com/GeleiDeng/RAID_2021_MRS_Fuzzing

W2: Centralized Exploration [38, 69]. In this scenario, an MRS is deployed to build a map of an unknown area. This workload is implemented in a centralized manner, where a GCS is used to manage the robots for exploration. Figure 5 shows the workload of this implementation. During the task, the GCS runs the exploration stack and identifies the frontiers of potential areas to be explored. It then assigns the frontiers to available robots by calculating the exploration cost and utility [16], and generates the paths from the corresponding robots' current positions to the frontiers. Each robot frequently exchanges three types of information with the GCS: (1) odometry messages denoting the robot's current position, (2) status messages denoting whether the robot is in the exploration or idle status, and (3) map messages containing the exploration results. The GCS merges the maps from different robots. It assigns new area for the robot which has finished its current exploration. The above process is repeated until the entire map is established.

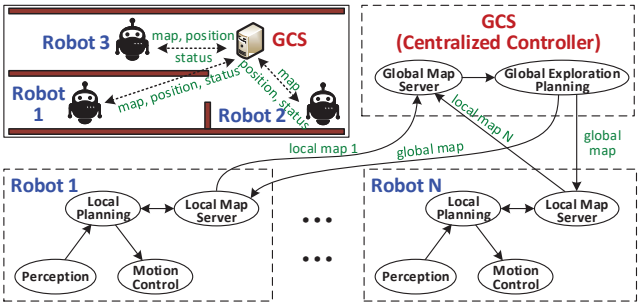


Figure 5: Workflow of W2

W3: Decentralized Exploration with Bidding [12, 17, 31, 42, 90]. This workload achieves the same function as **W2**, but in a decentralized manner. Robots talk to each other and adopt the bidding algorithm to reach agreement for frontier assignment. Figure 6 shows the workflow. During exploration, robots keep exchanging two types of messages: (1) a map message containing the local map maintained by the robot, and (2) a bidding message containing the robot's gains of exploring different frontiers. When a robot receives messages from other robots, it first merges their newly explored maps to its local one. Then it compares its gain with other robots' and selects the frontier where it has the highest gain. It declares to the system that it will explore this frontier, and then starts the task. These steps are repeated until the entire area is explored. At last, robots merge their local maps again to generate the final map.

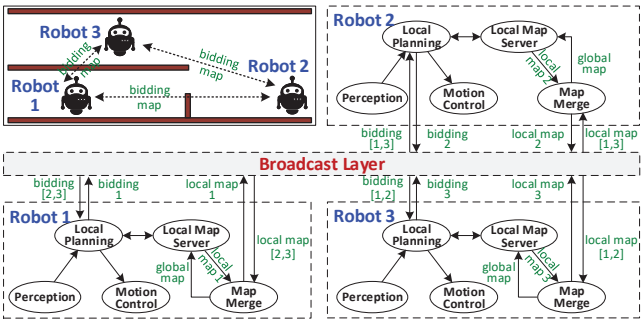


Figure 6: Workflow of W3

W4: Decentralized Exploration with Group Merging [16, 78, 80, 100]. This workload is similar to **W3**. The difference is that robots are highly distributed without knowing the relative position of each other. A robot cannot broadcast to all the robots: It can only talk to the robots which move into its sensing range. The workflow is shown in Figure 7. In this scenario, the group merging algorithm is adopted to achieve task allocation. Specifically, when two robots meet, they exchange the map information and verify whether their maps can be merged together. Robots with confirmed joint map regions form one group and share the explored maps via the wireless network until they move out of the communication range. At the end of the task, all the robots will share the same map information for the entire area.

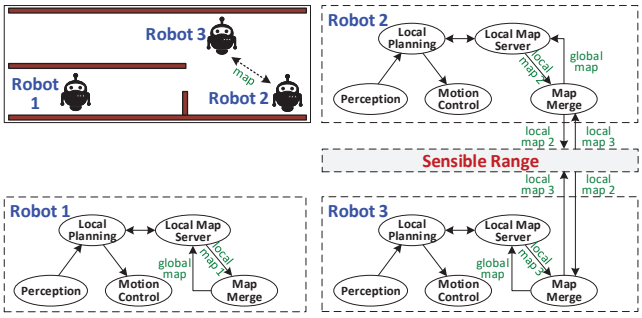


Figure 7: Workflow of W4

W5: Swarm formation [53, 94, 99, 105]. This workload is designed for a swarm system. It controls the swarm robots to achieve aggregation, dispersion and line formation. It is applied to InchBot [44], a novel swarm microrobotic platform that contains highly modular two-wheel mini robots with wireless sensing and communication capability. Figure 8 shows the workflow. Each robot obtains the relative positions of other robots within the communication range through the wireless sensor network. They achieve the aggregation or dispersion behaviors by maintaining a pre-defined average distance with others within the sensing range. In the line formation task, each robot moves to an ideal location to form a line with its adjacent robots, while maintaining a pre-defined distance in a similar way as dispersion. The formation commands are given by a controller outside the system, and then robots broadcast the commands to others within the sensing range.

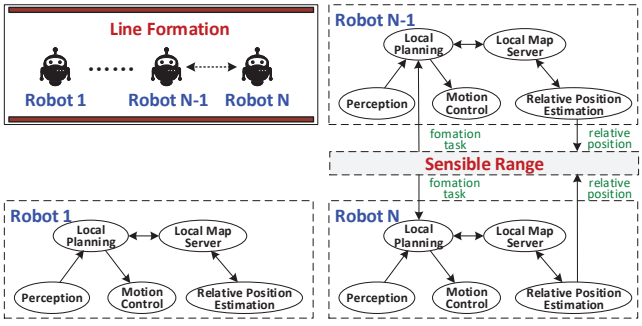


Figure 8: Workflow of W5

4.2 Metrics

Our workload suite also provides a set of metrics to measure the performance and efficiency of Multi-Robot workloads. They can be classified into the following two categories.

General metric. We measure the *execution time*, which is the total time spent in completing the workload. A severe attack can significantly increase the execution time and cause Denial-of-Service damage. An extreme case is that the execution time can be infinity as the workload can be never completed. This is a workload-independent metric and can be used to describe different scenarios.

Task-specific metrics. In addition to the above general metric, different workloads can also have diverse measurements for the performance and efficiency. (1) For a navigation workload, we measure the *navigation rate*, i.e., the percentage of the destination spots reached by the robots. We expect the system to cover as many desired destinations as possible. (2) For an exploration workload, we adopt the *map quality* to quantify the performance of the execution [98]. It directly reflects how well a map can be constructed by the robots. (3) For a formation workload, we introduce *formation similarity*, which denotes the similarity between the planned and actual patterns. Specifically, we adopt an Euclidean distance-based similarity measure between the two formation spaces:

$$s = e^{-\frac{\sum_{i=1}^{m} \|x_i - p_i\|_2}{m}}$$

where p_i is the ideal position of the i-th robot in the formation, x_i is its actual position, and m is the number of robots in the system.

5 MULTI-ROBOT SYSTEM RISK ANALYSIS

We leverage the proposed risk identification methodology in Section 3 to assess the five workloads described in Section 4. The five workloads are implemented and fuzzed in the simulation environment based on Algorithm 1. System requirement violations together with the mutated messages are recorded, and the root causes of those violations are manually inspected. We finally identify seven Byzantine risks in these implementations and characterize them into three classes of attacks, as described below.

5.1 Attack 1: Task Assignment Control

Multi-Robot Systems efficiently achieve the ultimate goals by breaking down the high-level task into sub-tasks and appropriately assigning them to qualified robots. This task assignment process involves a series of calculations to maximize the overall system gain based on the task, environment and current status of each robot. It can be performed in a centralized controller or distributed to individual robots depending on the coordination scheme.

If the Byzantine robot can manipulate the messages related to task assignment, it can compromise the assignment decisions and affect other robots. We identify two strategies to realize this attack, targeting different schemes and messages.

5.1.1 Fake location or status information in centralized systems. Typical centralized systems assign tasks to the most appropriate robots by considering their positions and statuses. For instance, in the workloads **W1** and **W2**, tasks are assigned to available robots closest to the target positions. The Byzantine robot can send fake location and status information to the controller, causing it to make wrong assignments. Specifically, in the navigation workload **W1**, the Byzantine robot can lie to the GCS that it is the closest to the target positions. Then it can intercept all the navigation tasks that are supposed to assigned to other robots. In the centralized exploration workload **W2**, idle robots have higher priority to get assignments. The Byzantine robot can send the *idle* status to the GCS, even it still has uncompleted tasks. Such messages can also mislead the GCS to assign more tasks to the Byzantine robot, while ignoring the correct candidates. More seriously, the malicious robot can occupy these tasks without finishing them. This can significantly decrease the completion degree of the workload.

5.1.2 Fake bidding information in decentralized exploration systems. The task assignment mechanism can be attacked in a decentralized MRS as well. For instance, in the workload **W3**, robots bid for the exploration task by calculating the overall gain based on their current positions and statuses. A Byzantine robot can easily steal tasks from others by broadcasting fake bidding information with extreme high gain values. Then it can just keep these tasks uncompleted to affect the system performance.

5.2 Attack 2: Map Merging Poisoning

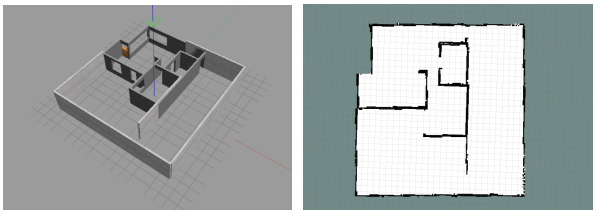
In exploration workloads, the final map is generated by merging local maps from all the robots. Unfortunately, existing map merging packages in the ROS platform (e.g., [43]) adopt the common map merging algorithms [97] without verifying the correctness of the input map data. Hence, a Byzantine robot can keep sending false map information to the map merging function to affect the exploration process. We identify two strategies against the workloads **W2**, **W3** and **W4** based on this attack.

5.2.1 False global map generation. In the workloads **W2** and **W3**, the adopted map merging packages [43, 97] from the ROS platform commonly assemble maps by generating the union of occupancy maps submitted by each robot and then performing noise reduction. A Byzantine robot can compromise this algorithm by sending wrong map data where empty cells are replaced by occupied cells. Then the final merged map gives wrong information for those cells.

5.2.2 Blocking group merging. In the decentralized exploration workload **W4**, robots exchange the map information after the confirmation of joint map areas. Such algorithm can mitigate the random false map generation attack to some extent, as the falsified map might be verified and corrected by other robots who have explored the area. However, the Byzantine robot can still craft a partially fake map to block the grouping process. For instance, it can just introduce false map information to the cells which are significantly far away from its current position, and unlikely to be explored by other robots. In this case, the faked data will not be verified by other robots and are merged directly. As a result, the faked information will block the merging of maps from benign robots.

5.3 Attack 3: Task Forwarding Manipulation

In some systems where task information is transmitted through robots without the centralized controller, a Byzantine robot can manipulate the task information such that the subsequent robots will receive and conduct wrong tasks. We identify one such attack that is applicable to the swarm formation workload **W5**.



(a) 3D view of the room in Gazebo. (b) 2D view of the room in Rviz.

Figure 9: Simulated environment for workloads W1 – W4.

Man-in-the-Middle attack. In a swarm system, robots are assumed to have limited sensing and communication ranges. Hence, a robot cannot send commands to all the robots directly. Therefore, it first issues the formation task to a random robot within the communication range. This robot then forwards the task to other robots it can talk to. The task messages are then propagated via the sensory network and reach every robot in the system. A Byzantine robot inside the propagation chain can change the task messages, causing some robots inside the network to execute false commands.

6 EVALUATION

In this section, we conduct simulation experiments to validate the Byzantine risks identified in Section 5. Evaluations with physical experiments can be found in the next section. More simulation and physical experimental results and video recordings are listed online[3].

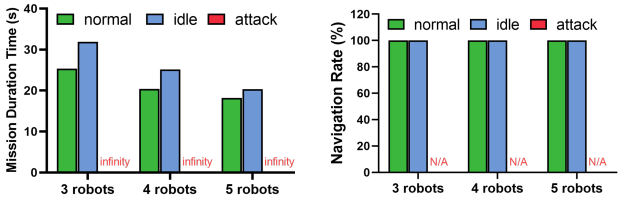
6.1 Experimental Setup

We select the Gazebo simulator [49] with Rviz [72] for the simulation of the MRS with five workloads. Gazebo is the mainstream open-source simulator that can accurately reflect the physical characteristics of robots. We configure Gazebo to simulate a group of robots with rigid body and workload environments. Rviz is a 3D visualization tool for ROS applications. It can display message contents with different ROS topics, and provide APIs for users to publish desired messages to the related topics. We use Rviz to visualize the 2D information from both the simulator and robot applications and publish navigation/exploration goals to the workloads.

We simulate the workloads **W1** - **W4** in a $14\times14m^2$ square room, which is further separated into multiple smaller compartments (Figure 9). We implement a homogeneous MRS with the TurtleBot3 robots [8]. The number of robots varies from 3 to 5 for each workload. Each robot is equipped with a 2D Lidar sensor covering a maximum sensing range of 10 meters to detect the surroundings. For the formation workload **W5**, we simulate a system with 10 to 20 InchBots [44] on an open surface.

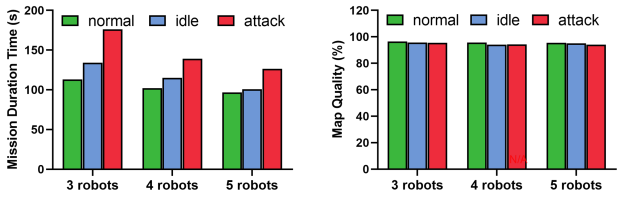
We consider two baselines for comparisons with our attack. (1) *Normal*: all the robots in the MRS are benign and follow the received instructions to complete the tasks. (2) *Idle*: there exists a Byzantine robot in the MRS. It stays idle without requiring any tasks or sending messages. This represents the simplest Byzantine attack which can degrade the system performance to some extent. For each workload in each case, we assume the GCS or the benign robots have the ground truth of the completion status of the tasks. So they can determine the time to stop the tasks.

[3]https://geleideng.github.io/RAID_2021_MRS_Byzantine/



(a) Mission completion time. (b) Navigation rate.

Figure 10: Task assignment control attack against W1.



(a) Mission duration time. (b) Map quality.

Figure 11: Task assignment control attack against W2.

All the simulations are conducted on a Unbuntu 16.04 laptop equipped with an Intel i7-9750H CPU and 32GB RAM. We adopt the ROS Kinetic version for all the MRS workloads. Each experiment below is repeated for 10 times and the average result is reported.

6.2 Evaluation Result

6.2.1 Task Assignment Control Attack. To launch this attack against **W1**, **W2** and **W3**, we randomly select one robot as the Byzantine robot and falsify the task assignment messages sent from it.

Figure 10 shows the results for the navigation workload **W1**. For the *idle* situation, the Byzantine robot increases the mission time by 26.1%, 23.0% and 11.5% for an MRS of 3, 4 and 5 robots respectively (Figure 10(a)). Due to the idle robot in the system, the performance of an MRS with r robots will be the same as that of an MRS with only $r-1$ robots. When the Byzantine robot performs our discovered attack, it obtains all the tasks but never completes them. Then the mission completion time is infinity while the navigation rate is zero, resulting in task failures.

Figure 11 shows the results of the centralized exploration workload **W2**. A malicious idle robot can increase the mission time by 18.2%, 12.9% and 4.2% for the three MRSs, respectively. If it performs the task assignment control attack, then the performance degradation will be much larger (55.5%, 36.4% and 30.8%). Different from **W1**, the Byzantine robot cannot cause failures in **W2**. The reason is that the GCS generates and assigns multiple frontiers for exploration simultaneously. Each robot can only require one task at one time. Hence, the Byzantine robot cannot steal all the tasks. The exploration task will be finally completed by the benign robots. The Byzantine robot can affect the optimal assignment process to cause longer delay. We further analyze the effects of the task assignment attacks on the process of map construction. Figure 12 shows the change of map accuracy for **W2** with three robots. The Byzantine robot starts to send malicious messages at 40s. After that, the map accuracy grows at a slower speed than the original scenario, delaying the task completion.

The performance of the bidding-based decentralized exploration workload **W3** is shown in Figure 13. Similarly, in the *idle* situation, the Byzantine robot only increases the mission completion time.

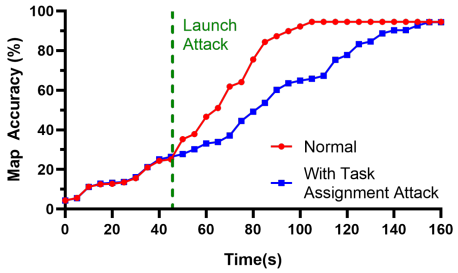
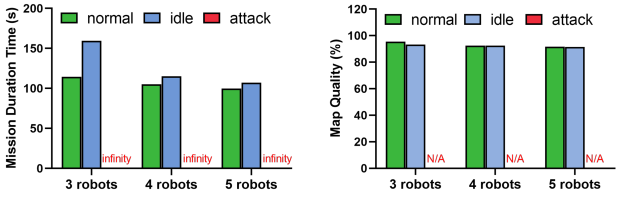
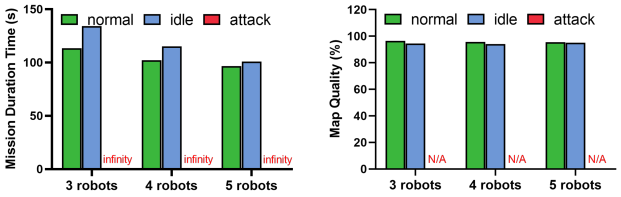


Figure 12: Map accuracy of W2 with 3 robots under the task assignment control attack.



(a) Mission duration time. (b) Map quality.

Figure 13: Task assignment control attack against W3.



(a) Mission duration time. (b) Map quality.

Figure 14: Map merging poisoning attack against W2.

However, different from **W2**, only one frontier can be assigned to a robot via the bidding algorithm each time in **W3**. Hence, with the task assignment control attack, the Byzantine robot can occupy all the frontiers to be explored but does not conduct the jobs, causing the failure of the exploration task.

Summary: We observe that the task assignment control attacks can cause task failures for the entire MRS in **W1** and **W3**. For **W2**, the workload can be completed due to the Byzantine robot's incapability of occupying all jobs. But it can still significantly degrade the performance of the entire system.

6.2.2 Map Merging Poisoning Attack. We implement this type of attacks in the exploration workloads **W2**, **W3** and **W4**, respectively. The Byzantine robot sends falsified map to the GCS or other robots in the *attack* situation.

Figure 14 shows the performance of the centralized exploration workload **W2**. Similarly, compared with the *normal* situation, the *idle* situation only affects the mission duration time. However, in the *attack* situation, the global map generated at the GCS is poisoned by the falsified map from the Byzantine robot. Therefore, GCS fails to generate correct paths for the robots to follow. This will cause an immediate system failure. Figure 15 illustrates the map accuracy during the workload execution. Without an attack, the map accuracy grows gradually to saturation. When the attack occurs at 40s, the correct map generated previously is poisoned, so the accuracy of the merged map will immediately drop to close to zero and will never arise anymore.

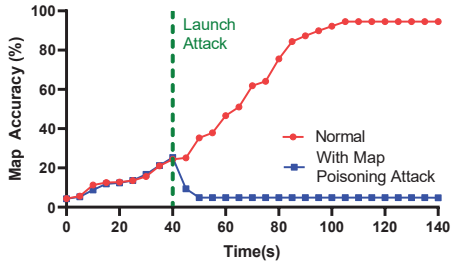
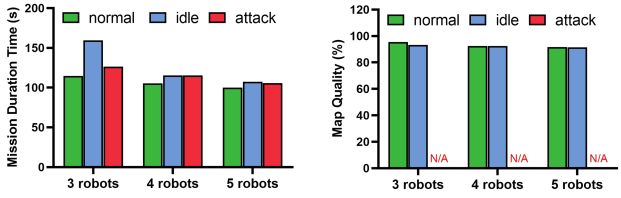
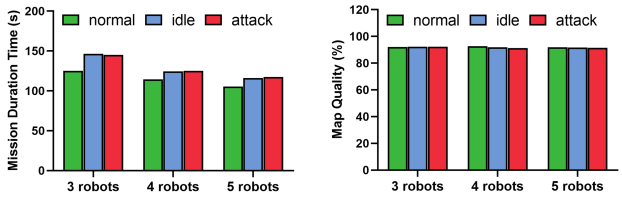


Figure 15: Map accuracy of W2 with 3 robots under the map merging poisoning attack.



(a) Mission duration time. (b) Map quality.

Figure 16: Map merging poisoning attack against W3.



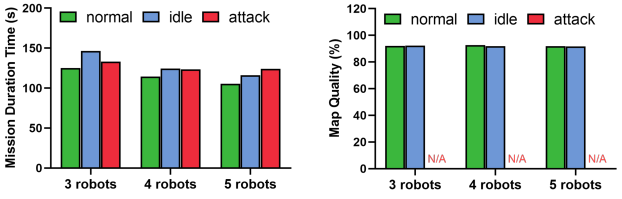
(a) Mission duration time. (b) Map Quality.

Figure 17: False map generation attack against W4.

Figure 16 shows the performance of the bidding-based decentralized exploration workload **W3**. The *idle* situation is similar with the previous workloads and attacks. Under the *attack* situation, each robot can function well even with the existence of a Byzantine robot as the exploration process relies on the local map which is correct for the benign robots. So the total mission completion time is not significantly affected. However, after the exploration is completed, the maps from these robots cannot be merged together due to the poisoned map information from the Byzantine robot. Hence, the workload can be treated as a failure without any maps produced.

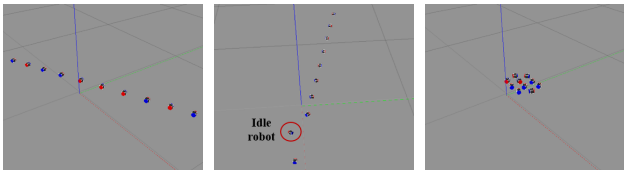
For the group merging-based decentralized exploration workload **W4**, we consider two strategies. (1) The Byzantine robot conducts a simple false map generation attack (section 5.2.1). Figure 17 shows the corresponding results. The falsified map from the Byzantine robot cannot be merged into the exploration cluster, which increases the overall execution time for other benign robots to explore. But the quality of the final map is unchanged. (2) The Byzantine robot introduces a partially fake map to block the grouping process (Section 5.2.2). It can firstly form a group with several robots within its communication range and poison the map via its fake map information, causing the rest of the robots not able to join the group. As a result, the mission duration increases, and the maps cannot be correctly merged (Figure 18).

Summary: Based on the above analysis, we conclude that both centralized (**W2**) and decentralized (**W3** and **W4**) exploration workloads are vulnerable to the map merging poisoning attack. Even



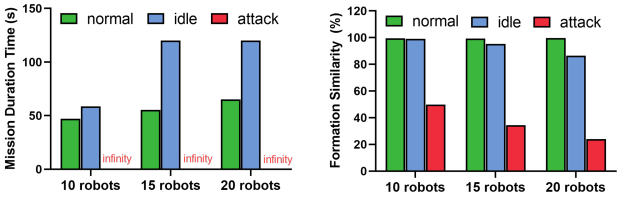
(a) Mission duration time. (b) Map quality.

Figure 18: Blocking group merging attack against W4.



(a) *Normal.* (b) *Idle.* (c) *Attack.*

Figure 19: Visualized attack effects for W5.



(a) Mission duration time. (b) Formation Similarity.

Figure 20: Task forwarding attack against W5.

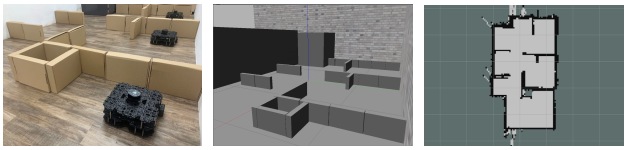
though a decentralized workload can mitigate some simple attacks, a smarter adversary can still craft fake maps to cause task failures.

6.2.3 Task Forwarding Attack. Finally, we consider this type of attack against the swarm formation workload (**W5**). The Byzantine robot can control the system formation behaviors by forwarding false task information. Figure 19 compares the formations of robots in the *normal* (a), *idle* (b) and *attack* (c) situations, when 10 robots are instructed to generate a line formation. Different from previous workloads, the *idle* robot can also affect the workload completion (i.e., formation). This is because even though the idle Byzantine robot does nothing, other robots can observe it and make formation decisions based on its wrong position. Hence, the idle robot increases the mission completion time and decrease the formation similarity. In the *attack* situation, the Byzantine robot has more severe impact on the formation since it can mislead other robots proactively to perform a wrong formation. Quantitative results are presented in Figure 20, where the formation similarity is calculated based on Section 4.2.

Summary: Systems that allow robots to propagate task information as middleman are vulnerable to this task forwarding attack. The selected swarm formation workload **W5** is a typical example.

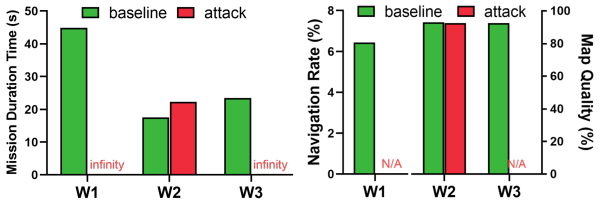
7 REAL-WORLD EVALUATION

To fully validate the Byzantine threats, we implement an MRS and deploy the attacks against different workloads in the physical world.



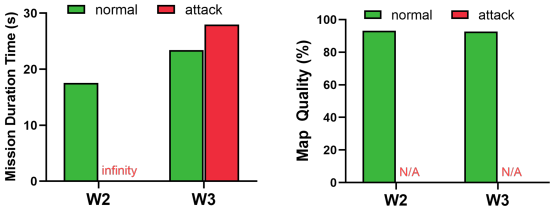
(a) Actual environment (b) 3D view in Gazebo (c) 2D view in rviz

Figure 21: Physical environment for real-world evaluation.



(a) Mission duration time. (b) Task-specific metrics.

Figure 22: Task assignment control attack against W1, W2 and W3 (Physical).



(a) Mission duration time. (b) Map quality.

Figure 23: Map merging poisoning attack against W2 and W3 (Physical).

7.1 Experimental Setup

Our testing environment is a $2.5 \times 5 m^2$ maze. We adopt three Turtlebot3 devices to form a Multi-Robot System. Each robot is equipped with a Raspberry Pi 3 chip [64] as the on-board processor, and a 360-degree 2D laser scanner [9] for SLAM. The ROS core nodes are deployed on a Ubuntu 16.04 server connected to the robots through the wireless network. Figure 21 shows the environment with the corresponding 3D view from Gazebo and 2D view from Rviz.

Due to the limited physical space and number of robots, we only implement the navigation workload **W1**, exploration workloads **W2** and **W3**. We believe the conclusions will be applied to the other two workloads too. For the centralized workloads **W1** and **W2**, the GCS nodes performing the path planning and map merging tasks are running on a server connected to the robots directly. For the decentralized workload **W3**, the processing nodes (path planning, mapping, etc.) of each robot are deployed on a server due to the limited computation capabilities of the on-board processor. To simulate the restricted communication range in the MRS, two robots are forbidden to share information if their distance is beyond a threshold (1m in our experiments). We launch the task assignment control attacks against **W1**, **W2** and **W3**, and the map merging poisoning attacks against **W2** and **W3**. We only compare the *normal* and *attack* situations.

7.2 Evaluation Results

Task assignment control attack. Figure 22 shows the impact of this attack against three workloads. For **W1**, We observe the task

can never be completed with a navigation rate of zero. For **W2**, the map can be finally constructed much longer time (27.7% increase in mission duration). The workload **W3** cannot be completed since the exploration tasks are not assigned to benign robots. These results are in general consistent with the simulation results in Section 6.

Map merging poisoning attack. Figure 23 shows the attack results against the two exploration workloads **W2** and **W3**. **W2** fails to complete, as the map at GCS is poisoned by the falsified data from the Byzantine robot. For **W3**, the robots can still perform and complete the exploration tasks. However, the final map cannot be correctly merged due to the falsified map sent by the Byzantine robot. These also match the simulated results in Section 6.

8 DISCUSSIONS AND RELATED WORKS

8.1 Countermeasures

MRS developers focus more on the development of motion algorithms to guarantee motion safety and task achievement, while ignoring the severity of Byzantine threats. To our best knowledge, there are no practical defense solutions deployed in current MRSs. We discuss two possible countermeasures that can help to alleviate the discovered Byzantine risks from Section 5. We expect that they can be adopted to enhance the security of MRSs in the near future.

The first direction is to implement message checking in MRSs. Messages sent by a robot imply their physical status, which should comply with some system rules. For instance, the distance between two positions of a robot recorded at two consecutive timestamps should be shorter than the maximum speed of the robot times the period duration. When a Byzantine robot launches the task assignment control attack in the navigation workload (Section 5.1), this system rule will be violated and GCS can detect the anomaly. We can design the corresponding rules for each type of communication messages, and enforce the rule checking in every robot in real-time. However, the Byzantine robot may realize such rules and carefully craft malicious messages that are not recognizable, but can still affect the system. How to design robust rules to reduce such possibilities is challenging but important as future work.

The second direction is to apply consensus protocols together with new coordination schemes to protect MRSs. A resilient consensus protocol [73] was introduced in swarm workloads such as formation control, flocking, and sensor fusion to detect Byzantine agents. Its main idea is that the system can be resilient to a number of F non-cooperative nodes by actively verifying information with neighbors as long as the network connectivity of the system is above $(2F + 1)$. Strobel et al. [82] leveraged the blockchain technology to detect and exclude Byzantine robots in a swarm system. Those methods require the system to have very large number of robots with high connectivity, which may not be realistic in some practical scenarios. Besides, the system's efficiency will be sacrificed since some messages may only contribute to the information verification instead of the actual workload. In the future, we will consider to design more efficient and comprehensive communication schemes and consensus protocols for various types of MRSs.

8.2 Related Works

Byzantine faults in Multi-Robot Systems. Byzantine faults in MRSs were first discussed and modeled as a convergence problem of robot networks, i.e., a set of robots are required to asymptotically reach the same but prior unknown location. Bouzid et al. [14] proved the necessary and sufficient conditions to achieve convergence under Byzantine attacks in *Obvious Robot Networks*, where robots cannot recall past computations and can only move in one-dimensional space. Bouzid et al. [13] extended the mathematical theory to two other swarm systems based on the ATOM model [84] and CORDA model [34]. Auger et al. [10] developed a certified framework to prove the convergence of robot networks using the COQ proof assistant. Molla et al. [58] designed deterministic algorithms to identify the lower bounds of time and memory for solving the dispersion problem on a ring of robots. Zikratov et al. [106] proposed a trust management framework to identify malicious Byzantine entities in multi-agent systems.

These prior works mainly focused on the theories of Byzantine faults with very simple robotic functionalities and tasks. In contrast, this paper presents the first *practical* study about the Byzantine threats in real-world implementations based on the Robot Operating System framework. We investigate the impact of Byzantine attacks on the complex workloads (e.g., navigation, exploration) with different coordination schemes. We also evaluate the discovered Byzantine attacks with both accurate simulations and physical experiments. These are never achieved in previous works.

Detecting vulnerabilities in robotic systems. Pogliani et al. [65] designed a new methodology to perform data flow analysis and discover vulnerabilities in the source code of industrial robot software. Recently, researchers applied the fuzzing technique to study the security and safety of robotic systems and Autonomous Vehicles (AVs). For instance, CPFuzz [77] was designed to find the safety violations in cyber-physical systems. RVFuzzer [48] fuzzes the configuration parameters and environmental factors to identify input validations bugs in robotic vehicles. PGFuzz [47] is a policy-guided fuzzing framework to discover any policy violations in the control programs of robotic vehicles. Fuzz testing for AVs usually focuses on a single vehicle [30, 36, 40, 52]. For example, [36] illustrated the application of fuzzing to test the vehicle's CAN bus; Li *et al* [52] proposed a testing framework, AV-FUZZER, to find safety violations of an autonomous driving system.

Different from the above works, our fuzzing method focuses on MRSs. For instance, PGFuzz fuzzes the input controller command and environmental variables to discover the potential vulnerabilities. It targets one single robot of specific kind, and the temporal logic formulas are extracted from the specification documents. In contrast, our work focuses on the interaction of multiple robots in a collaborative workload, and do not rely on the specification documents. Moreover, we propose different requirements for the secure and safe operation of an MRS in the STL formulas. We also identify the critical messages as the fuzzing space, and different strategies to mutate these messages for testing. This method is effective to identify Byzantine threats in an MRS implementation.

Other attacks against robotic systems. Past works have demonstrated that a variety of robotic components are vulnerable, and prone to different types of attacks. For instance, sensor spoofing attacks can spoof the sensor data (e.g. GPS [61, 75, 88, 95, 104], Lidar data [18, 79], optical images [22], gyroscopic data [81, 91, 92]) to cause the robots to make wrong decisions. An adversary can also tamper with the controller input (e.g., configuration or calibration

parameters, perceived states), making the robot instable, halt, or rush to wrong directions [68]. Moreover, recent works disclosed many known cyber security issues in the ROS framework, such as plaintext communication, lack of authentication or authorization [19], and denial-of-service vulnerability [28]. Finally, malware based on machine learning techniques [21] is also developed to maximize the attack impacts on ROS applications.

While the cyber-attacks against individual robots have been studied, research about the security of Multi-Robot Systems is still at an early stage. This drives us to investigate and evaluate different attack strategies and their damages on various MRS workloads.

9 CONCLUSION

In this paper, we perform an investigation towards the Byzantine threats in MRS workloads from the ROS platform. We propose a requirement-driven fuzzing methodology, which can automatically analyze potential Byzantine risks in an MRS workload. We build an MRS workload suite containing common implementations of typical Multi-Robot workloads and coordination schemes to evaluate our method. We identify three new forms of Byzantine attacks with five attack strategies, which are further validated by simulation and real-world experiments. We expect this study can raise the awareness of robotics researchers and developers about the severity of MRS Byzantine threats, and design new solutions to enhance the security and safety of existing MRSs.

ACKNOWLEDGMENTS

We thank our Shepherd Dr. Stefano Zanero and anonymous reviewers for their valuable comments. This work was supported in part by Singapore Ministry of Education (MOE) AcRF Tier 1 RG108/19 (S), NTU-Desay Research Program 2018-0980, Singapore MOE Academic Research Fund Tier 2 grant (MOE-T2EP20120-0004), Singapore National Research Foundation (NRF) under its National Cybersecurity R&D Program (NRF2018NCR-NCR005-0001 and NRF2018NCR-NSOE003-0001), and NRF Investigatorship (NRF-NRFI06-2020-0001).

REFERENCES

[1] 2020. Dji Onboard SDK. https://developer.dji.com/onboard-sdk/.
[2] 2020. Open source robot operating system. https://index.ros.org/
[3] 2020. Robot Vulnerability Database (RVD). https://github.com/aliasrobotics/RVD/.
[4] 2020. ROS 2 Robotic Systems Threat Model. https://design.ros2.org/articles/ros2_threat_model.html.
[5] 2020. ROS ABB Package. http://wiki.ros.org/abb/.
[6] 2020. ROS Index package list. https://index.ros.org/packages/
[7] 2020. ROS PR2 Package. http://wiki.ros.org/Robots/PR2/.
[8] 2021. TurtleBot official website. https://www.turtlebot.com/
[9] 2021. TurtleBot3 LDS-01 Laser Sensor. https://emanual.robotis.com/docs/en/platform/turtlebot3/appendix_lds_01/
[10] Cédric Auger, Zohir Bouzid, Pierre Courtieu, Sébastien Tixeuil, and Xavier Urbain. 2013. Certified Impossibility Results for Byzantine-Tolerant Mobile Robots. In *Stabilization, Safety, and Security of Distributed Systems*, Teruo Higashino, Yoshiaki Katayama, Toshimitsu Masuzawa, Maria Potop-Butucaru, and Masafumi Yamashita (Eds.). Springer International Publishing, Cham, 178–190.
[11] Laura Barnes, MaryAnne Fields, and Kimon Valavanis. 2007. Unmanned ground vehicle swarm formation control using potential fields. In *2007 Mediterranean Conference on Control Automation*. 1–8.
[12] Marc Berhault, He Huang, Pinar Keskinocak, Sven Koenig, Wedad Elmaghraby, Paul Griffin, and Anton Kleywegt. 2003. Robot exploration with combinatorial auctions. In *Proceedings 2003 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS 2003)*, Vol. 2. IEEE, 1957–1962.
[13] Zohir Bouzid, Maria Gradinariu Potop-Butucaru, and Sébastien Tixeuil. 2009. Byzantine Convergence in Robot Networks: The Price of Asynchrony. In *Principles of Distributed Systems*, Tarek Abdelzaher, Michel Raynal, and Nicola Santoro (Eds.). Springer Berlin Heidelberg, Berlin, Heidelberg, 54–70.
[14] Zohir Bouzid, Maria Gradinariu Potop-Butucaru, and Sébastien Tixeuil. 2009. Byzantine-Resilient Convergence in Oblivious Robot Networks. In *Distributed Computing and Networking*, Vijay Garg, Roger Wattenhofer, and Kishore Kothapalli (Eds.). Springer Berlin Heidelberg, Berlin, Heidelberg, 275–280.
[15] Sebastian Buck, Richard Hanten, C. Robert Pech, and Andreas Zell. 2017. Synchronous Dataflow and Visual Programming for Prototyping Robotic Algorithms. In *Intelligent Autonomous Systems 14*, Weidong Chen, Koh Hosoda, Emanuele Menegatti, Masahiro Shimizu, and Hesheng Wang (Eds.). Springer International Publishing, Cham, 911–923.
[16] Wolfram Burgard, Mark Moors, Cyrill Stachniss, and Frank E Schneider. 2005. Coordinated Multi-Robot Exploration. *IEEE Trans. Robot.* 21, 3 (2005), 376–386.
[17] Flavio Cabrera-Mora and Jizhong Xiao. 2012. A flooding algorithm for multi-robot exploration. *IEEE Transactions on Systems, Man, and Cybernetics, Part B (Cybernetics)* 42, 3 (2012), 850–863.
[18] Yulong Cao, Chaowei Xiao, Benjamin Cyr, Yimeng Zhou, Won Park, Sara Rampazzi, Qi Alfred Chen, Kevin Fu, and Z. Morley Mao. 2019. Adversarial Sensor Attack on LiDAR-based Perception in Autonomous Driving. In *ACM Conference on Computer and Communications Security (CCS)*. 2267–2281.
[19] Cesar Cerrudo and Lucas Apa. 2017. *Hacking robots before skynet.* Technical Report. IOActive, Inc.
[20] Hongjun Choi, Wen-Chuan Lee, Yousra Aafer, Fan Fei, Zhan Tu, Xiangyu Zhang, Dongyan Xu, and Xinyan Xinyan. 2018. Detecting attacks against robotic vehicles: A control invariant approach. In *ACM Conference on Computer and Communications Security*. 801–816.
[21] Keywhan Chung, Xiao Li, Peicheng Tang, Zeran Zhu, Zbigniew T. Kalbarczyk, Ravishankar K. Iyer, and Thenkurussi Kesavadas. 2019. Smart Malware that Uses Leaked Control Data of Robotic Applications: The Case of Raven-II Surgical Robots. In *22nd International Symposium on Research in Attacks, Intrusions and Defenses (RAID 2019)*. USENIX Association, Chaoyang District, Beijing, 337–351.
[22] Drew Davidson, Hao Wu, Rob Jellinek, Vikas Singh, and Thomas Ristenpart. 2016. Controlling UAVs with sensor input spoofing attacks. In *USENIX Workshop on Offensive Technologies*. 221–231.
[23] Mohammadreza Davoodi, Saba Faryadi, and Javad Mohammadpour Velni. 2016. A Graph Theoretic-Based Approach for Deploying Heterogeneous Multi-agent Systems with Application in Precision Agriculture. *IEEE Transactions on Robotics* 32, 6 (2016), 1498–1511.
[24] Mohammadreza Davoodi, Javad Mohammadpour Velni, and Changying Li. 2018. Coverage control with multiple ground robots for precision agriculture. *Mechanical Engineering* 140, 06 (2018), S4–S8.
[25] Nicholas DeMarinis, Stefanie Tellex, Vasileios P Kemerlis, George Konidaris, and Rodrigo Fonseca. 2019. Scanning the internet for ROS: A view of security in robotics research. In *International Conference on Robotics and Automation*. IEEE, Montreal, QC, Canada, 8517–8521.
[26] Jyotirmoy V Deshmukh, Alexandre Donzé, Shromona Ghosh, Xiaoqing Jin, Garvit Juniwal, and Sanjit A Seshia. 2017. Robust online monitoring of signal temporal logic. *Formal Methods in System Design* 51, 1 (2017), 5–30.
[27] Vishal Dey, Vikramkumar Pudi, Anupam Chattopadhyay, and Yuval Elovici. 2018. Security vulnerabilities of unmanned aerial vehicles and countermeasures: An experimental study. In *International Conference on VLSI Design and International Conference on Embedded Systems*. Pune, India, 398 – 403.
[28] Bernhard Dieber, Benjamin Breiling, Sebastian Taurer, Severin Kacianka, Stefan Rass, and Peter Schartner. 2017. Security for the robot operating system. *Robotics and Autonomous Systems* 98 (2017), 192–203.
[29] Valerio Digani, Lorenzo Sabattini, and Cristian Secchi. 2016. A Probabilistic Eulerian Traffic Model for the Coordination of Multiple AGVs in Automatic Warehouses. *IEEE Robotics and Automation Letters* 1, 1 (2016), 26–32.
[30] Tommaso Dreossi, Daniel J Fremont, Shromona Ghosh, Edward Kim, Hadi Ravanbakhsh, Marcell Vazquez-Chanlatte, and Sanjit A Seshia. 2019. Verifai: A toolkit for the formal design and analysis of artificial intelligence-based systems. In *International Conference on Computer Aided Verification*. Springer, 432–442.
[31] Juan C. Elizondo-Leal, Gabriel Ramírez-Torres, and Gregorio Toscano Pulido. 2008. Multi-robot Exploration and Mapping Using Self Biddings and Stop Signals. In *MICAI 2008: Advances in Artificial Intelligence*, Alexander Gelbukh and Eduardo F. Morales (Eds.). Springer Berlin Heidelberg, Berlin, Heidelberg, 615–625.
[32] Alessandro Farinelli, Luca Iocchi, and Daniele Nardi. 2016. Distributed on-line dynamic task assignment for multi-robot patrolling. *Autonomous Robots* 41, 6 (2016), 1321–1345.
[33] Fan Fei, Zhan Tu, Ruikun Yu, Taegyu Kim, Xiangyu Zhang, Dongyan Xu, and Xinyan Deng. 2018. Cross-layer retrofitting of UAVs against cyber-physical attacks. In *IEEE International Conference on Robotics and Automation*. 550–557.
[34] Paola Flocchini, Giuseppe Prencipe, Nicola Santoro, and Peter Widmayer. 2000. Distributed coordination of a set of autonomous mobile robots. In *Proceedings of the IEEE Intelligent Vehicles Symposium*. IEEE, 480–485.

[35] Dariush Forouher, Jan Hartmann, and Erik Maehle. 2014. Data Flow Analysis in ROS. In *ISR/Robotik 2014; 41st International Symposium on Robotics*. 1–6.

[36] Daniel S Fowler, Jeremy Bryans, Siraj Ahmed Shaikh, and Paul Wooderson. 2018. Fuzz testing for automotive cyber-security. In *2018 48th Annual IEEE/IFIP International Conference on Dependable Systems and Networks Workshops (DSN-W)*. IEEE, 239–246.

[37] Dieter Fox, Jonathan Ko, Kurt Konolige, Benson Limketkai, Dirk Schulz, and Benjamin Stewart. 2006. Distributed Multirobot Exploration and Mapping. *Proc. IEEE* 94, 7 (2006), 1325–1339.

[38] Stuart Golodetz, Tommaso Cavallari, Nicholas A Lord, Victor A Prisacariu, David W Murray, and Philip HS Torr. 2018. Collaborative large-scale dense 3d reconstruction with online inter-agent pose optimisation. *IEEE Transactions on Visualization and Computer Graphics* 24, 11 (2018), 2895–2905.

[39] Hongliang Guo, Yan Meng, and Yaochu Jin. 2011. Swarm robot pattern formation using a morphogenetic multi-cellular based self-organizing algorithm. In *2011 IEEE International Conference on Robotics and Automation*. 3205–3210.

[40] Jia Cheng Han and Zhi Quan Zhou. 2020. Metamorphic Fuzz Testing of Autonomous Vehicles. In *Proceedings of the IEEE/ACM 42nd International Conference on Software Engineering Workshops*. 380–385.

[41] Wenbiao Han and Mohsen A. Jafari. 2003. Controller synthesis via mapping task sequence to petri nets in multi-agent collaboration applications. In *IEEE International Conference on Robotics and Automation (ICRA)*. 4312–4317.

[42] Abdelfetah Hentout, Abderraouf Maoudj, Nesrine Kaid-Youcef, Djamila Hebib, and Brahim Bouzouia. 2020. Distributed multi-agent bidding-based approach for the collaborative mapping of unknown indoor environments by a homogeneous mobile robot team. *Journal of Intelligent Systems* 29, 1 (2020), 84–99.

[43] Jiří Hörner. 2016. Map-merging for multi-robot system. https://is.cuni.cz/webapps/zzp/detail/174125/

[44] Donghwa Jeong and Kiju Lee. 2013. InchBot: A novel swarm microrobotic platform. In *2013 IEEE/RSJ International Conference on Intelligent Robots and Systems*. 5565–5570.

[45] Kelin Jose and Dilip Kumar Pratihar. 2016. Task allocation and collision-free path planning of centralized multi-robots system for industrial plant inspection using heuristic methods. *Robotics and Autonomous Systems* 80 (2016), 34–42.

[46] Michael E. Kepler and Daniel J. Stilwell. 2020. An Approach to Reduce Communication for Multi-agent Mapping Applications. In *IEEE/RJS International Conference on Intelligent RObots and Systems (IROS)*. 4814–4820.

[47] Hyungsub Kim, Muslum Ozgur Ozmen, Antonio Bianchi, Z Berkay Celik, and Dongyan Xu. 2021. PGFUZZ: Policy-Guided Fuzzing for Robotic Vehicles. In *Network and Distributed System Security Symposium*.

[48] Taegyu Kim, Chung Hwan Kim, Junghwan Rhee, Fan Fei, Zhan Tu, Gregory Walkup, Xiangyu Zhang, Xinyan Deng, and Dongyan Xu. 2019. RVFUZZER: Finding Input Validation Bugs in Robotic Vehicles through Control-Guided Testing. In *Proceedings of the 28th USENIX Conference on Security Symposium* (Santa Clara, CA, USA) *(SEC'19)*. USENIX Association, USA, 425–442.

[49] Nathan Koenig and Andrew Howard. 2004. Design and use paradigms for Gazebo, an open-source multi-robot simulator. In *2004 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, Vol. 3. 2149–2154.

[50] CG Leela Krishna and Robin R Murphy. 2017. A review on cybersecurity vulnerabilities for unmanned aerial vehicles. In *IEEE International Symposium on Safety, Security and Rescue Robotics*. 194–199.

[51] N Vimal Kumar and C Selva Kumar. 2018. Development of collision free path planning algorithm for warehouse mobile robot. *Procedia computer science* 133 (2018), 456–463.

[52] Guanpeng Li, Yiran Li, Saurabh Jha, Timothy Tsai, Michael Sullivan, Siva Kumar Sastry Hari, Zbigniew Kalbarczyk, and Ravishankar Iyer. 2020. AV-FUZZER: Finding Safety Violations in Autonomous Driving Systems. In *2020 IEEE 31st International Symposium on Software Reliability Engineering (ISSRE)*. IEEE, 25–36.

[53] Guannan Li, David St-Onge, Carlo Pinciroli, Andrea Gasparri, Emanuele Garone, and Giovanni Beltrame. 2019. Decentralized progressive shape formation with robot swarms. *Autonomous Robots* 43, 6 (2019), 1505–1521.

[54] Yugang Liu and Goldie Nejat. 2016. Multirobot Cooperative Learning for Semi-autonomous Control in Urban Search and Rescue Applications. *Journal of Field Robotics* 33, 4 (2016), 512–536.

[55] Joaquin López, Diego Pérez, Enrique Paz, and Alejandro Santana. 2013. WatchBot: A building maintenance and surveillance system based on autonomous robots. *Robotics and Autonomous Systems* 61, 12 (2013), 1559–1571.

[56] Víctor Mayoral-Vilches, Martin Pinzger, Stefan Rass, Bernhard Dieber, and Endika Gil-Uriarte. 2020. Can ROS be used securely in industry? Red teaming ROS-Industrial. arXiv:2009.08211 [cs.RO]

[57] Jarrod R. Mcclean and Charles Farrar. 2013. A Preliminary Cyber-Physical Security Assessment of the Robot Operating System (ROS). In *Proceedings of SPIE*. 874110–1 – 874110–8.

[58] Anisur Rahaman Molla, Kaushik Mondal, and William K. Moses Jr au2. 2020. Efficient Dispersion on an Anonymous Ring in the Presence of Weak Byzantine Robots. arXiv:2004.11439 [cs.DC]

[59] Karl Muecke and Brian Powell. 2011. A Distributed, Heterogeneous, Target-Optimized Operating System for a Multi-robot Search and Rescue Application. *IEAAIE* 2 (2011), 266–275.

[60] Dejan Ničković and Tomoya Yamaguchi. 2020. RTAMT: Online robustness monitors from STL. In *International Symposium on Automated Technology for Verification and Analysis*. Springer, 564–571.

[61] NTyler Nighswander, Brent M. Ledvina, Jonathan Diamond, Robert Brumley, and David Brumley. 2012. GPS software attacks. In *ACM Conference on Computer and Communications Security (CCS)*. 450–461.

[62] Patrick Nolan, Derek A. Paley, and Kenneth Kroeger. 2017. Multi-UAS path planning for non-uniform data collection in precision agriculture. In *IEEE Aerospace Conference*. 1–12.

[63] Tenda Okimoto, Tony Ribeiro, Damien Bouchabou, and Katsumi Inoue. 2016. Mission Oriented Robust Multi-Team Formation and Its Application to Robot Rescue Simulation. In *International Joint Conference on Artificial Intelligence (IJCAI)*. 454–460.

[64] Raspberry Pi. [n.d.]. https://www.raspberrypi.org/

[65] Marcello Pogliani, Federico Maggi, Marco Balduzzi, Davide Quarta, and Stefano Zanero. 2020. Detecting Insecure Code Patterns in Industrial Robot Programs. In *Proceedings of the 15th ACM Asia Conference on Computer and Communications Security*. 759–771.

[66] David Portugal, Luca Iocchi, and Alessandro Farinelli. 2019. *A ROS-Based Framework for Simulation and Benchmarking of Multi-robot Patrolling Algorithms*. Springer International Publishing, Cham, 3–28.

[67] Veerajagadheswar Prabakaran, Mohan Rajesh Elara, Thejus Pathmakumar, and Shunsuke Nansai. 2018. Floor cleaning robot with reconfigurable mechanism. *Automation in Construction* 91 (2018), 155–165.

[68] Davide Quarta, Marcello Pogliani, Mario Polino, Federico Maggi, Andrea Maria Zanchettin, and Stefano Zanero. 2017. An experimental security analysis of an industrial robot controller. In *IEEE Symposium on Security and Privacy*. 268–286.

[69] Robert Reid, Andrew Cann, Calum Meiklejohn, Liam Poli, Adrian Boeing, and Thomas Braunl. 2013. Cooperative multi-robot navigation, exploration, mapping and object detection with ROS. In *2013 IEEE Intelligent Vehicles Symposium (IV)*. 1083–1088.

[70] Juan Jesús Roldán, Elena Peña-Tapia, Pablo Garcia-Aunon, Jaime Del Cerro, and Antonio Barrientos. 2019. Bringing Adaptive and Immersive Interfaces to Real-World Multi-Robot Scenarios: Application to Surveillance and Intervention in Infrastructures. *IEEE Access* 7 (2019), 86319–86335.

[71] Isai Roman-Ballesteros and Carlos F Pfeiffer. 2006. A Framework for Cooperative Multi-Robot Surveillance Tasks. In *Electronics, Robotics and Automotive Mechanics Conference (CERMA'06)*, Vol. 2. 163–170.

[72] Ros-Visualization. [n.d.]. ROS RViz 3D visualizer. https://github.com/ros-visualization/rviz

[73] David Saldana, Amanda Prorok, Shreyas Sundaram, Mario FM Campos, and Vijay Kumar. 2017. Resilient consensus for time-varying networks of dynamic agents. In *2017 American Control Conference (ACC)*. 252–258.

[74] Frank E Schneider, Dennis Wildermuth, and Hans-Ludwig Wolf. 2015. ELROB and EURATHLON: Improving search rescue robotics through real-world robot competitions. In *2015 10th International Workshop on Robot Motion and Control (RoMoCo)*. 118–123.

[75] Seong-Hun Seo, Byung-Hyun Lee, Sung-Hyuck Im, and Gyu-In Jee. 2015. Effect of Spoofing on Unmanned Aerial Vehicle using Counterfeited GPS Signal. *Journal of Positioning, Navigation, and Timing* 4, 2 (2015), 57–65.

[76] Franciszek Seredynski. 1997. Competitive Coevolutionary Multi-Agent Systems: The Application to Mapping and Scheduling Problems. *J. Parallel and Distrib. Comput.* 47, 1 (1997), 39–57.

[77] Fute Shang, Buhong Wang, Tengyao Li, Jiwei Tian, and Kunrui Cao. 2020. CPFuzz: Combining Fuzzing and Falsification of Cyber-Physical Systems. *IEEE Access* 8 (2020), 166951–166962.

[78] Weihua Sheng, Qingyan Yang, Jindong Tan, and Ning Xi. 2006. Distributed multi-robot coordination in area exploration. *Robotics and Autonomous Systems* 54, 12 (2006), 945–955.

[79] Hocheol Shin, Dohyun Kim, Yujin Kwon, and Yongdae Kim. 2017. Illusion and Dazzle: Adversarial Optical Channel Exploits Against Lidars for Automotive Applications. In *International Workshop on Cryptographic Hardware and Embedded Systems (CHES)*. 445–467.

[80] Andrew J Smith and Geoffrey A Hollinger. 2018. Distributed inference-based multi-robot exploration. *Autonomous Robots* 42, 8 (2018), 1651–1668.

[81] Yunmok Son, Hocheol Shin, Dongkwan Kim, Young-Seok Park, Juhwan Noh, Kibum Choi, Jungwoo Choi, and Yongdae Kim. 2015. Rocking Drones with Intentional Sound Noise on Gyroscopic Sensors. In *USENIX Security Symposium (USENIX Security 15)*. 881–896.

[82] Volker Strobel, Eduardo Castelló Ferrer, and Marco Dorigo. 2018. Managing Byzantine Robots via Blockchain Technology in a Swarm Robotics Collective Decision Making Scenario. In *Proceedings of the 17th International Conference on Autonomous Agents and MultiAgent Systems* (Stockholm, Sweden) *(AAMAS '18)*. International Foundation for Autonomous Agents and Multiagent Systems, Richland, SC, 541–549.

[83] Michael Sutton, Adam Greene, and Pedram Amini. 2007. *Fuzzing: Brute Force Vulnerability Discovery*. Addison-Wesley Professional.

[84] Ichiro Suzuki and Masafumi Yamashita. 1999. Distributed Anonymous Mobile Robots: Formation of Geometric Patterns. *SIAM J. Comput.* 28 (1999), 1347–1363.

[85] Mahmoud Tavakoli, Gonçalo Cabrita, Ricardo Faria, Lino Marques, and Anibal T. de Almeida. 2012. Cooperative multi-agent mapping of three-dimensional structures for pipeline inspection applications. *International Journal of Robotics Research* 31, 12 (2012), 1489–1503.

[86] Sebastian Thrun, Wolfram Burgard, and Dieter Fox. 2000. A Real-Time Algorithm for Mobile Robot Mapping With Applications to Multi-Robot and 3D Mapping. In *IEEE International Conference on Robotics and Automation (ICRA)*. IEEE, 321–328.

[87] Global Times. 2021. Mainframe malfunction causes dozens of drones to crash into building in SW China. https://www.globaltimes.cn/page/202101/1214165.shtml.

[88] Nils Ole Tippenhauer, Christina Pöpper, Kasper Bonne Rasmussen, and Srdjan Capkun. 2011. On the requirements for successful GPS spoofing attacks. In *ACM Conference on Computer and Communications Security (CCS)*. 75–86.

[89] Pratap Tokekar, Joshua Vander Hook, David J. Mulla, and Volkan Isler. 2016. Sensor Planning for a Symbiotic UAV and UGV System for Precision Agriculture. *IEEE Transactions on Robotics* 32, 6 (2016), 1498–1511.

[90] Craig Tovey, Michail G Lagoudakis, Sonal Jain, and Sven Koenig. 2005. The generation of bidding rules for auction-based robot coordination. In *Multi-Robot Systems. From Swarms to Intelligent Automata Volume III*. Springer, 3–14.

[91] Timothy Trippel, Ofir Weisse, Wenyuan Xu, Peter Honeyman, and Kevin Fu. 2017. WALNUT: Waging doubt on the integrity of MEMS accelerometers with acoustic injection attacks. In *2017 IEEE European symposium on security and privacy (EuroS&P)*. IEEE, 3–18.

[92] Yazhou Tu, Zhiqiang Lin, Insup Lee, and Xiali Hei. 2018. Injected and Delivered: Fabricating Implicit Control over Actuation Systems by Spoofing Inertial Sensors. In *USENIX Security Symposium (USENIX Security 18)*. 1545–1562.

[93] Victor Mayoral Vilches, Lander Usategui San Juan, Bernhard Dieber, Unai Ayucar Carbajo, and Endika Gil-Uriarte. 2019. Introducing the Robot Vulnerability Database (RVD). *CoRR* abs/1912.11299 (2019). arXiv:1912.11299 http://arxiv.org/abs/1912.11299

[94] Hanlin Wang and Michael Rubenstein. 2020. Shape Formation in Homogeneous Swarms Using Local Task Swapping. *IEEE Transactions on Robotics* 36, 3 (2020), 597–612.

[95] Jon S Warner and Roger G Johnston. 2002. A simple demonstration that the global positioning system (GPS) is vulnerable to spoofing. *Journal of Security Administration* 25, 2 (2002), 19–27.

[96] Wikipedia. 2021. Byzantine fault — Wikipedia, the free encyclopedia. https://en.wikipedia.org/wiki/Byzantine_fault

[97] Zhi Yan, Luc Fabresse, Jannik Laval, and Noury Bouraqadi. 2014. Team Size Optimization for Multi-robot Exploration. In *In Proceedings of the 4th International Conference on Simulation, Modeling, and Programming for Autonomous Robots (SIMPAR 2014)*. Bergamo, Italy, 438–449.

[98] Zhi Yan, Luc Fabresse, Jannik Laval, and Noury Bouraqadi. 2015. Metrics for performance benchmarking of multi-robot exploration. In *2015 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. 3407–3414.

[99] Liu Yang. 2020. Swarm Robot ROS Sim. https://github.com/yangliu28/swarm_robot_ros_sim.

[100] Jing Yuan, Yalou Huang, Tong Tao, and Fengchi Sun. 2010. A cooperative approach for multi-robot area exploration. In *2010 IEEE/RSJ International Conference on Intelligent Robots and Systems*. IEEE, 1390–1395.

[101] Sa Wang Yungang Bao Yuan Xu, Tianwei Zhang. 2021. Towards Practical Cloud Offloading for Low-cost Ground Vehicle Workloads. In *IEEE International Parallel & Distributed Processing Symposium (IPDPS)*.

[102] Tianwei Zhang Han Qiu Yungang Bao Yuan Xu, Gelei Deng. 2021. Novel Denial-of-Service Attacks against Cloud-based Multi-Robot Systems. *Information Sciences* (2021).

[103] Yungang Bao Yuan Xu, Tianwei Zhang. 2021. Analysis and Mitigation of Function Interaction Risks in Robot Apps. In *International Symposium on Research in Attacks, Intrusions and Defenses (RAID)*.

[104] Kexiong (Curtis) Zeng, Shinan Liu, Yuanchao Shu, Dong Wang, Haoyu Li, Yanzhi Dou, Gang Wang, and Yaling Yang. 2018. All Your GPS Are Belong To Us: Towards Stealthy Manipulation of Road Navigation Systems. In *USENIX Security Symposium (USENIX Security 18)*. 1527–1544.

[105] Hai Zhu, Jelle Juhl, Laura Ferranti, and Javier Alonso-Mora. 2019. Distributed multi-robot formation splitting and merging in dynamic environments. In *2019 International Conference on Robotics and Automation (ICRA)*. IEEE, 9080–9086.

[106] Igor Zikratov, Oleg Maslennikov, Ilya Lebedev, Aleksandr Ometov, and Sergey Andreev. 2016. Dynamic Trust Management Framework for Robotic Multi-Agent Systems. In *Internet of Things, Smart Spaces, and Next Generation Networks and Systems*, Olga Galinina, Sergey Balandin, and Yevgeni Koucheryavy (Eds.). Springer International Publishing, Cham, 339–348.