Investigating Security Threats in Multi-Tenant ROS 2 Systems

Lichen Xia, Xing Gao and Weisong Shi

Abstract— **Robot Operating System (ROS) has been widely used to develop robotic applications. The first generation of ROS generally lacks security features, and ROS 2 is introduced with security support. However, security concerns still exist for running ROS in practical multi-tenant environments. In this paper, we conduct an in-depth investigation into the security of ROS 2. We focus on vulnerabilities in ROS nodes and topics and intend to explore methods to break the isolation and security mechanisms systematically. We devise a set of strategies that can be exploited by attackers to escalate privilege or cause information leakage in a multi-tenant environment. These attacks can bypass existing isolation and security mechanisms, including ROS 2's native security module. To validate our findings, we employ simulations across various real-world scenarios to demonstrate how attackers could exploit these vulnerabilities to bypass existing security mechanisms. Finally, we present several defense practices to mitigate these identified threats.**

I. Introduction

Robot Operating System (ROS) has been widely adopted in both research and industrial sectors [1]. However, ROS lacks robust security features by design and thus is vulnerable to various attacks [2], [3]. To address the limitation, ROS 2 [4] is introduced with multiple security features provided. It adopts the Data Distribution Service (DDS) as its communication middleware and employs security extensions offering authentication, authorization, and encryption functionalities. With these security features, ROS 2 has gained huge popularity in existing robotic companies [5], [6].

Despite the success, there always exist security concerns for running ROS 2 in complex multi-robot systems (MRS) [7], [8], [9], where a number of heterogeneous robots, presumably belonging to different tenants, collaboratively work on particular workloads. It is unclear whether existing isolation and security mechanisms can provide enough protection against various malicious attacks such as privilege escalation. In addition, it is common for a robot to run multiple packages simultaneously to execute complicated tasks. However, The open-source nature of the ROS community could potentially enlarge the attack surface, as packages might be developed by less trusted third parties.

In this paper, we intend to systematically explore methods to break the isolation and security mechanisms in ROS 2 and understand the security impact in a multi-tenant environment (e.g., MRS). We primarily focus on vulnerabilities in the fundamental ROS units (i.e., nodes and topics) that can compromise existing security mechanisms. We devise a set of strategies that can be exploited by attackers to escalate privilege or cause information leakage. In particular, while ROS 2 ensures that newly created nodes will be confined under the same privilege policies as their parents, we find that the nodes can exploit hidden methods, such as internal APIs, to escape the privilege inheritance mechanism. Attackers can bypass the isolation of namespaces and domain ID, and even break SROS 2 due to the weak bond between nodes and security enclaves. We also find that the shared topics and inter-topic communication can potentially leak information in a multi-tenant environment due to the lack of fine-grained node permission granularity.

In addition, we investigate the security of a cross-system scenario where the system integrates both ROS 1 and ROS 2 packages. While ROS1-bridge brings great flexibility for interconnecting ROS 1 and ROS 2, it unfortunately creates new security loopholes. Due to the lack of security mechanisms in ROS 1 systems, the ROS 1 side of the bridge is exposed to attackers. Even worse, attackers can exploit the ROS1-bridge further to invalidate the security mechanisms of the connected ROS 2 system.

To demonstrate the effectiveness of identified vulnerabilities, we conduct experiments using popular ROS packages in real-world multi-tenant scenarios. Our results show that by exploiting these vulnerabilities, adversaries can potentially break the isolation, bypass access control, and receive messages from unauthorized topics. We further propose several defense practices to mitigate the identified vulnerabilities from different perspectives. Particularly, we develop a static analysis tool to inspect ROS packages for detecting potential topic name collisions with high accuracy.

Finally, we have reported the identified vulnerabilities to the ROS 2 development team, including privilege escalation, weak bonding in security enclaves, and abuse of the ROS1-bridge. They have acknowledged and discussed our concerns during their Project Management Committee (PMC) meeting.

II. Background

A. Robot Operating System

ROS [10] is an open-source framework designed to facilitate the software development for robots [11], [12]. There are two ROS generations (i.e., ROS 1 and ROS 2). ROS 2 shares fundamental concepts and features with ROS 1 but has general improvements in performance and security.

Node and topic. In ROS, a node is the basic executable unit for a single and modular functionality, such as controlling

We would like to thank the anonymous reviewers for their insightful and constructive comments, which helped us to improve the quality of this paper. This work was supported in part by the National Science Foundation (NSF) grants CNS-2140346, CNS-2317830, and CNS-2338837.

The authors are with the Department of Computer and Information Sciences, University of Delaware, Newark, DE 19716, USA. Email: {lxia, xgao, weisong}@udel.edu

sensors and managing motor output. Typically, a ROS system and package contain multiple nodes. Nodes can exchange messages via topics, which are identified by their names. Nodes can publish messages to a topic (e.g., as a publisher) or subscribe to a topic to receive messages (e.g., as a subscriber). Multiple publishers and subscribers can connect to the same topic without knowing each other.

Network isolation. A ROS system can contain multiple packages, which might come from unreliable third-party developers. To enforce basic isolation, ROS provides a namespace mechanism that can be used to group topics and nodes into isolated structures. The namespace is automatically added as the prefix of associated nodes and topics. Namespaces help avoid naming conflicts in ROS, ensuring that each component has a unique identity within its scope.

ROS Command Line Interface (CLI). The ROS CLI is a suite of tools enabling users to manage and inspect ROS systems through terminal commands. Through the CLI, users can easily navigate through and manipulate the ROS system, thereby accelerating the development, testing, and troubleshooting processes during the robotic development.

B. ROS 2 Versus ROS 1

ROS 2 utilizes DDS as the communication architecture [13] and includes several security features based on DDS. ROS 2 also provides the ROS Client Library (RCL) to abstract the complexity of DDS. RCL provides necessary user-space APIs for developers to write ROS 2 codes.

Domain ID. ROS 2 further incorporates a Domain ID networking isolation mechanism by enabling multiple logical networks to share a physical one. It is a numeric value that can be assigned to a group of nodes and topics by system owners, either through an environment variable or RCL APIs. ROS 2 nodes can only interact with topics within the same logical network. This ensures that messages are exchanged only between ROS 2 nodes with the same Domain ID.

Context. In ROS 2, a *context* is an internal state encapsulating some non-global states of nodes. It serves as the foundation for the node's initialization and configuration, providing various critical information during node registration, including namespace and communication settings. A node must be bound with only one context throughout its lifetime, while a context can be utilized by multiple nodes.

ROS1-bridge. While ROS 2 brings significant enhancement, many projects have not yet transformed and still adopt ROS 1. ROS1-bridge emerges as a transitional solution to ensure interoperability between the two systems. The design is straightforward: A ROS 2 node and a ROS 1 node are simultaneously created. The bridge then dynamically scans both the ROS 2 network and ROS 1 network to find matched topics. If a topic is found in both networks, the bridge is responsible for publishing the message received from one side to the matched topic on the other side. In this way, the ROS1-bridge enables message communication between ROS 1 and ROS 2 nodes, and developers can deploy both ROS 1 and ROS 2 packages in a united system. Thus, many large projects have adopted ROS1-bridge [14], [15], [16].

C. Secure ROS 2

The security in ROS 2 (SROS 2) offers a suite of security features [17]. To set up SROS 2 for a node, the system owner needs to allocate nodes with a security enclave. It includes a pair of a private key and a public certificate, both signed by a trusted Identity Certificate Authority. This enables node verification by checking the certificate [18]. Also, an enclave contains a permission file, which specifies its accessible topics, namespaces, and domains. This permission file grants permissions to nodes based on their names. As a result, only ROS 2 nodes with qualified names can use a specific enclave to join the ROS 2 system protected by SROS 2.

III. Motivation and Threat Model

A. Motivation

This work investigates potential vulnerabilities in ROS 2. A compromised ROS 2 system could lead to severe consequences, as it commonly operates in high-security environments (e.g., autonomous driving) that physically interact with humans. Thus, discovering security vulnerabilities in ROS 2 is important for ensuring reliable and safe system operations.

We consider a realistic multi-tenant robotic scenario where components (e.g., nodes) from different parties are running on the same ROS 2 system. With robots becoming popular, multi-tenant usage has become normal. For example, in a distributed multi-robot system (MRS), multiple industrial robots from different parties might collaboratively work together on a complicated task. A domestic robot might install and run multiple ROS 2 applications developed by different developers. A ROS 2 application could also integrate third-party ROS 2 packages to accelerate the software development process. As anyone can be the developer and share their applications/packages, third-party components may be malicious or compromised during the distribution [19].

We conduct a measurement study on open-source robotic projects based on ROS 2, and find that 45 of 54 open-source projects collected from GitHub utilize ROS 2 packages from external developers. These third-party packages are widely employed for key functions such as planning. The results suggest that the multi-tenant environment is quite common in ROS 2.

B. Threat Model

We examine a generalized multi-tenant application scenario where multiple ROS packages (e.g., applications) run concurrently within a unified ROS system. The container technology (e.g., Docker) is adopted to provide OS-level isolation for ROS applications. Thus, apps can only communicate with each other via the topic mechanism. We generally consider basic isolation mechanisms (e.g., namespaces and domain IDs) to be enabled unless explicitly mentioned.

Attackers' capabilities. We assume the attacker is one tenant and controls one or more nodes in the target multi-tenant ROS 2 system. For example, attackers might control one robot (and its nodes) in distributed MRS or one package running on top of one robot. With the controlled node, attackers can publish/subscribe to topics confined by the deployed isolation and security mechanism. Besides, attackers can further exploit tools provided by ROS. Attackers have two general goals. The first one is information leakage: to steal sensitive data such as extracting data from other applications. The second goal is privilege escalation: to escalate their privilege and then they may stealthily manipulate the behavior of other running packages or even the system.

IV. Potential Threats in ROS Systems

A. Node Privilege Escalation

1) Incomplete Context Privilege Inheritance: In ROS 2, the responsibility of a DDS participant is managed by the node and its context. While the node performs functionalities, its privileges (e.g., namespace and domain ID) are governed by its associated context. Nodes under the same context share identical privileges. To escalate its privilege, a node needs to modify the configurations of its running context, which is not allowed by ROS 2 via any public RCL APIs. In addition, ROS 2 adopts a node privilege inheritance mechanism: the newly spawned node (e.g., using RCL APIs) will automatically inherit its parent node's privilege.

However, since the context and the node are two separate objects, if a node can create a new context, it has the ability to escalate its privilege. Following this approach, we find two general ways to escalate the node's privilege in ROS 2. (1) *Hidden APIs.* Firstly, we find that the internal APIs [20] of the ROS Client Library (RCL), which are not intended for use by ROS 2 packages, are unfortunately exposed without any restriction. As a result, a node can actually access these APIs to create and configure new contexts (e.g., using a context constructor). Newly established contexts in this approach do not inherit settings from their predecessors (e.g., the creating nodes). It thus allows nodes to create contexts that have different (escalated) privileges, and further create new nodes inside the new context, causing privilege escalation. (2) *Abusing CLIs.* Secondly, a node can exploit ROS CLIs to create a new unrestricted context. The ROS CLIs will always start a new process to create new nodes. To ensure the state is independent from existing nodes, a new context is created and initialized. Thus, a node can use ROS CLIs to create a context with different privileges.

Privilege escalation. Through the above two methods, a malicious node can escalate its privileges, such as breaking the isolation of namespace and domain IDs. Particularly, the malicious node can first initialize a new context and then set the Namespace and Domain ID of this context to the target one (e.g., belonging to other tenants). Under this new context, any new nodes will not inherit their parent's privileges. Table I presents a list of features that can be manipulated. We find that, while ROS CLIs cannot directly set Domain IDs, attackers can pre-configure nodes using environment variables by setting the *ROS_DOMAIN_ID* environment variable. In addition to domain ID and namespace, attackers can also modify the topic mapping rules.

TABLE I: Features that can be manipulated.

Features	**RCL APIs**	**ROS CLI**	**Environment Var**
Domain ID	✓	-	✓
Namspace	✓	✓	-
Topic Remapping	✓	✓	-
SROS 2 (Enclave)	✓	✓	✓

2) Weak-bond SROS 2 files: SROS 2 enclaves enforce permissions on nodes by setting node name restrictions. This means that an enclave can be applied to multiple nodes simultaneously. Any node that fulfills the restrictions of the enclave can utilize the enclave and gain access to certain topics granted by the corresponding permission file.

Breaking SROS 2. Unfortunately, SROS 2 settings can be easily manipulated via hidden APIs and ROS CLIs. By creating new contexts and nodes, a malicious node either disables SROS 2 enforcement or assigns a new SROS 2 enclave to the newly created child nodes. The SROS 2 restrictions will not be inherited by child nodes, which can obtain all permissions granted by the newly attached SROS 2 enclaves (e.g., access to protected topics). In this way, attackers can dynamically exploit any leaked enclaves to escalate privileges. For example, in an MRS, if attackers control one robot with escalated privileges (e.g., can access restricted areas), attackers can simply distribute its enclaves to other robots, and other robots can also gain the same privilege.

Even worse, the system admin can invalidate an enclave via two methods: (1) Passively wait for the enclave to expire. However, the expiration time is set when the enclave is created, and there is no way to update it after distribution. (2) Update all SROS 2 files (for all nodes and physical devices), which requires restarting nodes. Both methods cause inconvenience and can potentially cause huge losses.

```
1 def topic_callback(self, msg):
2     os.environ["ROS_SECURITY_ENCLAVE_OVERRIDE"] = "/office
      /door_supervisor"
3     new_context = Context()
4     rclpy.init(args=['--ros-args', '--remap', '__ns:=/
       wrong_namespace'], context=new_context, domain_id=5)
5     attackNode = DoorRequestNode('door_request_node', True
       , new_context)
6     rclpy.spin(attackNode)
```

Listing 1: Code snippet of exploting RCL internal APIs

Listing 1 presents a code snippet that a malicious node creates a new node outside of its restrictions by exploiting RCL internal APIs. It first uses the internal API *Context()* to create a new context (line 3), and then initiates it with a different namespace (i.e., $wrong\_namespace$) and domain (line 4). The newly created node (i.e., $door\_request\_node$) is under the new context which breaks all restrictions on its parent node (line 5). Attackers can also set the environment variable *ROS_SECURITY_ENCLAVE_OVERRIDE* to assign a leaked enclave to the new node (line 2) for breaking SROS 2.

B. Topic Information Injection and Leakage

1) Topic Name Collision: ROS 2 nodes exchange messages via subscribing and publishing to topics based on topic names. One problem is name collision: different packages

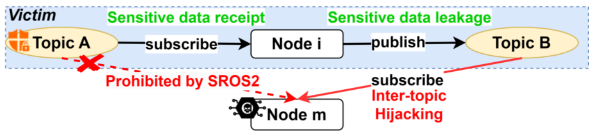


Fig. 1: Inter-topic hijacking demonstration.

might contain topics with identical names. Different tenants might unexpectedly exchange messages using the same topic, causing information leakage or data manipulation.

There are two scenarios that can cause unexpected topic name collision. First, packages might unintentionally have overlapping topics under the same namespace. Second, even if packages are running under different namespaces, it is quite common to have *global* topics that are not subject to namespaces. For example, the topics */tf* and */parameter_events* are two representative global topics in ROS. The */tf* topic is specialized for broadcasting 3D coordinate frames and their transformation information between packages. The */parameter_events* topic is used for notifying parameter events: any updates of the node's parameter will be published to notify all nodes in the ROS system. Those global topics are shared among different packages.

Lack of permission granularity. In both cases, attackers can intentionally exploit collude topics to steal sensitive information, even with SROS 2 enabled. The reason is that the current ROS 2 (with SROS 2) does not provide fine-grained permission granularity. SROS 2 only grants node access to specific topics based on permission files. Thus, once a node gains access to a topic (e.g., via topic collision), it has full privilege on the target topic. As a result, sensitive data sent by other nodes can be leaked via the collude topic.

Lack of node identification in topics. In addition, while SROS 2 ensures node authentication, this level of authentication does not extend to the messages that are exchanged in topics. In particular, with a publisher-subscriber model, multiple publishers can concurrently send messages to a single topic. However, there is no mechanism to enable subscribers to know the origin of the message (e.g., who sends this message), and further filter unwanted messages. As a result, an attacker can publish potentially harmful content to poison the topic, forcing all subscribers of the topic to receive unwanted or even dangerous messages.

2) Inter-Topic Hijacking: Component hijacking [21], [22] is a well-known security vulnerability in the Android system: an unprivileged component can hijack an exposed component to escalate privileges and access privileged components. Unfortunately, similar threats exist in ROS 2. For a topic (e.g., Topic A in Figure 1), SROS 2 can set permissions to prohibit malicious nodes (e.g., Node m) from subscribing to it, so unprivileged nodes cannot read information from the topic directly. However, intermediate nodes (e.g., Node i), which have the privilege to Topic A, might read messages from Topic A, and publish (either intentionally or accidentally) the information to another topic (e.g., Topic B) that is not protected by SROS 2. In this case, the malicious node can exploit intermediate nodes to access protected data. We refer to this threat as inter-topic hijacking. The root reason is that ROS 2 lacks fine-grained data flow tracking across topics.

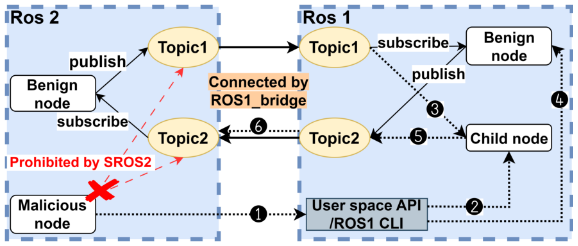


Fig. 2: Cross-system privilege escalation demonstration

C. Cross-System Privilege Escalation

While ROS1-bridge greatly eases the process for system owners to integrate both ROS 1 and ROS 2 packages, it, unfortunately, introduces additional security vulnerabilities in MRS. Attackers who control ROS 2 nodes can exploit ROS1-bridge to escalate privileges, even with SROS 2 enabled. Particularly, ROS 1 adopts namespace as the network-level isolation mechanism. However, there is no security mechanism to protect ROS 1 systems, including ROS 1 user-space APIs and ROS 1 CLIs. Thus, ROS 2 nodes can create new ROS 1 nodes with arbitrary names and namespaces by directly using ROS 1 user-space APIs or CLI, as step ❶ illustrated in Figure 2.

As ROS 1 and ROS 2 are separate systems, the newly created ROS 1 nodes are not confined by the originating ROS 2 nodes' context (step ❷). Meanwhile, due to the lack of security mechanisms in ROS 1, these newly created ROS 1 nodes can access any existing topics and nodes within the ROS 1 system (step ❸). Particularly, ROS 1 allows users to terminate ROS 1 nodes using CLI (step ❹), enabling attackers (e.g., malicious ROS 2 nodes) to launch denial-of-service (DoS) attacks to shut down any nodes without any privileges.

Even worse, the newly created ROS 1 nodes can also access the ROS1-bridge (step ❺), as ROS 1 has no security mechanism. Thus, malicious ROS 2 nodes can further exploit ROS1-bridge to attack ROS 2 nodes, such as accessing or publishing malicious messages to ROS 2 topics that are protected by SROS 2 (step ❻).

V. Evaluation

We conduct experiments using popular ROS packages in real-world scenarios. We run ROS 2 Humble [23] (one widely used version) on an Ubuntu server with an Intel i9-9940X CPU, 64GB of RAM, and a GeForce RTX 2080Ti GPU. For the cross-system experiment, we run ROS 1 Noetic [24] and ROS 2 Rolling [25] (which supports ROS1-bridge). *eProsima's Fast DDS* [26] is incorporated as the standard practice.

A. Node Privilege Escalation

We demonstrate the effectiveness of node-level attacks using Open-RMF [27], which is an open-source platform facilitating robot interoperability and management. We select the office scenario with SROS 2 enabled, as presented in Figure 3. This automated system contains two robots and

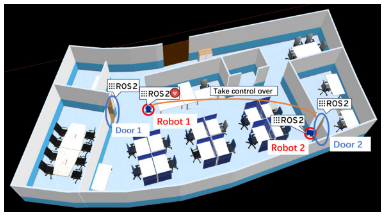


Fig. 3: Open RMF simulation for node privilege escalation.

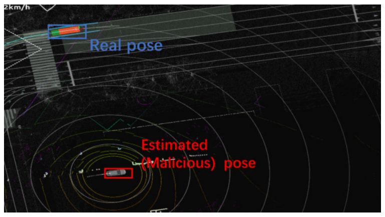
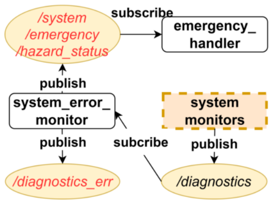
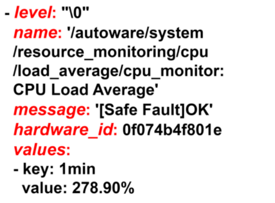


Fig. 4: Localization drifting while Autoware is attacked.



(a) Autoware errors handling process.



```
- level: "\0"
  name: '/autoware/system
  /resource_monitoring/cpu
  /load_average/cpu_monitor:
  CPU Load Average'
  message: '[Safe Fault]OK'
  hardware_id: 0f074b4f801e
  values:
  - key: 1min
    value: 278.90%
```

(b) Message from */diagnostics_err*

Fig. 5: Inter-topic hijacking in Autoware

two doors. Robot 2 has a higher priority and can deliver items to two private rooms, by sending "open" messages to the door node through ROS 2 topic */door_request*. Instead, Robot 1 is a cleaning robot controlled by attackers that can only work in the public area (i.e., cannot control either door).
Privilege escalation to break network isolation. Robot 2 and both doors execute in the same domain (i.e., with the same *Domain_ID* and under the same namespace */victim_ns*), but are isolated from Robot 1. By default, Robot 1 cannot access any topics under the namespace */victim_ns*. However, Robot 1 can create a new node, and assign it with the *Domain_ID* and namespace of the targeted door node. Malicious nodes on Robot 1 can further send messages by spoofing the *Sender_ID* and successfully control the target door. We have also conducted a similar experiment to demonstrate that SROS 2 can also be bypassed by this mechanism.

B. Topic Information Injection and Leakage

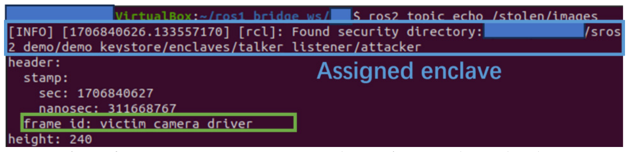
For topic-related threats, we launch the attack against an autonomous vehicle using Autoware [28], which is an open-source software stack designed for self-driving vehicles.
Spoofing attack on shared topic. Autoware comprises both official and third-party ROS 2 packages, offering multiple features such as 3D localization, path planning, and vehicle control. Several packages need to access the global topic */tf*, which records the critical states of the vehicle. For example, the Localization package publishes the estimated location and orientation to */tf*. The messages on */tf*, known as *Transforms*, are identified by *frame_ID* and *child_frame_ID*. However, these fields are not bound to the publisher's identity. Thus, any publisher can insert messages with spoofed IDs.

Figure 4 illustrates a scenario where an attacker publishes false location estimations to */tf* with spoofed IDs. Attackers can publish malicious Transforms with a frequency similar to legitimate ones. The attack can cause a significant deviation of the AV's estimated location (highlighted with the red color) from its actual physical location (blue). This discrepancy can lead to unsafe self-driving decisions (e.g., emergency braking or collisions) due to unstable localization.
Inter-topic hijacking attack. Figure 5a presents Autoware's workflow for handling system errors. In Autoware, system states are sent to the *system_error_monitor* node through the */diagnostics* topic. The processed data (including system hazard level and failure details) will be further published to the topic */system/emergency/hazard_status* so the *emergency_handler* has the necessary information to make the right decisions. Obviously, such data is sensitive and should not be accessible by normal nodes. However, we find another topic, */diagnostics_err*, which is a public interface for visualizing error messages, also receiving the same data.

Figure 5b shows a typical message from the */diagnostics_err* topic. This message encapsulates various critical data. The hazard level (\0) indicates this is not an error/warning message needed by the *emergency_handler*. Other detailed hardware states in the 'value' attribute are also not necessary. A malicious node, without access to the */diagnostics* topic, can collect critical hardware and software information by subscribing to the */diagnostics_err* topic.

C. Cross-System Privilege Escalation

To demonstrate that ROS1-bridge can cause cross-system risks, we conduct a case study using a demo camera application [29], where both ROS 2 and ROS 1-powered devices are required to collaborate. Specifically, the ROS 2 package, as the driver of a robot camera, publishes images to the ROS 2 topic */image*. The ROS 1 package executes remotely and provides the system owner with a Graphic User Interface (GUI) to supervise the robot. The ROS 1 package subscribes to the ROS 1 topic */image* and displays the received images on its GUI. Additionally, the ROS 1 GUI offers a vertical flip feature: users can flip images by publishing commands to the ROS 1 topic */flip_image*. In the meantime, the corresponding ROS 2 topic */flip_image* is subscribed by the ROS 2 camera driver on the robot. The identically named ROS 1 and ROS 2 topics are bridged by the ROS1-bridge. There is a malicious ROS 2 node controlled by attackers running on another robot. However, SROS 2 is enforced, preventing unauthorized access to the secured ROS 2 topics from the malicious node.
Data insertion. The malicious ROS 2 node can exploit ROS1-bridge to flip the image in the ROS 2 camera driver without permission for the ROS 2 topic */flip_image*. First, the malicious ROS 2 node creates a new ROS 1 node and registers it as the publisher of the ROS 1 topic */flip_image*. Due to the lack of access control for ROS 1 nodes, the newly created ROS 1 node can insert messages (e.g., flip the image) into the connected topic (*/flip_image*). These messages, unfortunately, will be automatically relayed to the ROS 2 topic */flip_image* via the ROS1-bridge. As a result, the image will be flipped by attackers, even without permission.
Private data leakage. Attackers can also intercept images from the ROS 2 camera driver without legitimate access to the target ROS 2 topic */image*. Similarly, the malicious node can create a ROS 1 node to subscribe to the ROS 1 topic */image*. Images published to the ROS 2 topic */image*



```
[redacted]VirtualBox:~/ros1 bridge ws/[redacted]$ ros2 topic echo /stolen/images
[INFO] [1706840626.133557170] [rcl]: Found security directory: [redacted]/sros
2 demo/demo keystore/enclaves/talker listener/attacker
header:
  stamp:
    sec: 1706840627
    nanosec: 311668767
  frame id: victim camera driver
height: 240
```

Fig. 6: Cross-system attack: private data leakage

will be automatically transferred to the corresponding ROS 1 topic */image* by the ROS 1 bridge. Figure 6 presents the metadata of an intercepted image. The images published by the ROS 2 camera driver share a common frame_id: *victim_camera_driver*, indicating that the captured image originates from the target ROS 2 camera driver.

DoS attack. Attackers can simply kill the ROS 1 GUI, which is managed by a ROS 1 node *rqt_image_view*. Malicious ROS 2 nodes can send a command "ROS kill *rqt_image_view*" via the ROS 1 CLI and the ROS 1 GUI will be shut down immediately without the identifying information of attackers.

VI. Discussion and Mitigation

Below we discuss several defensive practices.

Eliminate attack vectors. One straightforward countermeasure is restricting ROS 2 nodes from accessing relevant RCL internal APIs and CLIs (e.g., hiding the exposed methods). These steps do not require any significant changes to the ROS 2 system and can immediately eliminate many attack vectors. However, it may add restrictions to the functionality of ROS 2 apps, as legitimate apps may use these channels.

Hard-bonding security enclaves. The SROS 2 security enclave should not solely rely on the node's name for validity, which enables any node with a matching name to exploit a leaked enclave causing privilege escalation. Instead, an enclave could be specifically tied to a ROS 2 node.

Enable message-level authentication. If benign ROS 2 nodes can identify the source of received messages, they can filter out malicious publishers. One solution is to enable message-level authentication. This feature is particularly critical in scenarios where multiple nodes share a topic.

Package Inspection. Another mitigation is to inspect third-party ROS 2 packages before integrating them into ROS systems. For example, static code analysis can identify certain misbehavior code patterns.

To mitigate both intentional and accidental topic collisions, we have developed a static code analysis tool to identify topics generated by ROS 2 packages. Figure 7 illustrates the overall design. The high-level idea is to utilize pre-defined patterns to locate code snippets for creating topics, and further extract their names. Our tool searches code segments that create nodes and topics by checking corresponding user-space APIs [20]. When it finds related code snippets, it extracts names from the function arguments. For variables, we further trace their values in configuration files that store/load configuration information during package launch [30]. Our tool also searches third-party templates from external packages for creating publishers and subscribers. Note that the list of third-party templates can be accumulatively updated in a periodical fashion to include all major patterns.

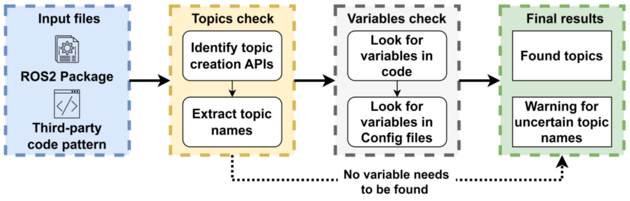

Input files
ROS2 Package
Third-party code pattern
Topics check
Identify topic creation APIs
Extract topic names
Variables check
Look for variables in code
Look for variables in Config files
Final results
Found topics
Warning for uncertain topic names
No variable needs to be found

Fig. 7: ROS2 package analysis tool for identifying possible topics created by the analyzed package

We have applied our tool to 123 ROS 2 packages across 55 open-source repositories, randomly collected from GitHub. We assess our tool using two metrics: 1) The proportion of accurately identified topics (i.e., true positive); and 2) The rate of incorrectly identified but non-existent topics (i.e., false positive). Among 539 manually extracted topics, our tool successfully identifies 92.6% of the topics while maintaining a false positive rate of only around 2.97%.

VII. Related Works

ROS 1 security. Security has been one of the main concerns for ROS 1 and attracted extensive research efforts [31], [32]. As ROS 1-based applications are known to be vulnerable to multiple threats [33], [34], [35], [36], many protocols and systems have been developed to secure the communications between nodes and applications [37], [38], [39], [40], [41].

ROS 2 and SROS 2 security. ROS 2 security also attracts much research attention [42], [43], [44], [45], [46]. DiLuoffo et al. [47], [48] provided a comprehensive overview of the SROS 2 model and highlighted that threats still exist, including eavesdropping attacks from malicious software. Our work differs from them as we investigate the security of ROS 2 fundamental components (e.g., nodes and topics) in multi-tenant environments.

A recent study [14] showed that ROS1-bridge introduces latency in ROS 2 with SROS 2 enabled. We further demonstrated that ROS1-bridge can be exploited to escalate privileges and attack ROS2. Additionally, Deng et al. [49] found that robots can reuse expired SROS 2 enclaves to regain permissions. We further reveal the weak bond between SROS 2 enclaves and nodes as the root cause.

Privilege escalation. Privilege escalation related vulnerabilities have been extensively studied in various domains including Android [50], [51], [52], Web applications [53], and distributed systems [54]. Particularly, component hijacking is a notorious vulnerability in Android that can cause privilege escalation. Our work is motivated by Android component hijacking but focuses on examining similar problems in ROS 2.

VIII. Conclusion

This paper conducts an in-depth investigation into the security of ROS 2 and its security extension. We uncover five general vulnerabilities distributed across the node, topic, and cross-system layers. For each vulnerability, we propose practical attacks and demonstrate how these attacks could be utilized to bypass the existing security features in real-world scenarios. We also discuss several mitigation strategies.

REFERENCES

[1] "The Rise of ROS: Nearly 55% of Total Commercial Robots Shipped in 2024 Will Have at Least One Robot Operating System Package Installed," 2019, accessed: 2024-01-20. [Online]. Available: https://www.businesswire.com/news/home/20190516005135/en

[2] N. DeMarinis, S. Tellex, V. P. Kemerlis, G. Konidaris, and R. Fonseca, "Scanning the Internet for ROS: A View of Security in Robotics Research," in 2019 International Conference on Robotics and Automation (ICRA). IEEE, 2019, pp. 8514–8521, available: https://ieeexplore.ieee.org/abstract/document/8794451.

[3] R. R. Teixeira, I. P. Maurell, and P. L. Drews, "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). IEEE, 2020, pp. 1–6, available: https://ieeexplore.ieee.org/abstract/document/9307107.

[4] B. G. C. L. S. Macenski, T. Foote and W. Woodall, "Robot Operating System 2: Design, architecture, and uses in the wild," Science Robotics, vol. 7, no. 66, p. eabm6074, 2022. [Online]. Available: https://www.science.org/doi/abs/10.1126/scirobotics.abm6074

[5] iRobot Corporation, "iRobot: Robot Vacuums and Mops," https://www.irobot.com/, 2024, accessed: 2024-01-20.

[6] ROBOTIS, "ROBOTIS Official Website," http://www.robotis.us, accessed: 2024-01-20.

[7] Amazon Web Services, Inc., "AWS RoboMaker," https://aws.amazon.com/cn/robomaker/, accessed: 2024-01-20.

[8] Open Robotics, "ROS 2 Multi-Robot Book," https://osrf.github.io/ros2multirobotbook/, 2024, accessed: 2024-01-31.

[9] G. L. T. C. H. D. L.-D. G. X. G. S. L. C. Q. E. Villemure, P. Arsenault and F. Ferland, "SwarmUS: An open hardware and software on-board platform for swarm robotics development," arXiv preprint arXiv:2203.02643, 2022.

[10] M. Quigley, K. Conley, B. Gerkey, J. Faust, T. Foote, J. Leibs, R. Wheeler, and A. Y. Ng, "ROS: an open-source Robot Operating System," in ICRA Workshop on Open Source Software, vol. 3, no. 3.2. Kobe, Japan, 2009, p. 5, available: http://lars.mec.ua.pt/public/LAR%20Projects/BinPicking/2016_RodrigoSalgueiro/LIB/ROS/icraoss09-ROS.pdf.

[11] T. Foote and K. Scott, "ROS Community Metrics Report," http://download.ros.org/downloads/metrics/metrics-report-2020-07.pdf, accessed: 31-10-2023.

[12] ROS Community, "robots.ros.org," https://robots.ros.org/, accessed: 31-10-2023.

[13] "DDS Foundation," https://www.dds-foundation.org/, accessed: 2024-01-10.

[14] S. Sandoval and P. Thulasiraman, "Cyber Security Assessment of the Robot Operating System 2 for Aerial Networks," in 2019 IEEE International Systems Conference (SysCon). IEEE, 2019, pp. 1–8, available: https://ieeexplore.ieee.org/abstract/document/8836824.

[15] "PX4 Autopilot," https://px4.io/, accessed: 2024-01-10.

[16] ROS Industrial Consortium, "ROS Industrial," https://rosindustrial.org, accessed: 2024-01-10.

[17] V. Mayoral-Vilches, R. White, G. Caiazza, and M. Arguedas, "SROS2: Securing ROS 2 Over DDS," in 2022 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS). IEEE, 2022, pp. 11 253–11 259, available: https://ieeexplore.ieee.org/abstract/document/9982129.

[18] A. M. S. G. M. Myers, R. Ankney and C. Adams, "X. 509 Internet public key infrastructure online certificate status protocol-OCSP," https://www.rfc-editor.org/rfc/rfc2560, Tech. Rep., 1999, accessed: 2024-01-20.

[19] S. Macenski, T. Foote, B. Gerkey, C. Lalancette, and W. Woodall, "Robot Operating System 2: Design, architecture, and uses in the wild," Science Robotics, vol. 7, no. 66, p. eabm6074, 2022. [Online]. Available: https://www.science.org/doi/abs/10.1126/scirobotics.abm6074

[20] Open Robotics, "rclcpp - ROS 2 C++ Client Library Documentation," https://docs.ros.org/en/humble/p/rclcpp/generated/index.html, 2022, accessed: 2024-01-20.

[21] Z. W. W. L. L. Lu, Z. Li and G. Jiang, "Chex: Statically vetting android apps for component hijacking vulnerabilities," in Proceedings of the 2012 ACM Conference on Computer and Communications Security, 2012, pp. 229–240.

[22] A. B. C. K. S. Poeplau, Y. Fratantonio and G. Vigna, "Execute this! analyzing unsafe and malicious dynamic code loading in android applications," in NDSS, vol. 14, 2014, pp. 23–26.

[23] Open Robotics, "ROS Humble Hawksbill Documentation," https://docs.ros.org/en/humble/index.html, accessed: 2024-01-31.

[24] ROS.org, "ROS Noetic Ninjemys," http://wiki.ros.org/noetic, 2024, accessed: 2024-01-31.

[25] Open Robotics, "ROS 2 Rolling Ridley Documentation," https://docs.ros.org/en/rolling/index.html, 2024, accessed: 2024-01-31.

[26] eProsima, "eProsima Fast DDS," https://www.eprosima.com/index.php/products-all/eprosima-fast-dds, 2024, accessed: 2024-01-31.

[27] Open-Rmf, "Open-RMF/RMF_DEMOS: Demonstrations of the openrmf software," GitHub, n.d. [Online]. Available: https://github.com/open-rmf/rmf_demos

[28] Y. I. Y. N. K. T. S. Kato, E. Takeuchi and T. Hamada, "An open approach to autonomous vehicles," IEEE Micro, vol. 35, no. 6, pp. 60–68, 2015.

[29] R. Developers, "Example 2: Run the bridge and exchange images in ros1bridge," https://github.com/ros2/ros1_bridge#example-2-run-the-bridge-and-exchange-images, 2024, accessed: 2024-01-31.

[30] ROS 2 Documentation Contributors, "Using ROS 2 Launch for Large Projects: Loading Parameters from a YAML File," https://docs.ros.org/en/humble/Tutorials/Intermediate/Launch/Using-ROS2-Launch-For-Large-Projects.html#loading-parameters-from-yaml-file, 2023, accessed: 2024-01-31.

[31] J. McClean, C. Stull, C. Farrar, and D. Mascarenas, "A Preliminary Cyber-Physical Security Assessment of the Robot Operating System (ROS)," in Unmanned Systems Technology XV, vol. 8741. SPIE, 2013, pp. 341–348.

[32] S.-Y. Jeong, I.-J. Choi, Y.-J. Kim, Y.-M. Shin, J.-H. Han, G.-H. Jung, and K.-G. Kim, "A Study on ROS Vulnerabilities and Countermeasures," in Proceedings of the Companion of the 2017 ACM/IEEE International Conference on Human-Robot Interaction, 2017, pp. 147–148, available: https://dl.acm.org/doi/abs/10.1145/3029798.3038437.

[33] B. Dieber, B. Breiling, S. Taurer, S. Kacianka, S. Rass, and P. Schartner, "Security for the Robot Operating System," Robotics and Autonomous Systems, vol. 98, pp. 192–203, 2017, available: https://www.sciencedirect.com/science/article/abs/pii/S0921889017302762.

[34] S. Rivera, S. Lagraa, and R. State, "ROSploit: Cybersecurity Tool for ROS," in 2019 Third IEEE International Conference on Robotic Computing (IRC). IEEE, 2019, pp. 415–416, available: https://ieeexplore.ieee.org/abstract/document/8675686.

[35] B. Dieber, R. White, S. Taurer, B. Breiling, G. Caiazza, H. Christensen, and A. Cortesi, "Penetration Testing ROS," Robot Operating System (ROS) The Complete Reference (Volume 4), pp. 183–225, 2020, available: https://link.springer.com/chapter/10.1007/978-3-030-20190-6_8.

[36] R. Carvalho, A. Cunha, N. Macedo, and A. Santos, "Verification of System-Wide Safety Properties of ROS Applications," in 2020 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS). IEEE, 2020, pp. 7249–7254, available: https://ieeexplore.ieee.org/abstract/document/9341085.

[37] R. Toris, C. Shue, and S. Chernova, "Message Authentication Codes for Secure Remote Non-Native Client Connections to ROS Enabled Robots," in 2014 IEEE International Conference on Technologies for Practical Robot Applications (TePRA). IEEE, 2014, pp. 1–6, available: https://ieeexplore.ieee.org/abstract/document/6869141.

[38] B. Dieber, S. Kacianka, S. Rass, and P. Schartner, "Application-Level Security for ROS-Based Applications," in 2016 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS). IEEE, 2016, pp. 4477–4482, available: https://ieeexplore.ieee.org/abstract/document/7759659.

[39] R. White, H. I. Christensen, and M. Quigley, "SROS: Securing ROS Over the Wire, in the Graph, and Through the Kernel," arXiv preprint arXiv:1611.07060, 2016, available: https://arxiv.org/abs/1611.07060.

[40] F. J. R. Lera, J. Balsa, F. Casado, C. Fernández, F. M. Rico, and V. Matellán, "Cybersecurity in Autonomous Systems: Evaluating the Performance of Hardening ROS," Málaga, Spain, vol. 47, 2016, available: https://robotica.unileon.es/vmo/pubs/waf2016.pdf.

[41] M. Mukhandi, D. Portugal, S. Pereira, and M. S. Couceiro, "A Novel Solution for Securing Robot Communications Based on the MQTT Protocol and ROS," in 2019 IEEE/SICE International Symposium on System Integration (SII). IEEE, 2019, pp. 608–613, available: https://ieeexplore.ieee.org/abstract/document/8700390.

[42] J. Kim, J. M. Smereka, C. Cheung, S. Nepal, and M. Grobler, “Security and Performance Considerations in ROS 2: A Balancing Act,” arXiv preprint arXiv:1809.09566, 2018, available: https://arxiv.org/abs/1809.09566.

[43] Y. Liu, Y. Guan, X. Li, R. Wang, and J. Zhang, “Formal Analysis and Verification of DDS in ROS2,” in 2018 16th ACM/IEEE International Conference on Formal Methods and Models for System Design (MEMOCODE). IEEE, 2018, pp. 1–5, available: https://ieeexplore.ieee.org/abstract/document/8556970.

[44] P. H. R. Y. Patel and D. Desai, “Analyzing Security Vulnerability and Forensic Investigation of ROS2: A Case Study,” in Proceedings of the 8th International Conference on Robotics and Artificial Intelligence, 2022, pp. 6–12.

[45] S. Yang, J. Guo, and X. Rui, “Formal analysis and detection for ros2 communication security vulnerability,” Electronics, vol. 13, no. 9, p. 1762, 2024.

[46] N. Goerke, D. Timmermann, and I. Baumgart, “Who controls your robot? an evaluation of ros security mechanisms,” in 2021 7th International conference on automation, robotics and applications (ICARA). IEEE, 2021, pp. 60–66.

[47] V. DiLuoffo, W. R. Michalson, and B. Sunar, “Robot Operating System 2: The Need for a Holistic Security Approach to Robotic Architectures,” International Journal of Advanced Robotic Systems, vol. 15, no. 3, p. 1729881418770011, 2018, available: https://journals.sagepub.com/doi/full/10.1177/1729881418770011.

[48] ——, “Credential masquerading and openssl spy: Exploring ros 2 using dds security,” arXiv preprint arXiv:1904.09179, 2019.

[49] G. Deng, G. Xu, Y. Zhou, T. Zhang, and Y. Liu, “On the (In)Security of Secure ROS2,” in Proceedings of the 2022 ACM SIGSAC Conference on Computer and Communications Security, 2022, pp. 739–753, available: https://dl.acm.org/doi/abs/10.1145/3548606.3560681.

[50] A. S. C. Z. Q. Z. M. Elsabagh, R. Johnson and Z. Lin, “FIRMSCOPE: Automatic uncovering of Privilege-Escalation vulnerabilities in Pre-Installed apps in android firmware,” in 29th USENIX Security Symposium (USENIX Security 20), 2020, pp. 2379–2396.

[51] A.-R. S. L. Davi, A. Dmitrienko and M. Winandy, “Privilege escalation attacks on android,” in Information Security: 13th International Conference, ISC 2010, Boca Raton, FL, USA, October 25-28, 2010, Revised Selected Papers 13. Springer, 2011, pp. 346–360.

[52] Z. L. J. D. R. Li, W. Diao and S. Guo, “Android custom permissions demystified: From privilege escalation to design shortcomings,” in 2021 IEEE Symposium on Security and Privacy (SP). IEEE, 2021, pp. 70–86.

[53] P. N. M. Monshizadeh and V. N. Venkatakrishnan, “Mace: Detecting privilege escalation vulnerabilities in web applications,” in Proceedings of the 2014 ACM SIGSAC Conference on Computer and Communications Security, 2014, pp. 690–701.

[54] S. L. Q. L. T. W. Z. Liu, H. Zhao and Y. Wang, “Privilege-Escalation Vulnerability Discovery for Large-scale RPC Services: Principle, Design, and Deployment,” in Proceedings of the 2021 ACM Asia Conference on Computer and Communications Security, 2021, pp. 565–577.