CSEC 793 Capstone in Computing Security
Project Report

ROS2 PREVALANCE AND SECURITY

May 19, 2020

John Lawrence
Department of Computing Security
College of Computing and Information Sciences
Rochester Institute of Technology
jbl4908@rit.edu

1 Abstract

This paper looks into the successor of the most common robotic management solution, ROS2, to determine its security issues. In doing so, the paper begins by covering the background and technical details on how ROS functions. Once this is complete, the vulnerabilities in ROS2 are explored and explained. These vulnerabilities are divided into two sections: code based and network based.

The code based vulnerabilities stem from the need to have access controls to important documents for ROS2 functionality. As such the solutions lie in better utilizing the layers below ROS2 in the OS level. Network vulnerabilities on the other hand are directly tied to the functionality in ROS2 with their new DSS standard. This paper highlights the 4 main parts of the new ROS2 data exchange between Talker and Listener and the best corresponding attack strategy for each.

This paper concludes by interpret the findings of our research and testing to determine the current position of cybersecurity within ROS2 and, in turn, overall robotic development. The results show that there is more concern for cybersecurity than ever before in the robotic development space, but is limited by ease of use practices and a lack of previous communication between cybersecurity and robotic professionals.

2 Introduction

The current environment of robotic frameworks is ever changing and have become more accessible to developers. We have seen a shift from using pendants and other strict programming requirements to more general systems much more akin to scripting. Because of this solutions such of ROS have taken up a large portion of current robotic projects. With the new successor ROS2 coming out the project as displayed in this paper aims to analyze this new upcoming framework and gauge the vulnerabilities it has along with the impact it will have on the robotic community. Using this we can accomplish the end goal of determining an overall risk posed by deploying ROS2 in future robotic projects.

The improvement and understanding of frameworks requires the study and conclusions of many separate analyzing parties. Through using tools to study the current deployment of ROS an idea of the future impact ROS2 will have can be derived. Vulnerabilities can be determined through testing with existing tools, such as packet sniffing, secure code analysis, and risk frameworks. Thus the goal of this project is to use tools and vulnerabilities to determine the level of security in ROS2 and the variance it can have depending on its deployment. This will be done using many of the aforementioned tools and study along with an in depth analysis and development of attacks against ROS2 network traffic on various security settings.

3 Background

Robotic systems have been developing with the new technologies in other fields of computer science and technology. One of the major breakthroughs has been the standardization of robotic protocols with projects such as ROS by the Open Source Robotics foundation. This framework in particular has gained notable traction in academia, being used by many universities and groups such as NASA [1]. What makes ROS stand out is its adoption and usage in industry as well, with Amazon offering a ROS solution in their AWS suite [2].

ROS is developed to support a single consolidated robot system or a distributed system. This is due to the way it handles the authoritative server and handling tasks. The main server, called ROScore, registers nodes which can be any device with an internet connection and the ROS libraries. In this way, the problem of scheduling tasks is taken on by the node which will be programmed with a specific function[3]. The node will then send any information it generates during its operation to ROScore over a 'topic'. The topic can then be viewed by any other node making the flow of information and tasks complete.

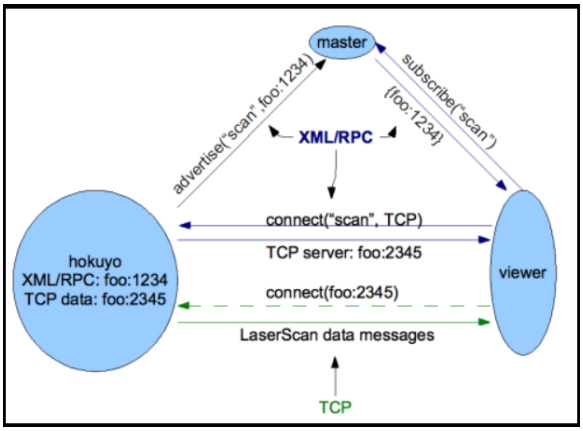

Figure 1: An Example of ROS Communication
[4]

Because of the scope of projects ROS has been involved with, security has become a concern. ROS was not developed with security in mind, notably lacking encryption on communication and authentication methods. During their review of the security in ROS protocols, Dieber et. al stated that the changes needed to be made in order to secure ROS would entail all modules being rewritten and recompiled[5].

The successor to ROS, ROS2, was nearing public release in 2017. In the meantime current ROS systems had to be secured somehow and the question of legacy security support was raised. The project SROS was released to provide basic encryption through a public key infrastructure using X.509 certificates, something similar to what would be seen in ROS2. This new library for ROS would ensure the remote procedure calls and communication

would be securely encrypted setting and is considered to have met the concerns[6].

To make improvements to both system efficiency and security the Open Source Robotics Foundation released the first public distribution of ROS2 in December 2017. It bases its security on the Data Distribution Service (DDS) security specification, implementing a public key infrastructure[7]. It has two modes of operation, the default being permissive which looks for security files but will run without them if they cannot be found. It also has an environment argument $ROS_SECURITY_ENABLE which can be set to false to turn off all added security.

The majority of security changes to ROS2 in comparison to ROS stems from the adoption of the DDS specification. To meet the new DDS spec it functions with a UDP multi cast over a selected group/domain[8]. This means that you can no longer have a node on a different LAN, perhaps intended to remove having nodes connect over internet. However, this also means there is no longer an authoritative ROScore for ROS2. The reason being that since every node is multi casting anyway there is nothing to parse and control.

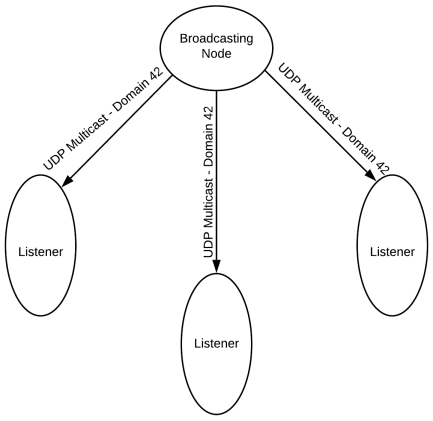

Figure 2: An Example of ROS2 Communication

DDS seeks to handle 4 relevant categories of threats: Unauthorized subscription, Unauthorized publication, Tampering and replay, Unauthorized access to data. To do so it requires the providing of: Confidential data samples, integrity of data, authentication and authorization of endpoints, message and data origin authentication, with an optional requirement for non-repudiation[9].

For secure communication to begin through DDS the two node must first have a matching set of 'Participant Security Info' sent in the **ParticipantBuitinTopicData** topic packet during communication[9]. The parameters for this security defining packet are provided in the Participant Security Attributes Table. Due to this exchange occurring before security has been established between ROS2 nodes the security of this exchange is entirely dependent

on pre-existing security controls unrelated to ROS2.

Member	Type
allow_unauthenticated_participants	Boolean
is_access_protected	Boolean
is_rtps_protected	Boolean
is_discovery_protected	Boolean
is_liveliness_protected	Boolean
plugin_particpant_attributes	AttributesMask
ac_participant_properties	PropertySeq

The controls will influence the usage of certain security control settings, such as encryption and digital signatures, to accompany network communications. This network communication uses Real-Time-Publish-Subscribe (RTPS) protocol, a specification that includes the discovery process along with the continuation of handshakes throughout the connection lifespan to ensure reliability.

RTPS is platform independent and as such is not constrained to being used only in OS specific implementations. It does this by maintaining its own specific classifications of objects on the network known as RTPS Entities and Classes. For the purposes of this paper all legitimate nodes used during the testing can be considered RTPS Endpoints as they are either a writer or a reader [10].

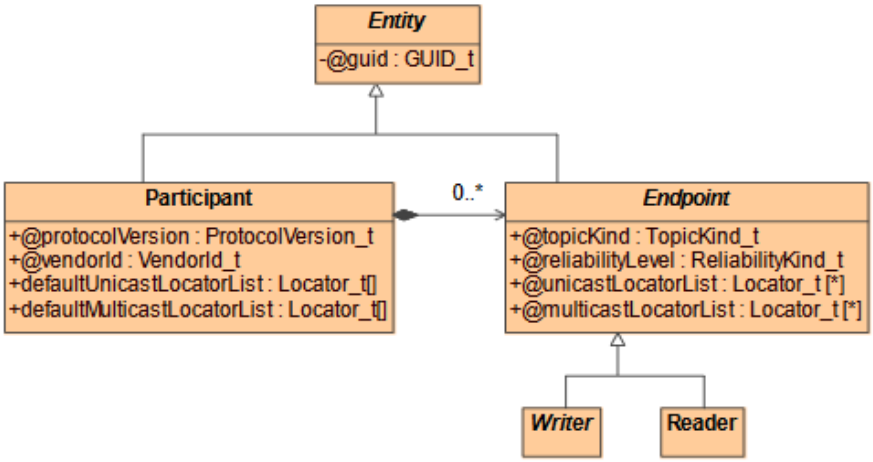

Figure 3: A diagram of the endpoints classification and attributes, some of which are shared with the participant and entity class.

While the DDS and RTPS specification do go further into how communications should be handled, this is intentionally hidden by ROS2. This is done by taking the most of the heavy work done in the new DDS specification and sliding it into the pre-existing template of 'publisher' and 'subscriber' of the old ROS libraries. In doing so a ROS user can seamlessly transition to ROS2 and the newer security controls will not constrain them [11].

While DDS, ROS2, and RTPS specifications are all intended to be platform agnostic the developed implementations by various parties have their differences. These are recognized by the ROS2 developers and they support a variety of "vendors" or implementations of DDS to be used with ROS2 [12]. These implementations create noticeable differences in how 'heavy' the deployment is on a system. For example, the OpenSplice implementation is generally considered to handle large deployments the best with low latency however it also requires 6 times as many threads as the other popular implementations provided by Connext and FastRTPS [13]. Because of these differences certain hardware or low-level OS attacks may present more or less risk depending on the exact implementation used.

In the coming literature review the previous analysis and study of these mechanics along with their security impact will be presented.

4 Literature Review

Analysis of ROS2 has been ongoing, with the security of the new encryption algorithms being often tested. In a paper by Jongkil Kim et al. they analyzed the handshake and encryption used by the ROS2 implementation of DDS. According to their findings, "the correctness of the security protocol is verifiable" for the handshake while the library they are using for SSL is "currently deficient such as zeroization of secret data"[14]. Based on these findings and the continuous work on ROS2 and ROS it is safe to say there is a further need for review and improvement in the robotic security space.

This weakness in the OpenSSL libraries used in ROS2 was further analyzed by DiLuoffo et al. during an analysis on the security implications of security weaknesses in layers below ROS2. It was found that the weaknesses in OpenSSL libraries were prevalent enough that they could be modified with no detection by ROS2, leading to what is considered secure communication by the nodes to not be secure[15]. Furthermore, weaknesses in default file system policy along with the usual installation location of ROS2 has certificates unprotected. This, along with the security configuration files, can be changed without being detected by ROS2. Giving an adversary with access to a ROS2 device the ability to substitute security settings and credentials with their own with no detection mechanisms available from ROS2.

The threat presented to ROS2 is certainly not lost on them, as the Open Robotics Foundation presented their own Threat Model for the ROS2 system. Their work presents a comprehensive look into the threat actors, entry points, and vulnerabilities along with their combinations. The dataflow used during their testing is based on the TurtleBot3 demo included with most versions of both ROS and ROS2 along with MARA to represent a modular industrial robot. From these results they determined a total of 27 vulnerabilities, 2 exploited physically and 25 being exploited over the network [16]. Of these 6 were rated low, 13 medium, 6 high, and 2 critical in accordance with the Robot Vulnerability Scoring System (RVSS)[17].

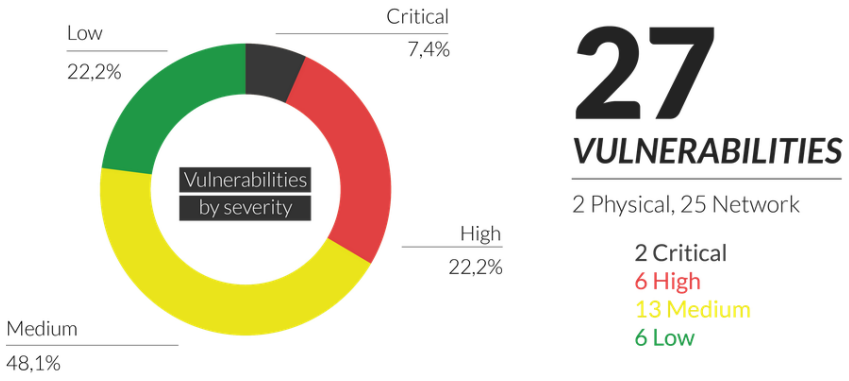

Figure 4: Vulnerability Results of the ROS2 Threat Model Analysis

DiLuoffo et al. in a seperate paper concerning the holistic approach to ROS2 security took a different approach to developing a threat model. In their model they defined 3 seperate adversarial models that could be used to account for most vulnerabilities. This could be done by putting each entry point in either a Software, Cognitive, or Side Channel adversarial model. Software includes the entry points associated with application logic like the OS and ROS2, DDS, and DDS security drivers. Cognitive layer are entry points concerned with the written programs themselves and how they modify received information. Finally side channel concerns the sensors themselves and how through methods such as transduction attacks received input can be spoofed. [18]. They deploy a variety of security plugins to account for the vulnerabilities found during their own analysis and the threat model created by the Open Robotics Foundation. The result was a clear loss of efficiency seeing latency increase, throughput decrease, and average Mbps requirements increase.

Table 4. Security measurement comparison.

Security enabled	Latency (average µs)	Throughput (average packets/s)	Speed (average Mbps)
Plain	260	70,772	35,669
Full	1363	14,382	72,485

Figure 5: Loss of efficiency from security plugins.

These findings do compliment the reasoning for ROS2 to not have most security enabled by default for functionality and ease of use reasons. A solution posed by them to this problem is taking advantage of the segmentation of domains in ROS2. This would be done by specifying plugins per domain to match the security needs, instead of using a one-size-fits-all domain.

The analysis and testing done in this paper will differ by focusing primarily on the network based communication between nodes and the vulnerabilities to confidentiality, integrity, and accessibility present. This will be gathered based on both the default deployment scenario which lacks many of the security controls as well as a security enabled deployment that is set to run in permissive mode. These results and recommendations will be less conceptual than the previous work and won't deal with having to cover all entry points such as the file system leading to greater focus on the topic of network security.

5 Project Idea and Implementation

After analyzing the security of ROS before and writing a paper on its vulnerabilities as of 2019, it was decided to look to its successor ROS2 to determine the future of security vulnerabilities in robotic frameworks. The vulnerabilities found in ROS would be the first elements to be analyzed for. Such vulnerabilities include plain-text messaging and a lack of encryption between nodes and an authoritative server.

Based on these previous findings, it is reasonable to expect not all issues of ROS2 being solved. This analysis is perhaps even more necessary as the standard for ROS2 (DDS) has changed between ROS and ROS2. This distributed system is also now using UDP for most communications. This lack of connection could make the transfer of information more prone to interference. Such interference could involve replaying packets to full denial of the ROS2 service.

To analyze ROS2 properly a test range was setup using 3 hosts. 2 of these hosts would be considered benign or legitimate ROS2 nodes using the 'dashing' release of ROS2. They will also have all the SROS2 libraries, which are the security addons to ROS2 that meet DDS security standards. The last of these 3 hosts would serve as a mock attacker. The will also be capable of downloading ROS2 'dashing' libraries for the purpose of writing malicious node code. The malicious actor will be capable of listening to the communication from the talker and aim to interfere with communications. This communication will be analyzed in two settings. The first setting is base install, with security setting defaults being mostly turned off. The second setting of testing will be when DDS security settings are turned on according to the base demo settings provided by the SROS2 security team [19].

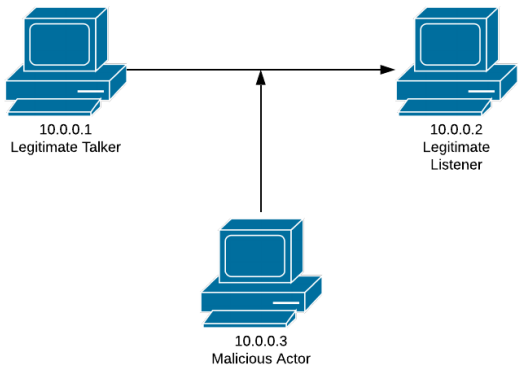

Figure 6: Topology of the test range

The settings for base install do not specify a specific group to multi cast to, security keys, or any security strategy. For the recommended security settings it is important to note that they differ depending on which provider of RTPS you are using. The default provider and

what will be used for this implementation is FastRTPS which comes standard with most ROS2 installs with the SROS2 package. ROS2 environment variables are then set to specify security settings and must match proper case. The variables set in this implementation are presented in the table below.

Environment Variable	Value
ROS_SECURITY_ROOT_DIRECTORY	[Directory chosen by administrator during keygen]
ROS_SECURITY_ENABLE	true
ROS_SECURITY_STRATEGY	Enforce
RMW_IMPLEMENTATION	rmw_fastrtps_cpp

The listening device will be running what is known as a 'subscriber' node in the ROS environment. To do so it will first setup the message type used, in this case we specify string for ease of use. And then we specify a topic that must match the topic name the publisher will send. From there we run the listener and wait.

The talker device will be running a 'publisher' node. It will have its message type and topic set to the same as the subscriber so that they are compatible. From there communications should start broadcasting. To ensure that communications are not interrupted we may choose to specify a UDP domain. We will go more into detail on the specifics of this communication and its security implications in the data analysis section.

The idea for how to exploit this comes down to exploiting the lack of authentication most likely found within this communication schema along with the lack of integrity. For this reason, it should be possible for a malicious host to not only hear all communication but also control it in any way they see fit.

Some attacks that will be utilized involve ARP cache poisoning, man in the middle, and spoofed source addresses. Using these tactics we can prove that a ROS2 environment cannot rely on the location packets are arriving or being sent to along with not being able to rely on the data sent. This is accomplished by crafting spoofed ARP frames using scapy with python 3. Paired with this will be attempts to craft spoofed packets in the IGMP (Internet Group Management Protocol) sequences. These sequences occur when the node first starts and when the node ceases operation.

The final attempted attacks will involve interrupting the data transactions between nodes. Doing this will involve spoofing packets to ruin the integrity of the data, taking a legitimate talkers space by either taking over it's addresses or creating another node that copies it, and finally by attempting to produce enough packets to limit operation of the listener through a DoS style attack.

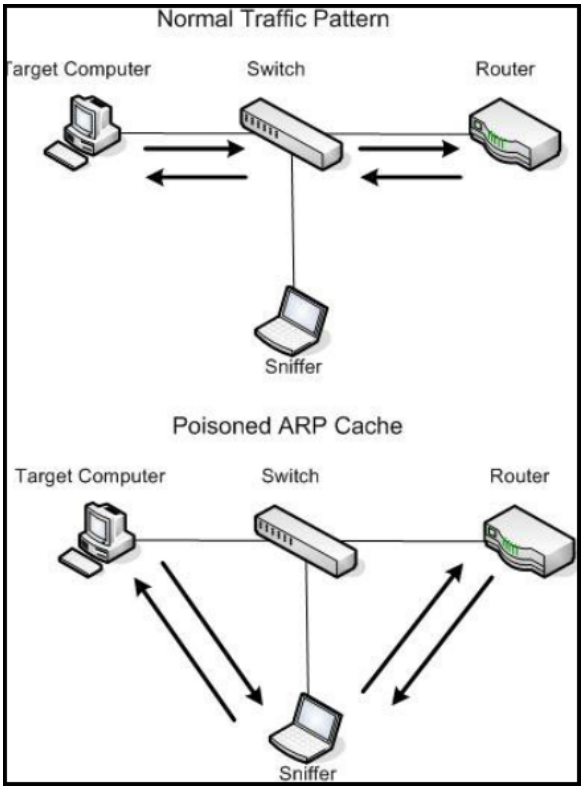

Figure 7: The Results of a Poisoned ARP Cache [20]

6 Testing and Experiments

In this section, describe your testing and experiment design and setup, and conduct the testing and experiments, and generate data.

Need to explain why your experiment design will do what is supposed to do, and describe the expected the result, and how the result may validate your ideas and/or support your project.

The testing began by setting up two separate environment based on the implementation standards. These two environments were differentiated primarily on the level of security they implemented into ROS2. Our first environment, hereby called environment 1, was a base install of ROS2 on two machines with a machine in the middle to represent the attacker [Fig 6]. Because of this, it was expected for this environment to lack critical security controls and methodologies compared to environment 2. Environment 2 was a ROS2 install on two machines with a machine in the middle as well [Fig 6]. The primary differences were the addons known as SROS2 (Secure ROS2) [19] being implemented on the two legitimate ROS2 nodes. The settings were configured to create encrypted communication using a public key infrastructure along with the selection of heading configuration settings as discussed in the background and implementation.

The goal of the initial testing phase was to inspect both the base code used in most ROS2 nodes along with capturing the expected traffic respective to the two environments. In doing so we could understand the whole process by which security can be tested, both on the hosts and on the network.

6.1 ROS2 Code Security

The core of all ROS2 systems is the Talker-Listener system, where a node generally will be focused on either producing and transmitting information or listening and interpreting that information for a various robotic subsystem to use. The Talker is usually a sensor of some sort, such as a camera, that is either capable of its own computational programming or is connected to a computer to perform that. A Talker node file is generally broken up into 2 sections, the talker class and the main function. The talker class is where most of the defining features of the system are made and the point where attackers would most likely want to modify.

Talker Element	Features
Class Talker(Node)	Functions: __init__(self) timer_callback(self)
main(args)	Initialize Talker node and 'spin' it up.

By modifying where the init is pointing to and what data type it is using the system could be entirely modified to produce spoofed values quickly. Because the key is created

before running the code and no integrity check is made, this leads to a possible vulnerability in integrity.

The Listener is comparatively less integral to the overall system, being made up of similar elements but with less settings and operations inside of them. Furthermore it is a single end node which produces less impact than the producing Talker that sends to many end nodes.

Element	Features
Class Listener(Node)	Functions: __init__(self) chatter_callback(self)
main(args)	Initialize Listener node and 'spin' it up.

For this reason should an organization setup security levels for file access the Talker should be considered to have a higher security requirement for access than the Listener files. This will of course depend on what robotic subsystem the Listener is interpreting for.

The code did not have to be changed between the two environment for testing. For this reason the security concerns for this section can be considered SROS agnostic and are more reliant on the layers below it (such as physical and OS security) for controls.

6.2 ROS2 Network Security

The differences in testing ROS and ROS2 on the basis of security are made clear when analyzing their network security. This is due to the shift towards the DDS security specification which brought many new protocols to replace the older and proprietary ROS protocols. The ROS team likes to maintain however that the Talker-Listener system is still present which is partly true for those who only work on the code level.

6.2.1 Attacks Not Requiring Key Interception

To begin we created environment 1 and setup a Wireshark capture inline between the two devices. Environment 1 is the base install, with no added SROS2 libraries to augment ROS2 network communications. This capture was started before both the Talker and Listener began running in order to catch the initial broadcast messages. The specific packet we are looking for is an IGMP join request as specified by the standard.

This packet indicates that a node has joined a group and is ready to start receiving messages from all talkers in that group. It is not encrypted in either environment 1 or 2 as it occurs before key setup and verification between the nodes and as such is always able to be viewed. This creates a vulnerability from relay attacks, specifically those which slightly modify the packet to have other addresses join the group against intended use cases.

While the join IGMP packet in of itself can be seen as a vulnerability in both environments it also produces a situation that can be taken advantage of by malicious actors listening on the network. This opportunity lies in the ARP frames sent out shortly after an IGMP join packet is sent.

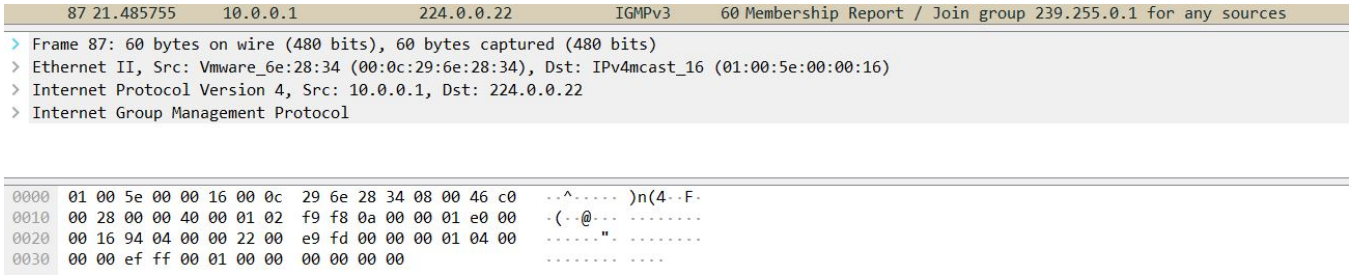

Figure 8: A standard ROS2 IGMP join group packet

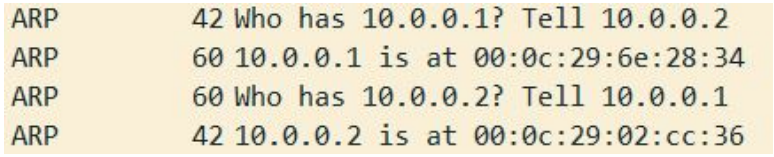

Figure 9: ARP requests by legitimate ROS2 talker and listener

Basing the range of time taken for these ARP frames to occur from our trials, this gives an attacker 2-4 seconds to send their own ARP poisoning attack before the nodes begin. The knowledge needed to successfully perform this attack requires just the Listener and/or Talker IP. The IP information can be gathered without needing to query the machines by listening for IGMP join requests. The Talker and Listener must publish one to join the network and it includes their IP. From there an attacker sends gratuitous ARP requests saying the MAC address for the IPs is at the attackers machine MAC address. From there the attacker has successfully created a Man-in-the-Middle style attack and controls the flow of packets between Talker and Listener.

An attack only available to attackers in an environment like environment 1 is IGMP leave spoofing. The attack is nearly identical to IGMP join spoofing as mentioned earlier, requiring only a few bits of a join packet to be modified in order to change it to a leave packet. In doing so an attacker can continuously block a listener from joining by sending a crafted leave packet as soon as a join packet is found.

The reason this is not as available to environment like environment 2 is because the leave will be encrypted with the key. As mentioned before the join is not encrypted for either environment as the key setup and verification. had not occurred yet. Since the leave occurs after an attacker will need to acquire the key first before.

Finally, in environment 1 all RTPS traffic is unencrypted including all data transmission between the Talker and Listener. Because of this even the data portion of the exchange is just as susceptible to confidentiality and integrity vulnerabilities as the join and leave IGMP portion.

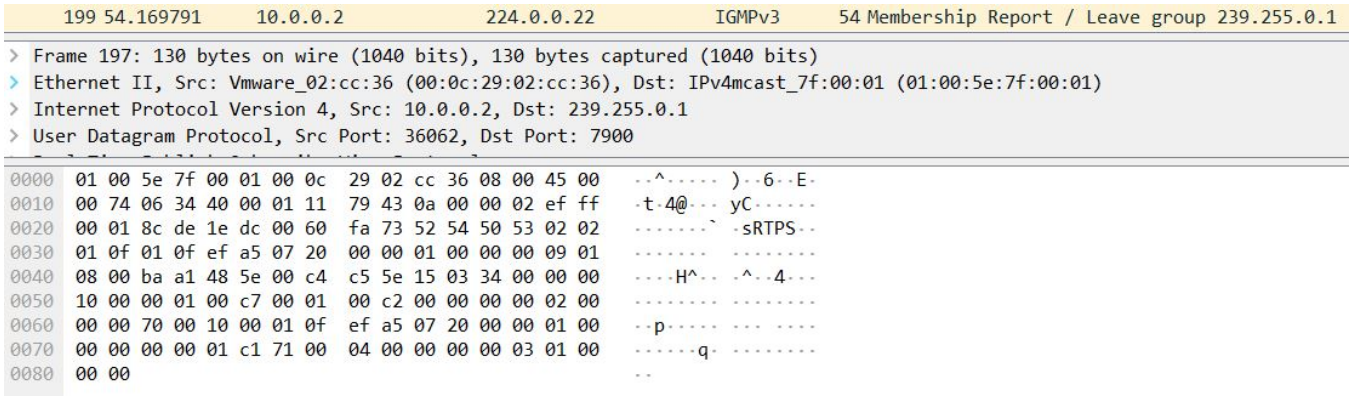

Figure 10: A standard ROS2 IGMP join leave packet

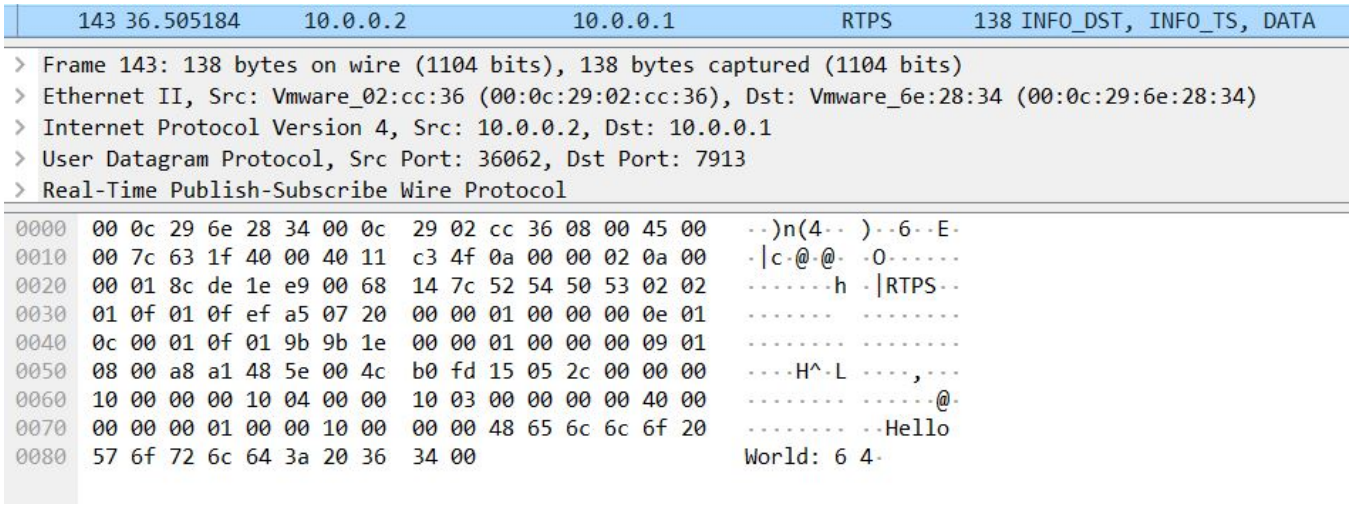

Figure 11: Environment 1 Data Exchange Packet

6.2.2 Attacks Requiring Key Interception

According to the SROS2 guides and documentation there is no native means of transferring keys between remote listener and talkers. Their recommendation in the guides for Linux mentions using SCP to transfer the keys. This of course begs the question of if SCP is a secure enough channel for key exchange why not use it also to secure data transmission? Regardless, for environment 2 we will assume key exchange is done using SCP as it is the most often recommended protocol.

For these attacks we will not cover how key interception or acquisition is performed and will base the attacks on if you already have the key.

"Then, we need to copy some keys to oldschool (listener node) to allow SROS 2 to authenticate and encrypt the transmissions. Since the keys are just text files, we can use scp to copy them"

Figure 12: SROS 2 Linux Instructions on Sharing Keys [19]

The first attack this opens up is the IGMP leave attack mentioned earlier for environment 1. To do so easily you can capture the join IGMP packet which is still unencrypted in environment 2. Modify the 41st byte of the join packet by increasing it by 1 and then modify the 47th byte by decreasing it by 1. This produces a leave packet for that node. Then encrypt it with the key and send over the existing RTPS connection.

Join Packet
0000 01 00 5e 00 00 16 00 0c 29 6e 28 34 08 00 46 c0
0010 00 28 00 00 40 00 01 02 f9 f8 0a 00 00 01 e0 00
0020 00 16 94 04 00 00 22 00 e9 fd 00 00 00 01 04 00
0030 00 00 ef ff 00 01 00 00 00 00 00 00

Leave Packet
0000 01 00 5e 00 00 16 00 0c 29 6e 28 34 08 00 46 c0
0010 00 28 00 00 40 00 01 02 f9 f8 0a 00 00 01 e0 00
0020 00 16 94 04 00 00 22 00 ea fd 00 00 00 01 03 00
0030 00 00 ef ff 00 01 00 00 00 00 00 00

Figure 13: Example of a Join packet and it's corresponding Leave packet

If you want to use a single join packet to produce a leave packet for any IP you can do so by modifying the hex between bytes 27-30. This range contains the IP address of the source of the IGMP packet, which in Fig. 13 is 10.0.0.1. Being able to successfully perform this attack allows an attacker to control which nodes are allowed to operate on the ROS2 system in an SROS encrypted system.

7 Conclusions

In this paper we analyzed the security stance of of ROS2 and covered the implications it would have for the future relationship between robotics and cybersecurity. Based upon the findings of this paper along with previous papers in this area of research the concept of security is becoming more seriously consider than in previous robotic framework projects.

This new level of consideration has not yet translated into sufficiently secure systems however, with multiple weak spots in implementation that would greatly benefit from a security control. Even without a security control a standard policy on important security processes such as key exchange would be a good first step.

Most of the security issues found within this paper stem from the new structure of network communications due to the adoption of the DSS standard. The figure below summarizes both the structure and the corresponding attack strategy for each section.

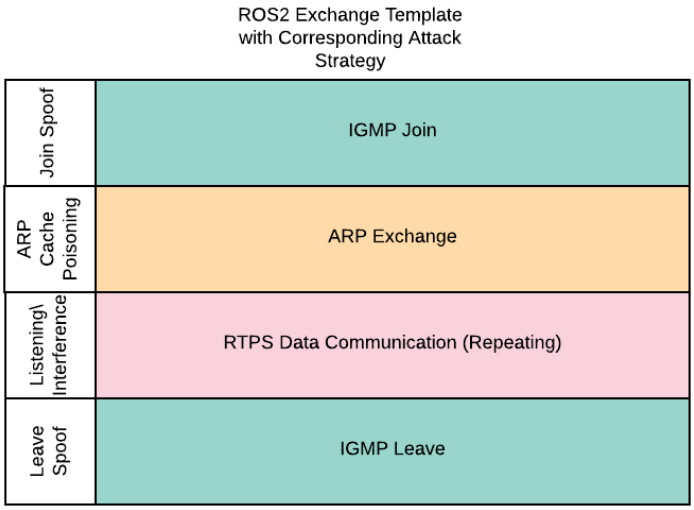

Figure 14: ROS2 Exchange and Attack Structure

The key source of these vulnerabilities can be tied to the ROS2 philosophy of get it to work fast and easy upon deployment. This certainly differentiates itself in a good way from other robotic frameworks, however as demonstrated in the paper it has security related side-effects.

By The Open Robotics Foundation conducting a vulnerability assessment of the ROS2 framework in late 2019 they have shown a motivation to begin taking cybersecurity with a higher level of concern. By partnering with external security initiatives both with and without robotic focuses it is likely they can reduce much of the cybersecurity risk posed to users of ROS2 and in turn better secure the future of robotic development.

7.1 Key Findings

- ROS2 reworks and standardizes it's security stance with adoption of DSS standard.
- Currently ROS2 does not include access control settings for ROS2 node code.
- Based upon both this paper's testing and The Open Robotic Foundations testing ROS2, the standard for robotic frameworks, contains multiple high severity vulnerabilities.
- ROS2 uses a PKI infrastructure for encryption, but does not have a standard policy for key distribution
- Through spoofing, cache poisoning, and interception ROS2 has multiple and easy methods for attacking the system's CIA triad (Confidentiality, Integrity, and Availability)
- ROS2 is ultimately a good step however with new efforts being made to follow modern security standards and perform vulnerability assessments.

8 Acknowledgment

I thank my project advisors Professor Ziming Zhou and Professor Sumita Mishra for their support and guidance during the production of this paper. I would like to thank the Rochester Institute of Technology Libraries for their assistance in the Literature Review and Background portion of this paper. Finally I would like to thank my family for their continued support in my academic endeavors.

References

[1] B. Gerkey, "Ros running on iss." [Online]. Available: https://www.ros.org/news/2014/09/ros-running-on-iss.html

[2] "Announcing aws robomaker: A new cloud robotics service," 2018. [Online]. Available: https://aws.amazon.com/about-aws/whats-new/2018/11/announcing-aws-robomaker-a-new-cloud-robotics-service/.

[3] I. Nielsen, Q.-V. Dang, G. Bocewicz, and Z. Banaszak, "A methodology for implementation of mobile robot in adaptive manufacturing environments," *Journal of Intelligent Manufacturing*, vol. 28, no. 5, pp. 1171–1188, 2017.

[4] V. Mayoral, "Ros technical overview," Jun 2014. [Online]. Available: http://wiki.ros.org/ROS/Technical Overview

[5] B. Dieber, B. Breiling, and S. Rass, "Security for the robot operating system," *ResearchGate*, Oct 2017.

[6] R. White, H. Christensen, and M. Quigley, "Sros: Securing ros over the wire,in the graph, and through the kernel," *HUMANOIDS 2016 Workshop: Towards Humanoid Robots OS*, Nov 2016. [Online]. Available: https://arxiv.org/pdf/1611.07060.pdf

[7] K. Fazzari, "Ros 2 dds-security integration," Oct 2019. [Online]. Available: https://design.ros2.org/articles/ros2_dds_security.html

[8] "Overview of ros 2 concepts," Feb 2020. [Online]. Available: https://index.ros.org/doc/ros2/Concepts/Overview-of-ROS-2-concepts/

[9] "Dds security specification version 1.1," Jul 2018. [Online]. Available: https://www.omg.org/spec/DDS-SECURITY/1.1/PDF

[10] "The real-time publish-subscribe protocol (rtps) dds interoperability wire protocol specification version 2.2," Sep 2014. [Online]. Available: https://www.omg.org/spec/DDSI-RTPS/2.2/PDF

[11] D. Thomas, "Ros 2 middleware interface," Sep 2017. [Online]. Available: http://design.ros2.org/articles/ros_middleware_interface.html

[12] "Ros 2 and different dds/rtps vendors," Feb 2020. [Online]. Available: https://index.ros.org/doc/ros2/Concepts/DDS-and-ROS-middleware-implementations/

[13] Y. Maruyama, S. Kato, and T. Azumi, "Exploring the performance of ros2," *Proceedings of the 13th International Conference on Embedded Software - EMSOFT 16*, 2016.

[14] J. Kim, J. M. Smereka, C. Cheung, S. Nepal, and M. Grobler, "Security and performance considerations in ros 2: A balancing act," *Commonwealth Scientificand Industrial Research Organization*, Sep 2018. [Online]. Available: https://arxiv.org/pdf/1809.09566.pdf

[15] V. DiLuoffo, W. R. Michaelson, and B. Sunar, *Credential Masquerading and OpenSSL Spy: Exploring ROS 2 using DDS security*, 2019.

[16] "Ros 2 robotic systems threat model," Oct 2019. [Online]. Available: https://design.ros2.org/articles/ros2_threat_model.html

[17] V. M. Vilches, "Towards an open standard for assessing the severity of robot security vulnerabilities, the robot vulnerability scoring system (rvss)," *Alias Robotics*, Sep 2019. [Online]. Available: https://arxiv.org/pdf/1807.10357.pdf

[18] V. DiLuoffo, W. R. Michaelson, and B. Sunar, "Robot operating system 2: The needfor a holistic security approach torobotic architectures," *International Journal of Advanced Robotic Systems*, 2018.

[19] "Try sros2 in linux," Nov 2019. [Online]. Available: https://github.com/ros2/sros2/blob/master/SROS2$_L$*inux.md*

[20] C. Sanders, "Understanding man-in-the-middle attacks - arp cache poisoning (part 1)," Sep 2017. [Online]. Available: http://techgenix.com/understanding-man-in-the-middle-attacks-arp-part1/