On the (In)Security of Secure ROS2

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

Guowen Xu*
Nanyang Technological University
Singapore
guowen.xu@ntu.edu.sg

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

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

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

ABSTRACT

Robot Operating System (ROS) has been the mainstream platform for research and development of robotic applications. This platform is well-known for lacking security features and efficiency for distributed robotic computations. To address these issues, ROS2 is recently developed by utilizing the Data Distribution Service (DDS) to provide security support. Integrated with DDS, ROS2 is expected to establish the basis for trustworthy robotic ecosystems.

In this paper, we systematically study the security of the current ROS2 implementation from three perspectives. By abstracting the key functions from the ROS2 native implementation, we first formally describe the ROS2 system communication workflow and model it using a concurrent modeling language. Second, we verify the model with some key security properties through a model checker, and successfully identify four security vulnerabilities in ROS2's native security module: Secure ROS2 (SROS2). To validate these flaws, we set up simulation and physical multi-robot testbeds running different real-world workloads developed by Open Robotics and Amazon AWS Robotics. We demonstrate that an adversary can exploit these vulnerabilities to totally invalidate the security protection offered by SROS2, and obtain unauthorized permissions or steal critical information. Third, to enhance the security of ROS2, we propose a general defense solution based on the private broadcast encryption scheme. We run different workloads and benchmarks to show the efficiency and security of our defense. Our findings have been acknowledge by ROS2 official, and the suggested mitigation has been implemented in the latest SROS2 version.

CCS CONCEPTS

• **Computer systems organization** → **Embedded systems**; *Redundancy*; Robotics; • **Networks** → Network reliability.

*Corresponding author

CCS '22, November 7–11, 2022, Los Angeles, CA, USA

ACM ISBN 978-1-4503-9450-5/22/11…$15.00
https://doi.org/10.1145/3548606.3560681

KEYWORDS

Robot Operating System, Protocol Vulnerability, Verification

ACM Reference Format:
Gelei Deng, Guowen Xu, Yuan Zhou, Tianwei Zhang, and Yang Liu. 2022. On the (In)Security of Secure ROS2. In *Proceedings of the 2022 ACM SIGSAC Conference on Computer and Communications Security (CCS '22), November 7–11, 2022, Los Angeles, CA, USA.* ACM, New York, NY, USA, 15 pages. https://doi.org/10.1145/3548606.3560681

1 INTRODUCTION

The robotics technology is playing an important role in the intellectualization of industry and our daily life. Its development is accelerated by the Robot Operating System (ROS). As the most popular robotic platform, ROS provides great ease for developing and managing robotic devices and applications [25]. However, ROS has its limitations by design. It lacks basic security features, leaving ROS-based systems extremely vulnerable [57, 58, 67]. Besides, it is not suitable for multi-robot systems (MRS) in real-time processing. All the robots have to connect to one master node for communication, which makes the system inflexible and inefficient.

To address these problems, ROS2 is developed as an upgrade to ROS. ROS2 uses the Data Distribution Service (DDS) as the communication middleware instead of the traditional master-based communication method, which brings two main advantages. First, DDS allows participants to work in a distributed fashion, which efficiently extends the ROS2-based applications to various multi-robot scenarios [15, 38, 50, 56, 87, 91]. Second, ROS2 develops its native security tool, SROS2, on top of DDS's built-in security modules. SROS2 provides many security features which are missing in ROS, such as network traffic encryption, authentication and access control. With these benefits, ROS2 rapidly gains huge popularity. An increased number of IT and robotic companies adopt ROS2 to develop their robotic products (e.g., Amazon Robomaker [80], iRobot [8], etc.)

While ROS2 aims to provide better protection than ROS, there are still unsolved security concerns about it. (1) The security of the new features in ROS2 is not thoroughly verified. These features may contain loopholes, which could be exploited to cause severe security, privacy and safety hazards. (2) The multi-robot scenario supported by ROS2 can bring new security challenges. A large quantity of heterogeneous robots from different parties and locations can be coordinated by the cloud service (e.g., Amazon RoboMaker) to complete complex tasks, which could potentially enlarge the attack

surface of the entire system. With the fast adaption of ROS2, a comprehensive study on its security is urgently needed.

In this paper, we present the first systematic investigation about the security of SROS2 with the following contributions. First, we design a method based on the model checking technique [28] for ROS2 verification (Section 4). Modeling every detail of the ROS2 system can be extremely challenging, because it involves multiple layers with thousands of functions, and the corresponding model can contain a huge number of states that may cause the state explosion issue [27]. To overcome this problem while accurately modeling the system, we leverage the code property graph [93] to represent the ROS2 client library and its DDS middleware implementation, and efficiently identify the key functions involved in inter-robot communication. We further eliminate the non-related components from the key function CPG representations, and analyze them to abstract the events describing the ROS2 inter-node communication workflow. Based on this, we model two key ROS2 components (*nodes* and *DDS participants*) and the communication environments as processes driven by those events. We formulate a set of desired security properties based on the official *ROS2 Robotic Systems Threat Model* [5], and leverage a model checker to automatically identify vulnerabilities that can lead to violations of these properties.

Second, with the aforementioned methodology, **we successfully identify four vulnerabilities existing across multiple ROS2 versions**, which can invalidate the SROS2 security mechanisms (Section 5). By exploiting those vulnerabilities, the adversary can (1) bypass access control to send arbitrary malicious messages to unauthorized ROS2 nodes, (2) receive confidential messages from unauthorized topics, or (3) extract sensitive information about the system security settings. We validate the exploitability and practicality of those vulnerabilities using four real-world workloads developed by Open Robotics [61] and Amazon AWS Robotics [80] through both simulation and physical experiments (Section 6). We confirm that a single malicious actor can easily terminate the entire system, mislead other benign robots to crash, and steal users' private information. These vulnerabilities have been reported to Open Robotics, the official maintainer of ROS2, and acknowledged by them. Following our suggestion, temporary mitigation methods have also been integrated into the ROS2 testing version.

Third, to thoroughly address the implementation flaws, we propose a general defense solution customized for ROS2 (Section 7). Patching these vulnerabilities separately requires careful modifications of the ROS2 source code to re-design the SROS2 access control functions, which can be a tremendous and tedious task. Instead, we propose to adopt the private broadcast encryption (PBE) primitive [16] to fundamentally address the security flaws in the SROS2 design. Our solution guarantees to provide secure access control as PBE is proved to have key indistinguishability under chosen-ciphertext attacks (IK-CCA). It can work with ROS2 as an individual security module without additional infrastructure support or modification of the ROS2 source code. We implement our solution as a lightweight Python library that can be imported directly by ROS2 applications. We deploy various workloads in our physical testbed to show that our solution can mitigate the discovered vulnerabilities with acceptable performance and resource overhead. We have open-source our solution on our submission website [9] to benefit the robotics community.

2 BACKGROUND

2.1 Robot Operating System

Robot Operating System (ROS) adopts a *node*-based structure, where each node is an independent process that executes certain functions. A typical robot application comprises many nodes distributed in one or multiple robot devices. These nodes exchange messages with each other to finish the task cooperatively. The node communication follows a publish-subscribe mode through a *topic*: each node can publish *messages* with a customized data structure to a topic, and all the nodes subscribed to that topic will receive the messages.

With more emerging scenarios, the design of ROS exhibits two fundamental drawbacks. First, ROS is not suitable for distributed MRS development. All the network traffics must go through a *master* node, and every robot needs to keep continuous network connection with this node. This makes the master node a single-point-of-failure and performance bottleneck. Second, ROS lacks the basic security mechanisms, and contains many security loopholes. While new security modules were developed by the community to patch these issues, they are not widely adopted in real-world applications. Up to now, the latest official ROS distributions have not included those extensions yet, making the majority of ROS-based systems vulnerable to various attacks [57, 58, 67, 86].

To thoroughly solve these issues, Open Robotics [61] proposed the new Robot Operating System 2 (ROS2) in 2014. ROS2 has the similar client library and user-level API structure as ROS, so previously developed ROS applications can be easily migrated to the ROS2 platform. At the network transport layer, ROS2 adopts the Data Distribution Service (DDS) protocol [30], which has the distributed communication capability and built-in security modules. Therefore, ROS2 enjoys all the functionalities from the original ROS, with new support for distributed computing, better performance and security enhancements. With the increased number of packages and projects migrating from ROS to ROS2, ROS2 is expected to establish the basis for the future robotic ecosystems.

2.2 Data Distribution Service

DDS is a mature middleware protocol adopted in ROS2 for real-time connectivity. It supports a publish-subscribe protocol called Data-Centric Publish-Subscribe (DCPS) [63]. The basic structure of DCPS is illustrated in Figure 1. A global data space is created to contain all the data objects (i.e., DDS topics). These DDS topics are similar as the topic objects in ROS, and can be accessed by DDS processes. A process that publishes or subscribes to a topic is called a *participant*. The communication between participants are regulated by a series of configurable parameters that control the behaviors of DDS, namely Quality of Service (QoS).

ROS2 interacts with DDS by calling the abstract DDS APIs (Figure 1). The userland code defines the function logic in the app, e.g., how the nodes communicate with others through topics, and how the received messages are processed. The code is then interpreted by the ROS2 Client Library (RCL) to form the node-based communication structure. This structure is further processed by the ROS2 DDS Middleware (RMW) to generate the corresponding DDS structure and configuration parameters. Finally, the DDS configurations are passed to the DDS APIs to build the DDS system structure. With these steps, ROS2 nodes and DDS participants establish a

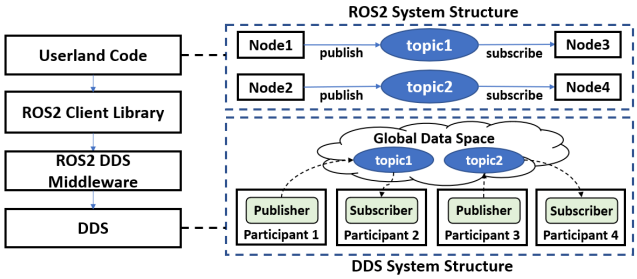


Figure 1: ROS2 DDS architecture with the DCPS protocol.

one-to-one relationship[1]. At runtime, when an ROS2 node tries to publish a message, ROS2 translates such behavior into a series of DDS API calls, and the actual communication is achieved through DDS. In this process, ROS2 works as a middleware and does not handle protocol-level details.

2.3 DDS Security

DDS has its native security specification [40] that adds security protections by defining a series of Service Plugin Interfaces (SPIs). The DDS SPIs provide five security features: authentication, access control, cryptography, logging and data tagging. They can be enabled and configured through the QoS parameters. ROS2 adopts the first three features from DDS as summarized below:

Authentication. This plug-in uses the Public Key Infrastructure (PKI) [59]: each participant has a public-private key pair and an x.509 certificate that binds its public key to its name. Through the PKI, a DDS participant can verify other participants' identity by checking their certificates. Each x.509 certificate must be signed by (or have a signature chain to) one trusted Certificate Authority (CA), which is typically set up by the robotic system owner.

Access control. This plug-in defines and enforces restrictions on the DDS-related capabilities of a given domain participant. It requires two XML files per domain participant, signed by the CA. (1) A *governance file* specifies the domain properties, e.g., if the domain can be joined by other participants, if it can be discovered in the network, etc. (2) A *permission file* specifies the permissions of the domain participant. It declares if a participant can publish or subscribe to specific topics. This permission file is used to configure the access control policies for system participants.

Cryptography. This plug-in declares the cryptography-related operations, e.g., encryption, decryption, signature, etc. Both the authentication and access control plug-ins utilize these primitives to achieve their functions. By default, enabling this plug-in will encrypt all the DDS network traffics using the established Advanced Encryption Standard in Galois Counter Mode (AES-GCM) [79].

2.4 Secure ROS2

ROS2 builds its security mechanisms based on the DDS security specification. The system owner declares the security configurations in the ROS2 userland code, which will be interpreted and passed to the DDS security plug-ins. This set of security features are collectively named "Secure ROS2" (SROS2).

[1]For performance optimization, ROS2 maps multiple nodes to one participant if these nodes share the same configurations. The design rationale is disclosed in [3].

Specifically, SROS2 provides command line integration [76] to enable the SROS2 features. It includes a key generation tool that helps the system owner act as the CA and generate the certificate/key files for the nodes in the system. SROS2 standardizes the security file formats, and specifies how the system owners should distribute those files to the robots. These files need to be put in a specific *keystore* folder following the pre-defined structure and naming rules, so that they can be loaded by SROS2 and passed to DDS as QoS parameters. Enabling SROS2 features brings the following security mechanisms to the system:

Traffic encryption. In the default settings of ROS and ROS2, traffics between nodes are in plaintext. Once SROS2 is enabled, the messages are encrypted by the DDS cryptography plug-in.

Access control. SROS2 enforces access control on the nodes by restricting the underlying DDS participants' capabilities. The system owner provides the governance and permission files for all the nodes. Then each node can only publish/subscribe to the topics declared in its corresponding permission file.

Topic information protection. In ROS, topic-related information is public and can be retrieved by the built-in RCL tool (i.e., `rostopic` [29]), which brings privacy concerns. SROS2 restricts users from reading such information from unauthorized topics, thus protecting the privacy of topics and relevant nodes.

3 THREAT MODEL

3.1 System Assumptions

We consider a distributed MRS where a number of robots collaboratively work on one workload under the guidance of a centralized Ground Control Station (GCS). The system is developed with ROS2 and fully secured by the SROS2 modules. We assume all the configurations are set correctly with the following properties: (1) There exists one physical controller serving as the system owner of the MRS. It defines the system functions through userland codes, and also defines the access control policies for each robot that joins the system. Robot users only have local privilege to control their own robots. (2) A trusted CA is controlled by the system owner and generates unforgeable digital certificates for all the nodes within the MRS. These certificates are distributed to robots by the system owner securely. The system owner has the capability of remotely updating the certificate files stored on the robots at runtime. (3) Network traffic is properly encrypted by the DDS cryptography plug-in. (4) The system owner correctly implements the Mandatory Access Control (MAC) [53] policies by creating the permission files following the SROS2 standards [76].

3.2 Adversary's Capabilities

Following previous works on robotic security [24, 31, 44], we assume that one robot in the MRS is malicious and fully controlled by the adversary. This assumption is realistic due to several reasons: (1) there are software vulnerabilities and bugs in the robotic applications [12], which can be exploited by the adversary to intrude into the system and take full control of a robot. (2) ROS2 has its open-source platform that allows developers over the world to upload and share their function packages [69]. Unfortunately, there is no security check on the submitted code, and an adversary can publish malicious packages for other developers to download [92].

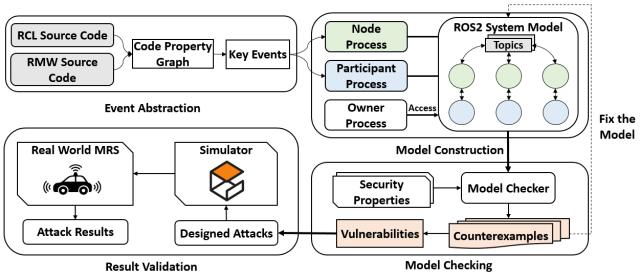


Figure 2: Methodology Overview

(3) Many cloud providers offer cloud-robotic services (e.g., AWS RoboMaker [80], Google Cloud Robotics [7]) to deploy robotic applications across the cloud and local robots. Robots from different parties and locations will be connected and coordinated by the cloud to complete the tasks. It is highly possible that some party is malicious and introduces an adversarial robot into the system, which tries to attack other robots via the interaction with the cloud.

The adversarial robot attempts to invalidate the SROS2 security features (especially the access control mechanism) and execute malicious operations in the MRS. These include (1) retrieving restricted information from unauthorized topics, (2) retrieving private node and topic configuration information, and (3) sending malicious messages to unauthorized topics.

The adversarial robot can perform arbitrary operations locally. However, it has the following limited capabilities when communicating with other actors in the system due to the presence of the SROS2 security mechanisms. (1) Due to the presence of SROS2, it can only communicates with the GCS and other robots by publishing and subscribing to relevant ROS2 topics using the functions defined by the ROS2 client library with valid security files. (2) It cannot forge digital certificates for authentication or break the encryption. However, it has the ability to read and use the certificates installed in its own robot. (3) It can passively eavesdrop all network traffics in its wireless communication range by switching its wireless adaptor to the *promiscuous* mode. This is feasible on vast majority of robots' on-board computers.

4 METHODOLOGY OF INVESTIGATING ROS2

We introduce a methodology to thoroughly inspect the security of ROS2 implementation. It consists of four steps (Figure 2). (1) We first abstract the key events related to the network communication from the ROS2 and SROS2 source code (Section 4.1). (2) We describe the ROS2 system with the formal language CSP# [85] by modeling the nodes, participants and their communications (Section 4.2). (3) We formalize the desired security requirements, and perform model checking on the constructed model under these requirements (Section 4.3). The model checker generates possible counterexamples, which are the system states that violate the requirements. (4) We analyze the counterexamples, summarize the vulnerabilities of SROS2 modules, and further verify their exploitability (Sections 5 and 6).

4.1 ROS2 Abstraction and Modeling

A typical ROS2 workload comprises three basic entities: nodes, participants, and the system owner. They interact with each other through a series of function calls to take actions, including policy updates, message communication, etc. The first step of our methodology is to identify the interactions between these entities and abstract them into a series of events that can be formally described. This approach enables formal verification of the abstracted system, but faces two main challenges. First, it is difficult to accurately identify the function call traces related to communication from the ROS2 source code. ROS2 is a massive system at three implementation levels (high-level API, RCL and RMW) with more than 500k lines of code in a mix of Python, C++ and C languages. Apart from core components for robot communication and control, it also contains numerous feature modules such as ROS1 adaptation, user experience enhancement, etc. Second, the implementation of inter-node communication processes also involve various inner-node functions, such as validating the userland code[2]. These functions are redundant in modeling the communication structure since we focus on the security issues of ROS2 caused by the inter-node actions.

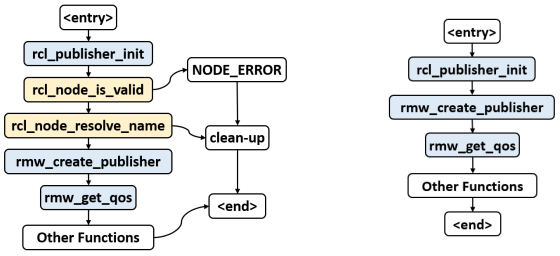
To address the above challenges, we adopt code property graph (CPG) [93] to represent the code structure, shortlist critical functions related to communication, and abstract the key events. CPG is a graph representation that merges the abstract syntax tree, control flow graph and program dependency graph into one joint data structure. Our strategy contains three main steps.

(1). Key Function Identification. We first locate the code sections that process communication messages from the large ROS2 code base. This can be achieved by tracking the data flow that involves the *message* variables in the CPG.

(2). CPG Purification. Next we further purify the CPG by removing the redundant function nodes that handle the inner-node behaviors but do not contribute to inter-node communications. As discussed previously, ROS2 implements validation mechanisms to ensure the validity of userland code. The execution of these functions results in either (a) its caller function continuing to execute if no error is reported, or (b) terminating the caller function execution and throwing an error. Either way will not change the normal interaction relations between the communication-related functions. Therefore, we consider eliminating them from the graph for easier modeling. Since these input validation components do not change the interaction relations between the communication-related functions, they exhibit the same pattern in the control flow of the CPG representation: input validation function nodes have direct outgoing edges to the error handler function nodes, which then terminate the control flow. Leveraging this property, we can efficiently identify them by traversing the graph and examining the outgoing edges for each node. The purified CPG can then be constructed by removing the error handling nodes and joining the other nodes together. Figure 3(a) demonstrates an example of a code snippet in RCL for publisher node creation (`rcl_publisher_init`), which has two functions for userland code validation (`rcl_node_is_valid` and `rcl_node_resolve_name`). By removing these nodes, we can reconstruct the abstracted graph that only includes the communication-related functions in Figure 3(b).

(3). Verification and Analysis. Now the CPG only contains key functions that directly control the interactions between ROS2 system entities. To ensure the correctness of the CPG, we locate the

[2]For instance, user-specified node name will be examined by both RCL and RMW to ensure its uniqueness.



(a) Initial control flow (b) Abstracted control flow

Figure 3: An example of identifying key functions in an RCL code snippet (`rcl_publisher_init`).

key functions in the ROS2 source code and check if their call relations comply with the abstracted CPG callgraph. Then, we manually analyze these key functions to understand the ROS2 inter-node communication workflow. Since the complexity of the CPG has been greatly reduced through previous steps, it is feasible and efficient to conduct verification and analysis manually.

Implementation. We apply a robust parser Joern [93] to parse the source code of RCL and RMW, generate and purify the CPG. Specifically, we first construct the CPG of the functions related to communication, which contains 1283 nodes. We summarize the exception keywords ("*ERROR*", "*err*", etc.) based on ROS2 coding practice and use them to label the error handler functions for CPG purification. After deleting the nodes directly connected to them, we establish the final abstracted CPG that contains 89 function nodes. We analyze these functions and summarize 23 key functions which are critical for inter-node communication. More details of our implementation are available at [9].

We further analyze the key functions and their dependencies, and figure out the inter-node communication workflow, as briefed below. The system owner first passes the security files to the user, who then stores these files in a self-defined path. To create an ROS2 node, the user initializes RCL with the security file path, and calls the `rcl_init_publisher` or `rcl_init_subscription` function depending on whether it is a publisher or subscriber. This function triggers the participant initialization handler in RMW. RMW verifies the integrity of the security files in the provided security path, and loads the access control policies as DDS QoS parameters. When the node publishes a message to a topic, the corresponding DDS participant calls the DDS API with this message. When a subscriber node subscribes to an ROS topic, its participant subscribes to the corresponding DDS topic so that it can receive any messages published to that topic. In this manner, ROS2 translates userland code to a complete communication structure, while the network-level communication is handled by DDS.

4.2 Model Construction

Following the prior works [82, 96], we use CSP# [85] to describe the ROS2 communication system. It is an extension of CSP (Communicating Sequential Processes) [42] that mixes high-level operators with low-level programs for efficient modeling and verification of software systems with concurrent events. This makes it suitable for modeling the abstracted ROS2 system with concurrent node communication processes. Based on the event abstraction in Section 4.1, we define three types of processes: *owner_proc, node_proc* and *parti_proc*. Figure 4 shows the abstracted diagram of our CSP# model. Below we breif the construction of the model for an ROS2 system with N nodes and M topics, while the detailed formal description of each process is available in our supporting material [9].

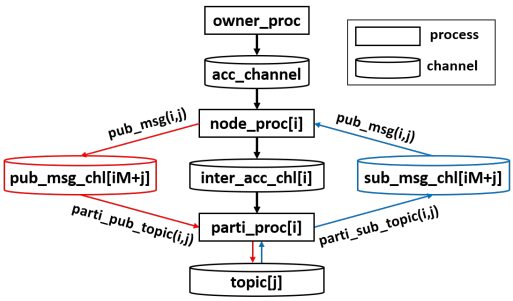


Figure 4: Partial diagram of the CSP# model for the ROS2 system (node i publishes/subscribes to topic j).

(1) *owner_proc* process models the system owner that defines the access control policies and updates them to the nodes through security files. Each security file stores the access control rules for at least one node, and is modeled in an array $owner\_access$: for a giving node i and a topic j, the Boolean vector $owner\_access[i, j] = [x, y, z]$ denotes if this node has publishing (x) and subscription (y) permissions, and knowledge of the topic's configurations (z). Each security file has a path (denoted as $path$) known by its corresponding node(s). Then *owner_proc* stores $path$ to the access channel, denoted as *acc_chl*. There can be N channels in the system, with each one associated to a node.

Nodes and participants should obey the access control policies defined by the system owner. For clear representation, we let $pub_{ij} = owner\_access[i, j][0]$ and $sub_{ij} = owner\_access[i, j][1]$ to denote the publishing and subscription access of node i to topic j.

(2) $node\_proc(i)$ process models ROS2 node i. It first *initializes* itself by loading the security files from the user-defined path, and initializes a participant with the loaded contents. The node does not directly handle the contents of the security file in this step. After initialization, the node can (i) *re-initialize* itself with a new security file path and the corresponding participant; (ii) *publish* messages to the participant via an internal publishing message channel; or (iii) *subscribe* messages from the participant via an internal subscription message channel. Note that while access control is defined at the node level, SROS2 does not enforce access control over the nodes, but relies on DDS to regulate the participants corresponding to the nodes. Thus nodes can freely execute the publishing/subscription functions to arbitrary topics, but the corresponding participants will get rejected if they do not have the proper access.

(3) $parti\_proc(i)$: this process models the participant created by node i. Upon initialization, the participant verifies the integrity of the security file contents provided by the node. It then retrieves the access control policies from the file and saves them into the corresponding internal access channel if the security file is valid. Then, it can (i) retrieve the messages from the internal publishing message channels and send them to the topics in the global data space of the DDS system; (ii) use its identity certificate to retrieve the messages of corresponding topics from the transport protocol and send them to the internal subscription message channels; or (iii) update the access control rules.

4.3 Model Checking

The above formal model enables the security checking of given security requirements and identification of possible violations via a model checker. If the model violates any security requirements, the model checker can automatically generate a counterexample, an execution trace that leads to the violation.

Security requirements. To identify the potential vulnerabilities in the ROS2 implementation, we first describe the desired security requirements for the system. These requirements are summarized from *ROS2 Robotic Systems Threat Model* [5], an official document describing the security goals, assets, and attack vectors in robotic systems. Following the previous work [49], we adopt the requirement engineering [23] technique to manually interpret the document. By mapping the security goals to the assets accessible to MRS participants, we conclude six security requirements for the system. Specifically, **R1** and **R2** are for system completeness, which ensures that all system entities participate in the communication process. **R3** to **R6** describe the security and privacy of the system entities. For each summarized requirement, we further describe it with the LTL (linear temporal logic) [64] formula. Let $\Box$, $\Diamond$, and U be the temporal operators "always", "eventually", and "until"; $\wedge$, $\vee$ and $\rightarrow$ be the logical operators "and", "or", and "implies". The security requirements for the ROS2 system can be formulated as below.

(R1) Each node in the system has access control rules to at least one topic, either for publishing or subscription. Let $npug_{ij}$ and $nsub_{ij}$ be the publishing and subscription access of node i to topic j, then $\Box \wedge_{i=0,\ldots,N-1} \sum_{j=0}^{M-1}(npub_{ij} + nsub_{ij}) >= 1$.

(R2) Each topic is accessible to at least one node to publish messages and at least one node to subscribe messages, i.e., $\Box \wedge_{j=0,\ldots,M-1} \sum_{i=0}^{N-1} npub_{ij} >= 1$ and $\Box \wedge_{j=0,\ldots,M-1} \sum_{i=0}^{N-1} nsub_{ij} >= 1$.

(R3) The access control rules to message publishing of a participant should always be the same as the one declared by the owner.Let pub_{ij} and $ppub_{ij}$ be the system-defined publishing access of node i to topic j, and the access at the participant level, then we have $\Box \wedge_{i=0,\ldots,N-1;j=0,\ldots,M-1} pub_{ij} == ppub_{ij}$.

(R4) The access control rules to message subscription of a participant should always be the same as the one declared by the owner. Let sub_{ij} and $psub_{ij}$ be the system-defined subscription access of node i to topic j, and the access at the participant level, then we have $\Box \wedge_{i=0,\ldots,N-1;j=0,\ldots,M-1} sub_{ij} == psub_{ij}$.

(R5) A participant i can publish (resp., subscribe) to a topic j only when $ppub_{ij} == 1$ (resp., $psub_{ij} == 1$) and the buffer of channel $topic[j]$ is not full (resp., empty). Let p_msg_{ij} and s_msg_{ij} be Boolean variables denoting whether participant i publishes and subscribes to topic j. Let $call(x, chl)$ be querying the buffer information of a channel chl, $cfull$ and $cempty$ denoting whether the channel is full or not. Then we have $\Box \wedge_{i=0,\ldots,N-1;j=0,\ldots,M-1} (p\_msg_{ij} == 1 \rightarrow ppub_{ij} == 1 \wedge call(cfull, topic[j]) == False) \wedge (s\_msg_{ij} == 1 \rightarrow psub_{ij} == 1 \wedge call(cempty, topic[j]) == False)$, where $call(operation, name)$ is a static method to query the buffer information of a channel in the model checker.

(R6) When a node i legally subscribes to a topic j, it can only access the messages sent to the topic by legal nodes, but no other information of the topic's publishers. Let I_j be the nodes that have access to publish messages to topic j, $g(i,k)$ be a Boolean value denoting whether the message subscribed by node i is equal to the message published by node k, and $f(i,k)$ be a Boolean value denoting whether node i knows the access of node k. Then we have $\Box \wedge_{j=1,2,\ldots,M} (nsub_{ij} == 1 \wedge \prod_{k \in I_j} npub_{kj} == 1) \rightarrow (\sum_{k \in I_j} g(i,k) == 1 \wedge \sum_{k \in I_j} f(i,k) == 0)$.

Implementation. Without loss of generality, we apply the popular Process Analysis Toolkit (PAT) tool [84] to automatically verify if the abstracted CSP# model in Section 4.2 satisfies the above security requirements. Particularly, we construct the system based on the SROS2 sample project *chatter* [76], which has two nodes and two topics. This project is selected for two reasons. First, it involves the complete message publishing and subscription process in a well-defined communication structure. Since ROS2 communication is node-to-node basis, increasing the number of nodes and topics does not necessarily increase the complexity of the checked model. Second, this project has the native security implementation developed by ROS2 official. As people develop projects following ROS2 examples, the default misconfigurations in this project can be inherited to other community projects. Thus, we consider this model to be adequate and suitable for identifying vulnerabilities. We implement the concrete system model and initialize the system state based on the project's default security configuration.

By verifying the model against the security requirements, we successfully identify multiple counterexamples in ROS2. Since the formal model is constructed strictly based on the key functions from the ROS2 implementation, all modeled processes and variables can be mapped to concrete objects in the source code. This enables us to quickly examine the related functions in the ROS2 implementation once a violation is detected, and identify the vulnerabilities led by the counterexamples. We analyze these vulnerabilities and demonstrate the exploits in Sections 5 and 6.

4.4 Discussion

While we select the *chatter* project in the system modeling process, our methodology can be applied to any ROS2 project and extended to other systems. This is because our strategy abstracts the ROS2 client library and DDS middleware into formally described processes and events. Fundamentally speaking, ROS2 projects are different only at the userland code level, which calls the low-level functions in different orders and quantities. We can easily model another ROS2 project by changing the number of topics, nodes, participants, and their publishing/subscription relationships.

We design our model checking approach to achieve *soundness* (i.e., each reported violation is indeed a reachable vulnerable system state) instead of *completeness* (i.e., identifying all the possible violations within the system). This is because our system modeling is parameterized by the number of nodes and topics, and it is impossible to achieve completeness due to the undecidability of parameterized system verification problem [13]. Thus, we follow the conventional approaches [44, 45, 49] to aim for soundness instead of completeness. The proposed model abstraction through node elimination results in a certain level of inaccuracy and may leave some vulnerabilities undiscovered. However, this process does not change the interaction relations between the communication-related functions, and thus guarantees the soundness of our approach.

Besides, we follow the official ROS2 threat model [5] to identify the vulnerabilities caused by false interactions between entities

within the system. There exist some vulnerabilities beyond the scope of this threat model, and our methodology will fail to detect them. For instance, we do not consider the function-level vulnerabilities such as improper input sanitization vulnerabilities. Also, we do not consider implicit information leakage via side channels. During our manual analysis, we indeed find one such network side channel in ROS2: when a message is published to a topic, the ROS2 DDS identifies the receiver participants, and sends the message to each one separately. Since the message is the same, the network packets to each participant have the same source IP address, similar packet lengths, and very close timestamps. This allows an adversary to infer sensitive information about other nodes and topics by analyzing the network traffic, even it is encrypted by SROS2. How to formally discover such kinds of vulnerabilities is orthogonal to this work, yet an important direction to explore in our future works.

5 SECURITY VULNERABILITIES IN ROS2

We analyze ROS2 and SROS2 implementations with the proposed methodology. Specifically, we examine the three most used and maintained ROS2 versions according to ROS Metrics [10]: ROS2 Galactic [75], Foxy [72] and Eloquent [71]. We successfully identify four vulnerabilities that exist across all versions of ROS2 and SROS2 implementations. In the rest of this paper, we select ROS2 Foxy [72] distribution, the most mature and widely used ROS2 version as our target, while our findings also apply to the other ROS2 versions. We present the counterexamples, model checking outputs as well as our analysis on the minor differences between ROS2 versions in our supporting material [9].

5.1 V1: Permission File Replacement

The first vulnerability is caused by violations of security requirements **R3** and **R4** in Section 4.3, where a malicious node can bypass the access control policies and publish or subscribe to unauthorized topics. *The root cause of this vulnerability is a ROS2 design flaw, where the adversary can abuse the local privilege to incur synchronization failures of access control policies.*

ROS2 enforces access control policies by passing the SROS2 security files to the DDS security plug-in through APIs (Section 2.4). This requires the access control policies to be updated and synchronized in three layers of the ROS2 architecture. (1) *System policies* are created by the system owner. They are declared in the signed permission files and distributed to the corresponding robots. (2) *SROS2 policies* are loaded by the SROS2 modules. Each robot declares the directory that contains the security files. The SROS2 modules verify the validity of these security files, and then pass them to the DDS layer through API calls. (3) *DDS QoS policies* are loaded by DDS QoS security plug-ins. It enforces access control on the DDS participants and the ROS2 nodes.

Ideally, access control policies in the three layers should be timely synchronized: once the system owner updates the policies during the workload execution, the corresponding security files should be updated on the robots; the policies declared in the security files are then loaded by the SROS2 modules and passed to the DDS participants immediately. However, we discover that an adversary could abuse the design flaw of the SROS2 permission file revocation process to interrupt the synchronization process, thus invalidate the SROS2 access control and further attack the system.

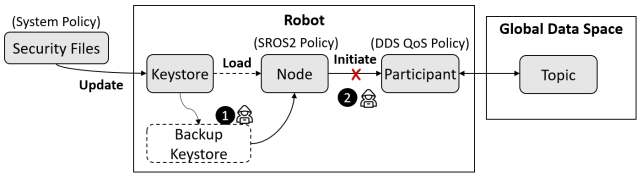


Figure 5: Unauthorized publishing/subscription through the vulnerabilities of V1 (❶) and V2 (❷).

As introduced in Section 2.3, the permission files store the access control policies. When a node publishes or subscribe to a topic, it provides the corresponding permission file stating the proper access to the topic. SROS2 rejects the action if the permission file does not contain a valid digital signature signed by the CA, thus enforces access control policies. However, SROS2 does not actively revoke the old permission files when the access control policies are updated. Instead, it simply replaces the old files with the new ones, or sets up a new directory to store the new files and changes the corresponding load pointers. Since a robot has *read* and *write* accesses to all the local files, an adversarial robot can store the expired permission files in a backup keystore directory, and then pass them to SROS2 instead of the updated one (❶ in Figure 5). These expired files can pass the CA signature verification and are loaded for policy enforcement. By doing so, the adversary can obtain publish and subscription access to some restricted topics, even its permissions have been explicitly denied in the updated files. A direct mitigation towards this vulnerability is active certificate revocation. By revocation of expired certificates and permission files, the adversary cannot bypass the SROS2 verification with the old permission files. ROS2 has taken our suggestion and added documentations on manual certificate revocation methods in ROS2 rolling [77], the feature testing ROS2 version. However, an complete and automated solution is not implemented yet.

5.2 V2: Outdated Node Service

Similar to V1, the second vulnerability also violates **R3** and **R4** in Section 4.3. *The root cause of this vulnerability is also the synchronization failures of access control policies in different SROS2 layers caused by a ROS2 design flaw, where the DDS QoS policies can only be updated during participant initialization.* An adversarial node can leverage the loophole in SROS2 function calls to refuse the update of access control policies on the corresponding participant.

Particularly in ROS2, node publishing and subscription are two independent actions controlled by the robot. When a node publishes or subscribes to a topic, RMW calls the DDS APIs and the SROS2 security files are loaded to the corresponding DDS participant as the QoS policy parameters. The participant then creates a data reader or writer following the QoS policy. When the system owner updates the access control policies of a node, the robot is required to relaunch the node's publishing or subscription service so the new policies can be updated to the DDS. This design is vulnerable because an adversarial robot can refuse to restart the services of its nodes (❷ in Figure 5), so it can continue accessing the topics, which are supposed to be revoked during permission updates.

5.3 V3: Default Mis-configuration

ROS2 provides a GUI plugin `rqt_graph` [70] to visualize the publishing-subscription relations between nodes and topics for the debugging purpose. To protect the recipient privacy [16], SROS2 disables this function and allows the system owner to configure the discoverability of each node and topic. By default, topics, nodes, and the publishers/subscribers to each topic are hidden after enabling SROS2. However, we identify one vulnerability that allows an adversarial robot to obtain sensitive information of other nodes and topics, i.e., violating the requirement **R6** in Section 4.3. *This vulnerability is caused by a default misconfiguration in the SROS2 implementation that contains insecure DDS QoS parameters.*

We find that SROS2 has some default mis-configurations that could cause cross-node information leakage. For instance, in the implementation of the SROS2 settings for RTPS DDS [41], the default option for the message communication is sign without encryption. A signed DDS message does not hide the its publisher/subscriber participants' information, and the adversarial node can read them to infer the network communication topology. This vulnerability was also reported by other developers as CVE-2019-19625 and CVE-2019-19627. The ROS2 community developed patches to fix them [73]. However, they are not merged into the ROS2 mainstream, making the current version still vulnerable.

5.4 V4: Permission File Inference

Similar to V3, this vulnerability can also cause cross-node information leakage, but from the permission files. *Its root cause is the insecure coding practice without the consideration of the principle of least privilege.* While the integrity of an SROS2 permission file is protected by its digital signature, its confidentiality is not guaranteed. ROS2 assumes that each node protects the confidentiality of its own files including the permission files, so all these files are in cleartext. Ideally, creation of the permission files should follow the principle of *least privilege* [34]: every node in the system can only access the topics necessary for its legitimate purpose. Unfortunately, we discover that a majority of permission files in the official ROS2 projects, including the SROS2 sample publisher-subscriber system [74] and the Open Robotics RMF Demos project [62], disobey this principle and contain excessive information. The adversary robot can easily read the sensitive attributes of other nodes directly from its own permission file including their security configurations and topic access. Workloads which follow or adopt these permission file templates from official projects could suffer severe privacy threats.

It is worth noting that this vulnerability is fundamentally different from the previous ones. While V1 to V3 target the underlying communication protocols, V4 originates from the owner-specified permission files. It can be mitigated by carefully defining the permissions with the principle of least privilege. So in the rest of this paper, we do not consider this vulnerability any more.

5.5 Discussion

The severity of these vulnerabilities is reflected in not only the possible consequences, but also their stealthiness. The existing ROS2/SROS2 mechanisms cannot effectively detect attacks from these vulnerabilities. Specifically, (1) in the current ROS2 communication protocol design, messages do not contain publisher information, and topics are typically designed to process homogeneous types of message without the capability and necessity of tracking the message sources. Therefore, when the adversarial robot exploits the unauthorized publishing/subscription vulnerabilities (V1 and V2) to actively send malicious messages, it is difficult to detect such an anomaly. One possible solution is to actively inspect messages in the network layer, log their sender/receiver IP addresses and construct their publishing-subscription relations. By checking this relation against the system communication graph generated by the ROS2 built-in tool `rostopic` [29], we can detect if a robot is sending unauthorized messages. However, as mentioned in Section 2.4, SROS2 prohibits the use of this tool. The system owner cannot enable this tool by sacrificing the privacy. (2) Exploiting V3 is a passive process, and the adversary does not need to actively communicate with other topics. It is also hard to monitor the occurrence of this attack. (3) SROS2 does not provide any logging features. Function calls and communication messages are not accountable, making it difficult to pinpoint the malicious actor after system failures.

6 VULNERABILITY EXPLOITATION

We validate the exploitability of the discovered vulnerabilities with various real-world ROS2 workloads in both simulation environments and physical testbeds. We show that exploiting these vulnerabilities could cause severe consequences, including but not limited to terminating the workloads, crashing the victim robots and damaging the surroundings, and stealing users' private information.

6.1 Simulation Setup

ROS2 Workloads. We select three open-source MRS workloads based on ROS2 from the Robotics Middleware Framework (RMF) project [62], which is developed by Open Robotics [61]. The project demonstrates the usage of heterogeneous robot teams (nine types of robot in total) in 5 real-world environments with the ROS2 platform. In each workload, robots are controlled by the GCS task planner to collaboratively work on different types of tasks. We select three workload environments: airport terminal, clinic world and campus. Details about these environments are available online at [9].

By design, tasks in the three workloads are split into simpler subtasks that can be completed by one robot to increase the overall system efficiency. The GCS allocates tasks by considering the robot status (e.g., location, battery life, etc.) and system goal. Therefore, each robot works on various subtasks during the execution of the workload, and requires different permissions to access different system resources that vary with the task.

Configurations. We deploy the above workloads in the Gazebo simulator [51] and ROS2 Foxy distribution. All workloads are set up by following the default configurations listed in their project sources. We deploy the SROS2 security features to all the workloads based on the threat model in Section 3.

6.2 Simulation Evaluation

It is worth highlighting that each discovered vulnerability is general to affect different ROS2 workloads with different attack consequences. Without loss of generality, we adopt one workload to

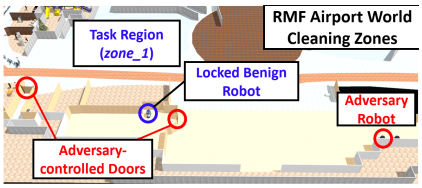


Figure 6: Exploiting V1 to terminate the workload (airport world).

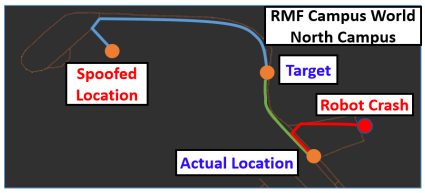


Figure 7: Exploiting V2 to crash the victim robot (campus world).

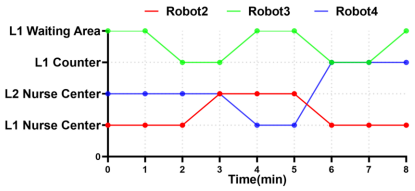


Figure 8: Exploiting V3 to steal the victim robot's states (clinic world).

demonstrate each vulnerability and one possible consequence. Below we describe the exploitation procedures.

V1: Permission File Replacement. As described in Section 5.1, an adversary can pass expired permission files to SROS2 to bypass the access control. Specifically, the adversary can backup the permission files to a local directory which is not accessible to the system owner. After each permission file update, it can replace the latest permission file with any one of the old permission files that contains the permission he needs. In this way, the adversarial robot can bypass the system access control policy, and access the unauthorized topics that was once assigned to it.

We implement a prototype-of-concept attack on the airport terminal workload. We show one possible attack consequence, where the adversarial robot can cause task completion failures by manipulating its access permission to unauthorized environments. As shown in Figure 6, a *CleanerBot* with the task of cleaning the region *zone_1* only has access to the topics related to the resources in this region. When a robot completes this task, the GCS assigns a new task region and updates the permission file so that it only contains access to the topics about the new region, while previous access permissions are revoked at the same time. A robot can exploit **V1** to retain the old permissions and access topics that should only be available to other robots. Our experiment shows that an adversarial *CleanerBot* can eventually obtain the publish/subscribe access to all the topics required by the cleaning tasks, which include the access to control the operation of automatic doors in different cleaning regions as shown in Figure 6. By sending the *close* command to the door control topic, the adversarial robot hinders the movement of other robots and causes workload execution failures.

V2: Outdated Node Service. To retain old permissions, the adversarial robot can also refuse to re-initialize the nodes after the policy update (Section 5.2). We design an attack on the campus workload, where multiple robots deliver items using GPS localization. Each robot streams its location to its corresponding adapter topic so that the GCS can coordinate the overall delivery task accordingly. By exploiting V2, an adversarial robot can retain the publishing access to the previous adapter topic regardless of its current legitimate publishers. It can then send forged GPS data to this topic for the GCS to process. As a result, the task controller will calculate the path based on the spoofed GPS location provided by the adversarial robot as long as it sends fake messages with higher frequency to overwhelms correct messages from the benign robot. Figure 7 shows one possible attack consequence from our simulation experiment: the adversary carefully selects a spoofed location so that the GCS generates a wrong path (blue) and assigns it to the benign robot. The benign robot will follow the trajectory (red) but from its actual location, and crash into the obstacles.

V3: Default Mis-configuration. The adversary can leverage the default mis-configuration in DDS to obtain critical information (Section 5.3). In the default DDS (eProsima Fast DDS), the variable *rtps_protection_kind* defines whether the RTPS message is protected by encryption, which is 'SIGN' by default. Therefore, we can exploit the vulnerability of CVE-2019-19625 [2] with the ROS2 robot fingerprinting tool `Aztarna` [88] to list all nodes and topics. By regularly examining the map resource topics subscribed by each robot, the adversarial robot can record the locations of all other robots and infer their tasks. For example, Figure 8 shows the attack result in the RMF clinic world workload. The adversarial robot captures the location of other robots every minute. Based on such information, it can infer that robot 2 is patrolling between the nurse rooms at level1 and level2; robot 3 is performing guidance tasks between the counter and the waiting area; robot 4 is delivering items between different locations. In real-world scenarios, robot tasks can be closely related to users' personal information. Various works have highlighted that robotic systems (e.g., surgical robots) in hospitals are vulnerable to cyber attacks [26, 35] and have critical privacy issues [81, 83]. Exploiting V3 provides a new attack opportunity to steal personal information in such sensitive scenarios.

6.3 Physical Evaluation

We further validate these vulnerabilities in a physical testbed, which proves it is practical to exploit them to cause severe consequences.

Physical testbed setup. We set up a cloud-based MRS workload from Amazon RoboMaker [14], developed by AWS Robotics [80] and JdeRobot [46]. It considers the operation environment in the Amazon warehouse. We implement this environment with three physical Turtlebot 3 Waffle Pi robots [68] and AWS Elastic Compute (EC2). More details of our physical setups and configurations can be found online at [9].

Evaluation results. Following the attack processes described in Section 5, we implement the exploits to **V1**, **V2** and **V3**, respectively. After gaining unauthorized access to different resources through the exploitation, the adversarial robot can cause various attack outcomes. Here we only demonstrate some possible consequences.

For V1 and V2, we observe that the adversarial robot can directly cause system failures and robot crashes similar to the simulation results in Section 6.2. Particularly, the GCS relies on the real-time position information provided by robots to calculate their trajectories and ensure no collisions during the workload. However, The adversarial robot can easily trick the GCS to design a wrong trajectory by constantly sending spoofed location messages to the topic belonging to other robots. In practice, we observe that the victim robot crashes into walls and other robots when the local obstacle avoidance function is not enabled. When we manually enable this

function, the victim robot just stops functioning because obstacle avoidance contradicts the commands given by the GCS. For V3, we find that it leads to sensitive information leakage similar to the results in Section 6.2. Exploiting V3 allows us to generate the nodes and topics communication topology, which directly reveals the number of robots, current tasks and system control structure.

7 A GENERAL DEFENSE SOLUTION

It is necessary to fix the above threats and make SROS2 really secure. While changing the ROS2 underlying protocol from DDS to other established ones seems to be feasible, it does not address the issues. This is because **V1** to **V3** are rooted in SROS2 design flaws that violate the security considerations in MRS, which are independent of the underlying protocol. There exist straightforward solutions to mitigate each vulnerability individually. For instance, **V1** can be mitigated by updating a node's certificates whenever its access policy is updated; for **V2**, the system owner can enforce all participants to temporarily leave the system and then rejoin during policy update; **V3** can be mitigated by correcting the default misconfigurations. However, these ad-hoc solutions could bring inconvenience for the workload execution and system maintenance. Furthermore, V1 and V2 cannot be fully patched due to the physical limits in the MRS scenarios. The system owner cannot constantly monitor all robots' security configurations and function execution at runtime considering the unstable network in real-world workloads.

7.1 Design Rationale

We aim to design a unified defense solution, which could fundamentally address the identified vulnerabilities in SROS2. The main goal is to *refine the ROS2 communication process to securely and efficiently distribute messages among participants*. Specifically, it should exhibit three properties. (1) Security: the ROS2 access control is expected to be correctly enforced, and the confidentiality of nodes' and topics' information should be strongly preserved. (2) Efficiency: the overhead of the solution should be acceptable in the MRS workload context. (3) Compatibility: the solution can be integrated to ROS2 without any additional infrastructure. Attribute-based encryption (ABE) solutions are mature and widely applied to enforce access control in various types of systems [11, 22, 89] including DDS [48]. However, these primitives are inefficient because they provide fine-grained access control with redundant functionalities in the context of ROS2.

To this end, we introduce a lightweight solution specifically for robotic systems with the private broadcast encryption (PBE) primitive [16] as the underlying technology. Compared with other generalized encryption systems, our method is customized to ROS2 to meet the design requirements so that it is very efficient and fully compatible with ROS2 without modifying its underlying source code. If the ROS2 system is not equipped with SROS2, our method can provide the same mandatory access control. If SROS2 is enabled, our method can prevent all the identified vulnerabilities in Section 5. Our solution can defeat a stronger threat model than the one in Section 3: it can protect the system even if there exist multiple adversarial robots that collude and exchange information with each other. We justify the security of our solution through rigorous proof, formal verification and physical experiments (Section 7.4).

7.2 Methodology Description

We incorporate the PBE scheme into the ROS2 communication system. This scheme uses public key encryption with key indistinguishability under the chosen-ciphertext attacks (IK-CCA) [37] to encrypt the ciphertext component for each recipient. It then generates a random signature and verification key for a one-time, strongly unforgeable signature scheme. It includes the verification key in each public key encryption and then signs the entire ciphertext with the signing key. To be precise, let G be a group with g as the generator, where the computational Diffie-Hellman problem (CDH) [36] is hard but the decisional Diffie-Hellman problem (DDH)[21] is easy[3]. H is a hash function mapping $H : G \rightarrow \{0,1\}^{\lambda}$ for a security parameter λ modeled as a random oracle. Hence, given a strongly correct IK-CCA public key encryption scheme (**Int**, **Keygen**, **Enc**, **Dec**), a strongly existentially unforgeable signature scheme (**SigGen**, **Sig**, **Ver**), and a pair of semantically secure symmetric key encryption and decryption algorithms (**E**, **D**), the PBE system can be described as follows.

1. **Setup**(λ): Run $I \leftarrow \mathbf{Int}(\lambda)$ to get the global parameter I.
2. **Keygen**(I): Given I, generate $(pk_i, sk_i) \leftarrow \mathbf{Gen}(I)$ for each node $i \in N$. Also generate key pairs $(vk_i, vsk_i) \leftarrow \mathbf{SigGen}(I)$ for each node $i \in N$ for signature and verification processes. Then, choose a random exponent α_i and let $pk'_i = (pk_i, g^{\alpha_i})$, $sk'_i = (sk_i, \alpha_i)$. (pk'_i, sk'_i) and (vk_i, vsk_i) are sent to node i and publish pk'_i.
3. **Encrypt**(P, m): Consider that node k with signature key and verification key (vk_k, vsk_k) wants to send a message m to nodes in a selected subset $P \subset N$. Node k runs the following procedures:
   3.1. Randomly choose a one-time symmetric key K used to encrypt m.
   3.2. Randomly select a one-time exponent t and set $W = g^t$.
   3.3. For every node $i \in P$, compute
   $$c_{pk_i} \leftarrow \mathbf{H}(g^{t\alpha_i}) \| \mathbf{Enc}_{pk_i}(vk_k \| g^{t\alpha_i} \| K)$$
   3.4. Let C_1 be the concatenation of the c_{pk_i} ordered by their values of $\mathbf{H}(g^{t\alpha_i})$.
   3.5. Encrypt m as $C_2 \leftarrow \mathbf{E}_K(m)$.
   3.6. Generate the signature for the above ciphertext as $\mu \leftarrow \mathbf{Sig}_{vsk_k}(W \| C_1 \| C_2)$.
   3.7. Broadcast ciphertext $C = \mu \| W \| C_1 \| C_2$ to all nodes.
4. **Decrypt**$((sk_j, \alpha_j)_{j \in N}, C)$: Each node $j \in N$, parse $C = \mu \| W \| C_1 \| C_2$ and $C_1 = c_1 \| ... \| c_p$, then run the following procedures:
   4.1. Calculate $r = \mathbf{H}(W^{\alpha_j}) = \mathbf{H}(g^{\alpha_j t})$.
   4.2. Find c_l such that $c_l = r \| c$. If it does not exist, return $\perp$ and stop.
   4.3. Compute $d \leftarrow \mathbf{Dec}(sk_j, c)$. If d is $\perp$, return $\perp$ and stop. Otherwise, parse d as $vk_k \| u \| K$.
   4.4. If $u \neq W^{\alpha_j}$, return $\perp$ and stop.
   4.5. If $\mathbf{Ver}_{vk_k}(W \| C_1 \| C_2, \mu)$, return $m = \mathbf{D}_K(C_2)$; otherwise, return $\perp$.

The above scheme can be adopted in ROS2 with the following steps.

1. The CA generates pairs of certificates and private keys for nodes in the ROS2 system with **Gen**. Then the system owner updates the certificate/key pairs to each node.

[3]For formal definitions of these NP problems, please refer to [16].

2. The system owner formulates the access control policies and updates them to all nodes. It then passes access control knowledge to nodes accordingly. Each node knows the topics to publish/subscribe to. A node with publishing (resp., subscription) access is provided with the public (resp., verification) keys of its subscribers (resp., publishers) N.
3. When a node publishes a message m to a selected subset groups $P \subset N$, it encrypts the message with the public keys of the recipient nodes in P following the encryption function **Encrypt** described above and publishes the ciphertext C to the topic. While all the nodes in N can subscribe to the topic, only the receiver nodes in P have proper read access and can extract m from C using the decryption function **Decrypt**.
4. After a node extracts m through the decryption function, it examines if the verification key vk obtained from the decryption process is in the verification key list provided by the system owner. Otherwise it discards the message m because it comes from an untrusted node.
5. When the access control policies need to be updated, the system owner updates the new access control knowledge to the related nodes by encrypting the knowledge and publishing it to the nodes accordingly following step **3**.
6. When a new node is introduced into the system after initialization, the CA generates key pairs and the system owner updates them to the node accordingly. The system owner then broadcasts the public key of the new nodes together with the updated access control policies to the existing nodes following step **3**. The same process applies to the node revocation scenario.

7.3 Implementation

We implement the proposed defense as a lightweight Python3 package [9]. The system owner can set up our defense in an existing ROS2 workload with three steps. First, CA generates the public and private key pairs required by the selected Elliptic Curve Cryptography (ECC) scheme for all nodes in the system. This process is the same as certificate/private key generation process required by SROS2, so it is supported by the SROS2 command line tool without additional infrastructure for implementation. Second, the system owner installs the defense scheme. The encryption and decryption functions can be easily imported from the Python3 package. Third, each robot encrypts the message with the public keys of the intended receivers before sending it out. The receiver robots can decrypt ciphertext messages as long as they are in the receivers list.

7.4 Security Evaluation

We perform the security assessment of our defense in three aspect.

7.4.1 Theoretical Analysis. Given the strongly correct IK-CCA public key encryption scheme (**Int**, **Keygen**, **Enc**, **Dec**), a strongly existentially unforgeable signature scheme (**SigGen**, **Sig**, **Ver**), and a pair of semantically secure symmetric key encryption and decryption algorithms (**E**, **D**), the aforementioned PBE system has been proven to be secure under chosen-ciphertext attacks. We describe how the PBE holds the requirements in Section 4.3. Specifically, considering that node k wants to send a message m to nodes in a selected subset $P \subset N$, we have the following two theorems, where the first one is used to fix vulnerabilities V1 and V2, while the second one is used to fix V3. The complete proofs are available in our supporting materials [9].

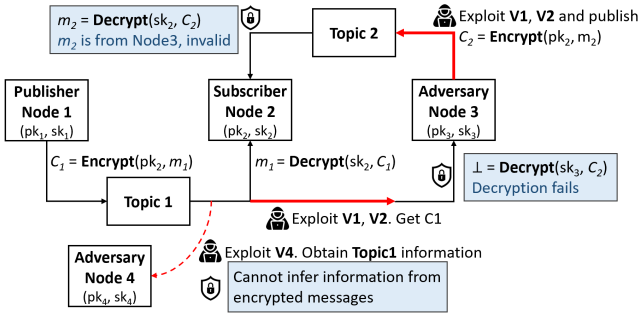


Figure 9: Mitigating vulnerabilities with our defense.

Theorem 1. *If node k is malicious, the above PBE system holds that node k cannot pretend to be other honest users to send ciphertext.*

Theorem 2. *If node k is benign, for any adversary $\mathcal{A}$, the above PBE system holds that $\mathcal{A}$ cannot infer the identity of benign nodes in P. Particularly, if there is no malicious node in P, $\mathcal{A}$ cannot obtain useful information about message m.*

Remark: The proposed defense scheme can protect the ROS2-based MRS against the identified vulnerabilities, as shown in Figure 9. For V1 and V2, the adversarial node can bypass the SROS2 access control and publish to unauthorized topics. However according to Theorem 1, PBE prevents it from pretending to be honest users for sending ciphertext. Thus, the receiver nodes could identify that the publisher is malicious. Similarly, for an adversarial node that exploits V1 and V2 to subscribe to unauthorized topics, it cannot decrypt the messages received from the topics and obtain any useful information according to Theorem 2. For V3, the adversarial node observes the network traffic and infers the secret information. With our defense, a node can subscribe to any topics yet only retrieve useful information from the authorized ones. Thus, the network traffics between any two nodes do not necessarily mean that they are exchanging valid information. So the adversary cannot gather sensitive information with V3. With these properties, our solution provides recipient privacy and the same mandatory access control as ROS2, so it can be implemented individually or on top of ROS2.

7.4.2 Formal Verification. We formally verify the security of the proposed scheme. First, we construct a formal model for the encryption scheme and perform formal verification with ProVerif [19]. No protocol weaknesses are identified when verifying the model against the security requirements. We then follow the model checking approach in Section 4 to verify the security of its integration in ROS2. Specifically, we identify the events that should be performed by the system actors as defined in Section 7.2. We then abstract them and describe them with CSP#, and extend the previously constructed CSP# model in Section 4.2 to describe the ROS2 system with the proposed defense solution. By verifying the new model with PAT, we confirm that the identified vulnerabilities have been fixed with no additional counterexamples generated.

7.4.3 Empirical Validation. We repeat the physical attacks launched in Section 6.3 with the same setups. After enabling our defense scheme, we observe that all three identified vulnerabilities are no longer exploitable. Specifically, for V1 and V2, our defense provides

the authentication service between the message sender and receiver. So even the adversarial robot can retain the old access permissions by replacing the permission files or refusing to restart the node service, it is still not able to access the unauthorized topics. For V3, our defense scheme prevents the adversarial robot from distinguishing valid communications from the invalid ones when monitoring the network traffic. Therefore, the adversary is not able to infer the communication topology using any fingerprint tools. It is worth noting that the proposed solution cannot defeat DoS attacks. In fact, how to design full defenses against DoS attacks from insiders in robotic systems is still an open problem [17, 91], because the adversary can use system knowledge to craft DoS messages that follow the protocol. Nevertheless, our design is less vulnerable to DoS attacks compared to other cryptographic solutions. Specifically, in step **4** of the scheme (Section 7.2), a node does not perform any decryptions if the received message is from an outsider (**4.1**) , and only performs partial decryption (**4.3**) if the message is from an insider adversary, which increases the DoS difficulty.

7.5 Efficiency Evaluation

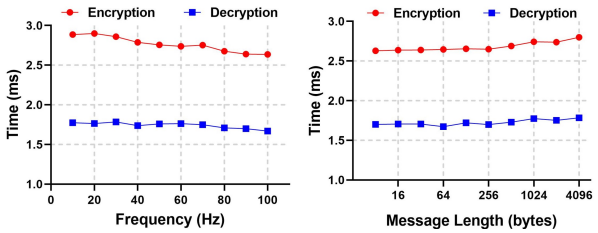
We evaluate the performance and resource consumption of our solution using the physical testbed. Below we present the main experimental results, while the physical experiment setups and experimental data are available in our project website [9].

7.5.1 Performance Evaluation. We first measure the impact of the additional operations (e.g., encryption, decryption) on the performance of the MRS. We adopt the mainstream *ROS2 Performance Test* benchmark [4] developed by ApexAI [6]. and make necessary modificationsto adapt to our defense scheme. We deploy two Turtlebot robots to run the publisher node and subscriber node respectively, connected to the same local area network.

We compare four settings. (1) *Normal*: ROS2 without any security features; (2) *SROS2*: ROS2 with SROS2 enabled. (3) *PBE*: ROS2 with the proposed defense; (4) *Both*: ROS2 with both SROS2 and proposed defense. For each experiment, we execute the task for 60 seconds, and repeat it for 10 times to obtain the average results.

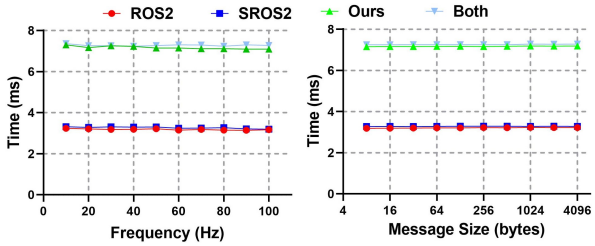
Evaluation results. First, we explore the average encryption and decryption cost of our defense for different message sizes and publishing frequencies. We vary the publishing frequency from 10 Hz to 100 Hz, and the cleartext length from 8 Bytes to 4096 Bytes, covering the common configurations in most robotic system components. The encryption and decryption overhead of the PBE scheme is shown in Figure 10. We observe that the cost of those operations is slightly increased with the message length: the average encryption/decryption time of a 4KB message is 6.4%/4.9% longer than a 8B message. We also observe the cost is slightly decreased with a higher publishing frequency. This might be due to the CPU Dynamic Voltage and Frequency Scaling (DVFS) optimization feature.

Second, we compare the end-to-end latency for the entire system with four security settings. We also select the above ranges of message sizes and publishing frequencies. The results are shown in Figure 11. Our solution introduces around 4*ms* latency for each communication. Compared with Figure 10, such cost is mainly from the encryption/decryption operations. In real-world robotic systems (especially the cloud-based), the network latency is much higher (in the order of seconds). Therefore an MRS is commonly



(a) Impact of frequency (b) Impact of message size

Figure 10: Encryption/Decryption time cost under different frequencies and message sizes



(a) Impact of frequency (b) Impact of message size

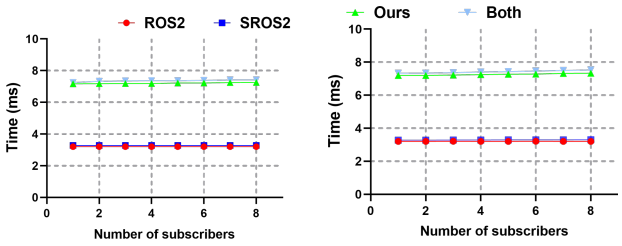
Figure 11: Communication latency of four implementations with various message lengths and frequencies

designed to be delay-tolerant [1], and this overhead can be ignored. We further measure the data loss rate during transmission, and observe that our scheme causes less than 0.01% of data loss at 100Hz frequency with the message sizes of 1KB and 4KB. It happens during communication initialization, when the first few packets are not delivered to the subscriber. This "initial loss" is also observed by other works [55], and does not affect the system operation.

Third, we explore the scalability of the proposed defense. We examine the system latency in two experiment settings: (1) one publisher publishes to multiple nodes; (2) a number of nodes connected in series, where the intermediate nodes act as both publishers and subscribers. For each scenario, we vary the number of subscription nodes from 1 to 8 and message sizes of 64 and 4096 bytes. The publishing frequency is fixed at 20 Hz. The communication latency of different system configurations is shown in Figures 12 and 13. We observe that the end-to-end latency does not increase significantly when more nodes subscribe to one publisher. When nodes are connected in sequence, the latency increases linearly with the number of communication nodes. In practice, an intermediate node needs to process the incoming message or control the actuators to operate accordingly before transmitting the message to the next one. This process can compensate the overhead incurred by our solution. Overall, our defense does not incur significant latency compared to SROS2, and can easily scale to large systems.

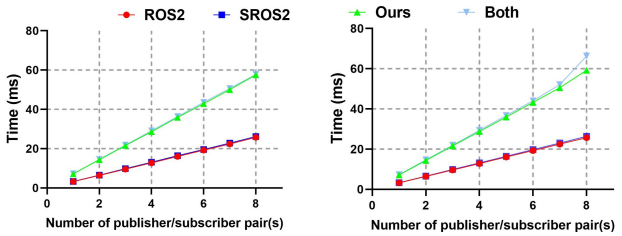
7.5.2 Resource Consumption Evaluation. We measure the resource consumption in our defense, which is critical for robots with limited computing capability. We select and implement two most widely adopted MRS workloads: navigation [47] and exploration [65].

Evaluation results. We measure the runtime CPU and RAM utilization of the on-board processors on the Turtlebots, as shown in Figure 14. For the CPU usage (Figure 14(a)), the navigation and exploration workloads require 42.5% and 45.2% of CPU resources,



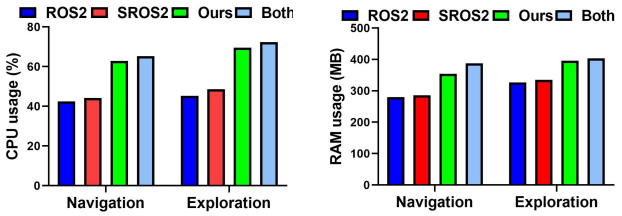
(a) 64-byte payload (b) 4096-byte payload

Figure 12: Communication latency of four implementations with one publisher and various numbers of subscribers



(a) 64-byte payload (b) 4096-byte payload

Figure 13: Communication latency of four implementations with various numbers of publisher-subscriber pairs



(a) CPU utilization (b) RAM utilization

Figure 14: Resource consumption of four implementations.

respectively, and enabling SROS2 does not increase the CPU utilization significantly. With the proposed defense, the CPU utilization of these two workloads are increased to 62.9% and 72.3%, respectively. Such overhead is acceptable since the CPU cores are still not saturated. In real-world workloads, on-robot processors are under-utilized most of the time [95] because they should meet the performance requirement of the most computational extensive subtask, which only takes very little operation time. Our solution only takes the redundant computational power during the workload execution. Also, we believe the CPU utilization can be further optimized by migrating the current Python implementation to C++, which is also supported by ROS2. For RAM utilization (Figure 14(b)), the two workloads require 279.6 MB and 326.9 MB of memory. The defense increases the RAM consumption to 354.0 MB and 396.2 MB. This is far lower than the capacity of common robotic processors (e.g. 1GB RAM for the Raspberry Pi 3B+ model in our experiments).

Based on the above results, we conclude that the CPU and RAM utilization of our defense is acceptable on commercial robots with single-board processors. Note that it might cause performance issues when we implement this scheme on tiny robots with very limited computing resources (e.g., swarm robots). In the future, we will further optimize our implementation for these scenarios.

8 RELATED WORKS

Model checking. Model checking has been widely adopted to verify the correctness and security of systems [32, 33, 60, 66]. Recently, researchers applied this strategy to verify robotic and autonomous applications, such as DoS vulnerabilities in connected vehicle protocols [44], safety properties of ROS-based robotic applications [20], hierarchical properties of swarm robot systems [43], security, liveness and priority of the DDS without considering the ROS2 implementation [54]. Different from the works which focus on either applications or individual components of the ROS/ROS2 system, we mainly target the fundamental implementations of the ROS2 security features.

Access control with cryptography. Barth *et al.* [16] proposed the first private broadcast encryption scheme to achieve identity-based access control among messages. Then, many variants (e.g., attribute-based encryption (ABE), puncturable encryption) were proposed to achieve more precise access control with the attribute of the system participants [18, 39, 52, 78]. For instance, Bethencourt *et al.* [18] developed a ciphertext-policy based ABE that allows tree-based access policies. Yu *et al.* [94] proposed a solution for indirect attribute and user revocation.

ROS2 and DDS Security. Previous works including [90] explore the efficient and automatic generation of SROS2 permission files for ROS2 projects. Other work [54] formally verify the security of DDS in ROS2. These works leverage existing SROS2 features and do not consider that adversaries could bypass SROS2 through its native vulnerabilities. Instead, we propose the first study over the security of ROS2 implementation. The vulnerabilities discussed in this work thus cannot be identified or patched by the previous solutions.

9 CONCLUSION

In this paper, we perform a thorough and systematic security analysis about ROS2 with the DDS security features. We design a formal method to model the ROS2 system and security requirements. We identify four vulnerabilities in the implementation of ROS2, which can invalidate the security mechanism of DDS, and threaten the robotic workloads. To fundamentally address these issues, we design a practical and lightweight defense methodology with the private broadcast encryption. We have reported our discoveries to the ROS2 official and are working with them on possible mitigation. We hope these vulnerabilities can be fixed very soon to advance the secure development of robotic systems and applications.

ACKNOWLEDGMENTS

We thank the anonymous reviewers for their constructive feedback. We also thank Dr. Chaoshun Zuo for his comments and generous help to this work. This work was supported in part by Ministry of Education, Singapore under its Academic Research Fund (AcRF) Tier 1 (RG108/19 (S)), Tier 2 (MOE-T2EP20120-0004) and Tier 3 (MOET32020-0004), NTU-DESAY SV Research Program under Grant 2018-0980, Singapore National Research Foundation (NRF) under its National Cybersecurity R&D Program (NRF2018NCR-NCR005-0001).

REFERENCES

[1] 2009. Supporting Navigation in Multi-Robot Systems through Delay Tolerant Network Communication. *IFAC Proceedings Volumes* 42, 22 (2009), 25–30. 1st IFAC Workshop on Networked Robotics.

[2] 2019. CVE-2019-19625 Detail. https://nvd.nist.gov/vuln/detail/CVE-2019-19625.

[3] 2020. ROS2 Node to Participant Mapping. https://design.ros2.org/articles/Node_to_Participant_mapping.html

[4] 2021. ApexAI ROS2 Performance Test [Online]. https://gitlab.com/ApexAI/performance_test.

[5] 2021. ROS 2 robotic systems threat model. https://design.ros2.org/articles/ros2_threat_model.html.

[6] 2022. Apex AI: The Vehicle OS Company. https://www.apex.ai/.

[7] 2022. Cloud Robotics Core: Kubernetes, Federation, App Management [Online]. https://googlecloudrobotics.github.io/core/.

[8] 2022. IRobot: Robot Vacuum and Mop. https://www.irobot.com/

[9] 2022. On the (In)Security of Secure ROS2. https://sites.google.com/view/secure-sros2.

[10] 2022. ROS Metrics. https://metrics.ros.org/rosdistro_rosdistro.html

[11] Shashank Agrawal and Melissa Chase. 2017. FAME: Fast Attribute-Based Message Encryption. In *Proceedings of the 2017 ACM SIGSAC Conference on Computer and Communications Security* (Dallas, Texas, USA) *(CCS '17)*. Association for Computing Machinery, New York, NY, USA, 665–682.

[12] Aliasrobotics. 2020. Robot Vulnerability Database (RVD) [Online]. https://github.com/aliasrobotics/RVD/.

[13] Krzysztof R. Apt and Dexter Kozen. 1986. Limits for automatic verification of finite-state concurrent systems. *Inf. Process. Lett.* 22, 6 (1986), 307–309.

[14] AWS-Robotics. 2021. Aws-Robotics/AWS RoboMaker Small Warehouse World. https://github.com/aws-robotics/aws-robomaker-small-warehouse-world

[15] Agata Barciś, Michał Barciś, and Christian Bettstetter. 2019. Robots that Sync and Swarm: A Proof of Concept in ROS 2. In *2019 International Symposium on Multi-Robot and Multi-Agent Systems (MRS)*. 98–104.

[16] Adam Barth, Dan Boneh, and Brent Waters. 2006. Privacy in Encrypted Content Distribution Using Private Broadcast Encryption. In *Financial Cryptography and Data Security*, Giovanni Di Crescenzo and Avi Rubin (Eds.). Springer Berlin Heidelberg, Berlin, Heidelberg, 52–64.

[17] Elena Basan, Mikhail Medvedev, and Stanislav Teterevyatnikov. 2018. Analysis of the Impact of Denial of Service Attacks on the Group of Robots. In *2018 International Conference on Cyber-Enabled Distributed Computing and Knowledge Discovery (CyberC)*. 63–638.

[18] John Bethencourt, Amit Sahai, and Brent Waters. 2007. Ciphertext-Policy Attribute-Based Encryption. In *2007 IEEE Symposium on Security and Privacy (SP '07)*. 321–334.

[19] B. Blanchet. 2001. An efficient cryptographic protocol verifier based on prolog rules. In *Proceedings. 14th IEEE Computer Security Foundations Workshop, 2001*. 82–96.

[20] Renato Carvalho, Alcino Cunha, Nuno Macedo, and André Santos. 2020. Verification of system-wide safety properties of ROS applications. In *2020 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. 7249–7254.

[21] Wouter Castryck, Jana Sotáková, and Frederik Vercauteren. 2020. Breaking the decisional Diffie-Hellman problem for class group actions using genus theory. In *Annual International Cryptology Conference*. Springer, 92–120.

[22] Jie Chen, Romain Gay, and Hoeteck Wee. 2015. Improved Dual System ABE in Prime-Order Groups via Predicate Encodings. In *Advances in Cryptology - EUROCRYPT 2015*, Elisabeth Oswald and Marc Fischlin (Eds.). Springer Berlin Heidelberg, Berlin, Heidelberg, 595–624.

[23] Peter Chen, Marjon Dean, Don Ojoko-Adams, Hassan Osman, Lilian Lopez, Nick Xie, and Nancy Mead. 2004. System Quality Requirements Engineering (SQUARE) Methodology: Case Study on Asset Management System. (12 2004), 326.

[24] Qi Alfred Chen, Yucheng Yin, Yiheng Feng, Z Morley Mao, and Henry X Liu. 2018. Exposing Congestion Attack on Emerging Connected Vehicle based Traffic Signal Control.. In *25th Annual Network and Distributed System Security Symposium (NDSS)*.

[25] Kun Cheng, Yuan Zhou, Bihuan Chen, Rui Wang, Yuebin Bai, and Yang Liu. 2020. Guardauto: A decentralized runtime protection system for autonomous driving. *IEEE Trans. Comput.* 70, 10 (2020), 1569–1581.

[26] Key-whan 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, Chaoyang District, Beijing, China, September 23-25, 2019*.

[27] Edmund M. Clarke, William Klieber, Miloš Nováček, and Paolo Zuliani. 2012. *Model Checking and the State Explosion Problem*. Springer Berlin Heidelberg, 1–30.

[28] Edmund M Clarke Jr, Orna Grumberg, Daniel Kroening, Doron Peled, and Helmut Veith. 2018. *Model checking*.

[29] Ken Conley. 2011. ROS command line tool: rostopic. http://library.isr.ist.utl.pt/docs/roswiki/rostopic.html.

[30] DDS Foundation. 2020. Data Distribution Services. https://www.dds-foundation.org/.

[31] 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*. 17–32.

[32] Zuohua Ding, Yuan Zhou, Mingyue Jiang, and MengChu Zhou. 2014. A new class of Petri nets for modeling and property verification of switched stochastic systems. *IEEE Transactions on Systems, Man, and Cybernetics: Systems* 45, 7 (2014), 1087–1100.

[33] Zuohua Ding, Yuan Zhou, and MengChu Zhou. 2014. Stability analysis of switched fuzzy systems via model checking. *IEEE Transactions on Fuzzy Systems* 22, 6 (2014), 1503–1514.

[34] Murph Finnicum and Samuel T King. 2011. Building Secure Robot Applications.. In *6th USENIX Workshop on Hot Topics in Security, HotSec'11*.

[35] Eduard Fosch-Villaronga and Tobias Mahler. 2021. Cybersecurity, safety and robots: Strengthening the link between cybersecurity and safety in the context of care robots. *Computer Law & Security Review* 41 (2021), 105528.

[36] Sanjam Garg and Mohammad Hajiabadi. 2018. Trapdoor functions from the computational Diffie-Hellman assumption. In *Annual International Cryptology Conference*. Springer, 362–391.

[37] Romain Gay. 2020. A New Paradigm for Public-Key Functional Encryption for Degree-2 Polynomials.. In *Public Key Cryptography (1)*. 95–120.

[38] Shreyas Gokhale. 2020. JdeMultiBot: Multi-Robot exercises for Robotics Academy In ROS2. (2020).

[39] Vipul Goyal, Abhishek Jain, Omkant Pandey, and Amit Sahai. 2008. Bounded Ciphertext Policy Attribute Based Encryption. In *Automata, Languages and Programming*, Luca Aceto, Ivan Damgård, Leslie Ann Goldberg, Magnús M. Halldórsson, Anna Ingólfsdóttir, and Igor Walukiewicz (Eds.). Springer Berlin Heidelberg, Berlin, Heidelberg, 579–591.

[40] Object Management Group. 2018. DDS security [Online]. https://www.omg.org/spec/DDS-SECURITY/1.1/PDF.

[41] Object Management Group. 2018. The real-time publish-subscribe protocol (RTPS) DDS - OMG [Online]. https://www.omg.org/spec/DDSI-RTPS/2.3/Beta1/PDF.

[42] Charles Antony Richard Hoare. 1978. Communicating sequential processes. *Commun. ACM* 21, 8 (1978), 666–677.

[43] Chi Hu, Wei Dong, Yonghui Yang, Hao Shi, and Ge Zhou. 2019. Runtime verification on hierarchical properties of ROS-based robot swarms. *IEEE Transactions on Reliability* 69, 2 (2019), 674–689.

[44] Shengtuo Hu, Qi Alfred Chen, Jiachen Sun, Yiheng Feng, Z. Morley Mao, and Henry X. Liu. 2021. Automated Discovery of Denial-of-Service Vulnerabilities in Connected Vehicle Protocols. In *30th USENIX Security Symposium (USENIX Security 21)*. 3219–3236.

[45] Syed Hussain, Omar Chowdhury, Shagufta Mehnaz, and Elisa Bertino. 2018. LTEInspector: A systematic approach for adversarial testing of 4G LTE. In *Network and Distributed Systems Security (NDSS) Symposium 2018*.

[46] JdeRobot. 2020. JdeRobot: Open toolkit for developing Robotics applications. https://jderobot.github.io/.

[47] 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.

[48] Hwimin Kim, Dae-Kyoo Kim, and Alaa Alaerjan. 2021. ABAC-Based Security Model for DDS. *IEEE Transactions on Dependable and Secure Computing* (2021).

[49] Hyungsub Kim, Muslum Ozgur Ozmen, Antonio Bianchi, Z. Berkay Celik, and Dongyan Xu. 2021. PGFUZZ: Policy-Guided Fuzzing for Robotic Vehicles. In *28th Annual Network and Distributed System Security Symposium, NDSS 2021, virtually, February 21-25, 2021*. The Internet Society.

[50] Misook Kim, SangGyu Kim, ByoungYoul Song, Young-sook Jeong, and Hong Seong Park. 2021. Study on Requirements of Cloud-based Environments for Easy Development of ROS Modules. In *2021 18th International Conference on Ubiquitous Robots (UR)*. 48–51.

[51] N. Koenig and A. 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 vol.3.

[52] Allison Lewko, Tatsuaki Okamoto, Amit Sahai, Katsuyuki Takashima, and Brent Waters. 2010. Fully Secure Functional Encryption: Attribute-Based Encryption and (Hierarchical) Inner Product Encryption. In *Advances in Cryptology – EUROCRYPT 2010*, Henri Gilbert (Ed.). Springer Berlin Heidelberg, Berlin, Heidelberg, 62–91.

[53] Hakan Lindqvist. 2006. Mandatory access control. *Master's thesis in computing science, Umea University, Department of Computing Science, SE-901* 87 (2006).

[54] Yanan Liu, Yong Guan, Xiaojuan Li, Rui Wang, and Jie Zhang. 2018. Formal analysis and verification of DDS in ROS2. In *2018 16th ACM/IEEE International Conference on Formal Methods and Models for System Design (MEMOCODE)*. 1–5.

[55] Yuya Maruyama, S. Kato, and Takuya Azumi. 2016. Exploring the performance of ROS2. *2016 International Conference on Embedded Software (EMSOFT)* (2016), 1–10.

[56] Petra Mazdin, Michal Barciś, Hermann Hellwagner, and Bernhard Rinner. 2020. Distributed Task Assignment in Multi-Robot Systems based on Information

Utility. In *2020 IEEE 16th International Conference on Automation Science and Engineering (CASE)*. 734–740.
[57] Jarrod McClean, Christopher Stull, Charles Farrar, and David Mascareñas. 2013. A preliminary cyber-physical security assessment of the Robot Operating System (ROS). In *Unmanned Systems Technology XV*, Robert E. Karlsen, Douglas W. Gage, Charles M. Shoemaker, and Grant R. Gerhart (Eds.), Vol. 8741. International Society for Optics and Photonics, SPIE, 341 – 348.
[58] Jarrod R. Mcclean and Charles Farrar. 2013. A Preliminary Cyber-Physical Security Assessment of the Robot Operating System (ROS). In *Proceedings of SPIE*.
[59] Michael Myers, Rich Ankney, Ambarish Malpani, Slava Galperin, and Carlisle Adams. 1999. X. 509 Internet public key infrastructure online certificate status protocol-OCSP. (1999).
[60] Dang Tu Nguyen, Chengyu Song, Zhiyun Qian, Srikanth V Krishnamurthy, Edward JM Colbert, and Patrick McDaniel. 2018. IotSan: Fortifying the safety of IoT systems. In *Proceedings of the 14th International Conference on emerging Networking EXperiments and Technologies*. 191–203.
[61] Open Source Robotics Foundation. 2021. Open Robotics. https://www.openrobotics.org/.
[62] OpenRMF. 2022. Open-RMF/RMF Demos: Demonstrations of The OpenRMF software [Online]. https://github.com/open-rmf/rmf_demos
[63] Gerardo Pardo-Castellote, Bert Farabaugh, and Rick Warren. 2005. An introduction to DDS and data-centric communications. *Real-Time Innovations* (2005).
[64] Amir Pnueli. 1977. The temporal logic of programs. In *18th Annual Symposium on Foundations of Computer Science (sfcs 1977)*. 46–57.
[65] 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.
[66] Ronald W Ritchey and Paul Ammann. 2000. Using model checking to analyze network vulnerabilities. In *Proceeding 2000 IEEE Symposium on Security and Privacy. S&P 2000*. IEEE, 156–165.
[67] Sean Rivera and Radu State. 2021. Securing Robots: An Integrated Approach for Security Challenges and Monitoring for the Robotic Operating System (ROS). In *2021 IFIP/IEEE International Symposium on Integrated Network Management (IM)*. IEEE, 754–759.
[68] Robotis. 2021. TurtleBot 3. https://www.turtlebot.com/.
[69] ROS. 2020. ROS Index package list. https://index.ros.org/packages/.
[70] Ros-Visualization. 2020. Ros-visualization/RQT_GRAPH. https://github.com/ros-visualization/rqt_graph.
[71] ROS2. 2019. ROS 2 Foxy Elusor. https://docs.ros.org/en/eloquent/index.html.
[72] ROS2. 2020. ROS 2 Foxy Fitzroy. https://docs.ros.org/en/foxy/Releases/Release-Foxy-Fitzroy.html.
[73] ROS2. 2020. Set rtps_protection_kind to encrypt by default pull request #171 · ROS2/SROS2. https://github.com/ros2/sros2/pull/171.
[74] ROS2. 2020. SROS2 Project Sample Policies. https://github.com/ros2/sros2/tree/master/sros2/sros2/policy.
[75] ROS2. 2021. ROS 2 Galatic Geochelone. https://docs.ros.org/en/galactic/index.html.
[76] ROS2. 2021. SROS2. https://github.com/ros2/sros2.
[77] ROS2. 2022. ROS 2 Rolling. https://docs.ros.org/en/rolling/index.html.
[78] Amit Sahai and Brent Waters. 2005. Fuzzy Identity-Based Encryption. In *Advances in Cryptology – EUROCRYPT 2005*, Ronald Cramer (Ed.). 457–473.
[79] Joseph Salowey, Abhijit Choudhury, and David McGrew. 2008. AES Galois Counter Mode (GCM) cipher suites for TLS. *Request for Comments* 5288 (2008).
[80] Amazon Web Services. 2021. AWS Robotics. https://aws.amazon.com/robomaker/.
[81] Ryan Shah and Shishir Nagaraja. 2019. Privacy with Surgical Robotics: Challenges in Applying Contextual Privacy Theory. *CoRR* abs/1909.01862 (2019). arXiv:1909.01862 http://arxiv.org/abs/1909.01862
[82] Alireza Souri, Amir Masoud Rahmani, Nima Jafari Navimipour, and Reza Rezaei. 2019. A symbolic model checking approach in formal verification of Distributed Systems. *Human-centric Computing and Information Sciences* 9, 1 (2019). https://doi.org/10.1186/s13673-019-0165-x
[83] Bernd Carsten Stahl and Mark Coeckelbergh. 2016. Ethics of healthcare robotics: Towards responsible research and innovation. *Robotics and Autonomous Systems* 86 (2016), 152–161.
[84] Jun Sun, Yang Liu, and Jin Song Dong. 2008. Model Checking CSP Revisited: Introducing a Process Analysis Toolkit. In *Leveraging Applications of Formal Methods, Verification and Validation*, Tiziana Margaria and Bernhard Steffen (Eds.). Springer Berlin Heidelberg, Berlin, Heidelberg, 307–322.
[85] Jun Sun, Yang Liu, Jin Song Dong, and Chunqing Chen. 2009. Integrating specification and programs for system modeling and verification. In *2009 Third IEEE International Symposium on Theoretical Aspects of Software Engineering*. 127–135.
[86] Rafael R. Teixeira, Igor P. Maurell, and Paulo L.J. Drews. 2020. Security on ROS: analyzing and exploiting vulnerabilities of ROS-based systems. In *2020 Latin American Robotics Symposium (LARS), 2020 Brazilian Symposium on Robotics (SBR) and 2020 Workshop on Robotics in Education (WRE)*. 1–6.
[87] Andrea Testa, Andrea Camisa, and Giuseppe Notarstefano. 2021. ChoiRbot: A ROS 2 Toolbox for Cooperative Robotics. *IEEE Robotics and Automation Letters* 6, 2 (2021), 2714–2720.
[88] Víctor Vilches, Gorka Olalde, Xabier Baskaran, Alejandro Cordero, Lander Juan, Endika Gil-Uriarte, Odei Urabain, and Laura Kirschgens. 2018. Aztarna, a footprinting tool for robots.
[89] Brent Waters. 2011. Ciphertext-Policy Attribute-Based Encryption: An Expressive, Efficient, and Provably Secure Realization. In *Public Key Cryptography – PKC 2011*, Dario Catalano, Nelly Fazio, Rosario Gennaro, and Antonio Nicolosi (Eds.). Springer Berlin Heidelberg, Berlin, Heidelberg, 53–70.
[90] Ruffin White, Henrik I. Christensen, Gianluca Caiazza, and Agostino Cortesi. 2018. Procedurally Provisioned Access Control for Robotic Systems. In *2018 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*.
[91] Yuan Xu, Gelei Deng, Tianwei Zhang, Han Qiu, and Yungang Bao. 2021. Novel denial-of-service attacks against cloud-based multi-robot systems. *Information Sciences* 576 (2021), 329–344. https://www.sciencedirect.com/science/article/pii/S002002552100654X
[92] Yuan Xu, Tianwei Zhang, and Yungang Bao. 2021. Analysis and Mitigation of Function Interaction Risks in Robot Apps. *24th International Symposium on Research in Attacks, Intrusions and Defenses* (2021).
[93] Fabian Yamaguchi, Nico Golde, Daniel Arp, and Konrad Rieck. 2014. Modeling and Discovering Vulnerabilities with Code Property Graphs. In *2014 IEEE Symposium on Security and Privacy*. 590–604.
[94] Shucheng Yu, Cong Wang, Kui Ren, and Wenjing Lou. 2010. Attribute Based Data Sharing with Attribute Revocation. In *Proceedings of the 5th ACM Symposium on Information, Computer and Communications Security* (Beijing, China) *(ASIACCS '10)*. Association for Computing Machinery, New York, NY, USA, 261–270.
[95] Yizhe Zhang, Lianjun Li, Michael Ripperger, Jorge Nicho, Malathi Veeraraghavan, and Andrea Fumagalli. 2018. Gilbreth: A Conveyor-Belt Based Pick-and-Sort Industrial Robotics Application. In *2018 Second IEEE International Conference on Robotic Computing (IRC)*. 17–24.
[96] Huiquan Zhu, Jing Sun, Jin Song Dong, and Shang-Wei Lin. 2015. From verified model to executable program: The Pat Approach. *Innovations in Systems and Software Engineering* 12, 1 (2015), 1–26.