# 20 On the Security of Robotic Applications Using ROS

*David Portugal, Miguel A. Santos, Samuel Pereira, and Micael S. Couceiro*

## CONTENTS

## INTRODUCTION

Robots are becoming a part of our everyday life in society. Besides the wide use in industry, nowadays robots have an important role in household, medical, space, and military applications. Recent research has shown significant innovation in other fields as well, such as social robotics, field robotics, and intelligent vehicles. The widespread use of robots in these markets has been brought forward due to recent advancements in control, artificial intelligence (AI), decision-making, robot learning, localization and mapping, motion planning, computer vision, sensors development, and other relevant areas [1].

Naturally, with the growing proliferation of robots in our society arises the concern of security. However, this is an often overlooked issue in robotic systems, as the focus is commonly placed in robot functionality and innovation. Unauthorized access to a robot, or a network used by robots, may seriously compromise the system, potentially leading to unacceptable consequences, such as putting in danger humans who share the environment with the robot(s) [2].

The potential benefits of robotics are clear and widely documented. However, they also introduce new security and privacy concerns. Robotic systems are built on top of traditional computing platforms, being connected to actuators, and other sensors and hardware, such as cameras. Thus, not only are robots vulnerable to the same cyberattacks as traditional computing systems as they become networked and connected to the Internet, but they also unveil a whole new set of security issues that can result in privacy concerns if hacked or, even worse, causing physical harm.

The mass adoption of robots is likely to increase the possibilities of attacks. This is particularly problematic in defense, medical and other critical fields involving humans [2]. However, the consideration of security and privacy risks has clearly not been a priority for roboticists neither in academia nor in the industry. With the new risks introduced by networked robots, now is the ideal time to take a deep look at the current security practices before robots with serious flaws become ubiquitous [3].

The European Commission (EC) has been following the legal and ethical issues raised by new technologies, and recently proposed a series of recommendations on civil law rules on robotics and AI [4]. In the interest of safety and privacy, the EC created a code of ethical conduct for researchers and developers of robotic technology. The code is voluntary, used to provide guidelines for those involved in the development and use of robotics and AI technology to comply with set standards. They also proposed the creation of a European agency for robotics and AI, whose primary purpose would be to supply public authorities with information regarding technical, ethical, and regulatory issues in these fields.

Despite recent efforts, EC-funded R&D Robotic projects often overlook security and privacy issues. This is also a widespread practice in robotics research and even in available commercial robotic solutions (*cf.* section "Background"). Therefore, building secure robot applications in general purpose programming environments is imperative. This is one of the motivations of the ongoing STOP R&D project,* which aims at deploying a commercial security system of distributed and cooperative robots by 2020. In spite of the targeted ground-breaking real-world application, the team, comprising public and private entities, has been mitigating common mistakes made during the design of complex systems, such as multi-robot solutions [5].

Numerous different robotic frameworks have been developed in the last decades, whereas the Robot Operating System (ROS)† [6] is undoubtedly the most popular. Its series of features led the adoption of ROS around the world, being the closest one to become the standard middleware that the robotics community urgently needed. Thus, several robots from vendors and research institutes run ROS. The use of Internet-connected robots is still limited to research laboratories in the majority of cases, but their application in the future seems inevitable [7]. In this chapter, a deeper look is taken into the security issue in robotic applications using ROS. We overview and discuss several initiatives for securing ROS, and we provide general security measures to be implemented to avoid devastating privacy and security consequences, based on lessons learned in the STOP project.

In the next section, we overview seminal work on the effort to promote security and privacy in robotics, and in section "Security Concerns in ROS," we identify known security issues in ROS. Afterwards, in section "Initiatives for Securing Robot Applications on ROS," we present and discuss several recent initiatives that augment the security levels in ROS applications. Additionally, overall measurements and recommendations are described to protect data in robotic projects, and finally, the chapter ends with conclusions and future work.

## BACKGROUND

The discussion of robotic safety and security dates back to Asimov's Three Laws of Robotics in 1950 [8], which state that:

1. A robot may not injure a human being or, through inaction, allow a human being to come to harm.
2. A robot must obey the orders given by human beings except where such orders would conflict with the first law;
3. A robot must protect its own existence as long as such protection does not conflict with the first or second laws.

However, with the progressive advancements in robotics, researchers have shown that these laws alone are not sufficient to govern robot behavior [3].

The term *Cryptobotics* has been proposed by Morante et al. [2] as a unifying term for research and applications of computer and micro-controllers' security measures in robotics. In their work,

* http://stop.ingeniarius.pt
† http://www.ros.org/

the authors highlight the need to implement encrypted communications and to analyze the impact of encryption in real-time performance, thus raising awareness for developers to determine whether it is viable to integrate these mechanisms depending on their specific use case. Finnicum and King [9] identify key factors for improving security and privacy in robotic applications, such as user identification, correctly exposing the application privileges, and user control of their own privacy. Additionally, the authors propose a layered software architecture grounded on a security kernel for robotic applications. Moreover, Adi [10] focuses on security requirements for designing robot identification technology, analogously to human societies, thus enabling secure transactions between robots with unique provable identities.

In Reference 11, a security enhancement to the Interoperable Telesurgery Protocol (ITP) is described. It addresses four key security aspects: communication, authentication, authorization, and security policy development and enforcement, based on published standards to meet the stringent requirements of telesurgical robotics. Despite ensuring privacy and information integrity on individual communication channels, the communication between a telesurgical master and slave requires a highly robust, redundant, and secure communication link. The authors refer that a protocol to meet these requirements still does not exist.

On the other hand, Yong et al. [12] identified risks to families with children when using wirelessly controlled robot toys, and propose risk mitigation solutions to toy companies and consumers at different levels. These include: (i) VoIP protection strategies (parental control and anonymity); (ii) secure remote control connections (encryption and authentication); (iii) wireless connections (home network Wi-Fi security features); (iv) parental control strategies embedded in commercial robot toys; (v) camera protection strategies (sound effects, authentication); (vi) shared risk perspective (government and toy company liability, require legal obligations to protect the safety of consumers); (vii) general protection strategies (strong passwords, require wired connection for initial setup, avoid ad hoc mode, use updated firmware, avoid default ports, etc.).

Security has also been a matter of concern in multi-agent networks. For instance, Caiti et al. [13] described a methodology for secure cooperation within a network of autonomous mobile underwater sensors connected through an acoustic communication network. The algorithm proposed is intrinsically robust: with loss of communication among the vehicles, the coverage performance, that is, the mission goal, is degraded but not lost. The proposed form of graceful degradation provides a reactive measure against Denial of Service (DoS). To ensure trustworthiness in the available information, a security suite based on the group communication paradigm has been designed, with the goal of minimizing the information exchange among the vehicles, and reducing the communication overhead introduced by security in terms of number and message size. The authors introduce two main data security services: Secure Dispatching Service (SDS) and the Key Management Service (KMS). SDS is responsible for protecting confidentiality and authenticity of messages by encrypting and decrypting them, as well as generating and verifying proofs of their authenticity. KMS is responsible for revoking the current key and distributing a new one either periodically or upon a vehicle leaving.

Moreover, security challenges in swarm robotics have been identified in Reference 14. These include: (i) resource constraints in storage, communication bandwidth, computational restrictions and, most importantly, energy, due to the small sizes of devices; (ii) physical capture and tampering of robots, which may influence the swarm behavior; (iii) challenging control of swarm approaches due to the lack of hierarchic points of control and inherent distributed decision making; (iv) employment of different types of explicit and implicit communication, which can be jammed, intercepted, or otherwise disturbed; (v) physical mobility, which has an impact on entity authentication and additional security issues, (vi) identity within the swarm to guarantee and confirm data origin, legitimate communication, confidentiality, integrity, and availability; (vii) cryptographic key management due to the dynamic and interactive nature of a swarm; (viii) intrusion detection due to the autonomous nature of robots and collective emergent behavior; (ix) managing learning by adaptation due to the introduction of changes by malicious entities, causing robots to adapt in an undesired way. These challenges identified by the authors, even though addressed within the context of swarm robotics, transfer to most distributed multi-agent networks. Therefore, addressing them early would prevent undesirable consequences for many applications of this type of technology.

Under a different context, [15] reports the numerous vulnerabilities that smart home appliance users are facing, showing how networking monitoring tools and regular distributed denial-of-service (DDoS) attack techniques can easily be used to attack the Internet of things (IoT) network for illicit purposes. A promising TOR*-based anonymous communication approach was implemented to help smart home appliance users protect their privacy and make the smart home appliance system more secure from cyber attacks. Results show that this is a suitable approach to solve recent security problems in TCP-based smart home appliances.

The literature also presents several works which identify vulnerabilities in robot architectures. Researchers from the University of Washington have demonstrated the ability to maliciously control a wide range of functions of a Raven II teleoperated robotic surgery system, completely ignoring or overriding command inputs from the surgeon. They also found out that the emergency stop mechanism can be abused to execute attacks [16]. In a similar study, an AR.Drone 2.0 quadcopter was hacked, exposing evident security vulnerabilities [17]. Several attack scenarios were drawn, demonstrating how a malicious attacker can gain full control of the quadcopter. Other studies have identified additional commercially available platforms with critical cybersecurity issues [3,7]. Besides proposing several mitigation measures in their works, all authors conclude that most of these attacks could have easily been prevented by using well-established and readily available security mechanisms, including encryption and authentication.

All the above-presented challenges and vulnerabilities show that important security lessons and challenges for current and future robot owners are at stake, and they are a direct consequence of vendors prioritizing time-to-market and researchers prioritizing functionality and innovation over security implementation and testing. Value-sensitive design (VSD) methodologies [18], which take key human values into account, are an appropriate example of methods that vendors and researchers should consider during the design phases of robotics and networked systems.

It is noteworthy, however, that despite the generalized lack of focus on robot cybersecurity, BeamPro, a telepresence robot, is a notable exception [2]. BeamPro uses secure protocols, symmetric encryption, and data authentication. Nevertheless, despite the existence of ROS drivers for BeamPro, it is a proprietary technology, and its replicability is limited to other products marketed by the same company. In this chapter, the focus is on the *de facto* standard in robotics, ROS, with the intent to significantly improve the efficiency and security in software development for robots, not only in research, but also for robotics start-ups or robot-related businesses within major companies.

## SECURITY CONCERNS IN ROS

ROS is a very popular robotics middleware whose major goals are hardware abstraction, low-level device control, implementation of commonly used functionalities, message-passing between processes, and package management [6]. ROS promotes code reuse with different hardware by providing a large amount of libraries available for the community, like laser-based SLAM [19], 3D point cloud-based object recognition [20], as well as tools for visualization, recording experiments, and much more. ROS is based on a publish-subscribe and message passing system which utilizes the extensible markup language remote procedure call (XML-RPC) protocol. This allows native clients from multiple platforms and languages to send and receive data in a peer-to-peer manner.

Despite the clear advantages of integrating robots in ROS, it lacks any security protection feature by default, which makes robots prone to malicious attacks. This is, in fact, one of the reasons for ROS being preferred in research and having not yet fully established itself in industry. In this section, we survey the known security issues with ROS.

* TOR stands for The Onion Router, a worldwide network of servers that enable people to browse the Internet anonymously.

The communication between nodes in ROS uses *clear text* through TCP/IP and UDP/IP. ROS only checks the MD5 sum of the message structure to guarantee that the parties agree on the message layout. Unencrypted text has benefits in ease of use, debugging, and performance. However, this allows an unauthorized listener to easily intercept and interpret the form of the message, gather information and spoof fake messages into the system [21].

ROS modularity has been highly praised in the community. However, it also poses disadvantages, such as the *exposure of TCP ports*, which provide no authentication. The use of unsecured and unprotected TCP ports and the *lack of authentication mechanisms* may result in malicious talkers, which can fiddle with the system, the injection of messages (person-in-the-middle) and replacing existing talkers/listeners, thus directing external packets towards the ROS ports.

Furthermore, ROS has anonymous publish/subscribe semantics. Therefore, general nodes are not aware of who they are communicating with [2]. The system uses *weak authorization schemes* with no sender verification, no checks for data integrity and authenticity, and no definition of access levels. Certain remote clients should not have access to the entire ROS system as this allows users to send direct commands to robots, which may bypass safety thresholds. Since ROS provides no conflict resolution measurements, different nodes can concurrently inject velocity commands to a mobile base, control an actuator, a manipulator, and so on.

Regarding communication, ROS *requires bidirectional networking* between all computers. Thus, the firewall settings of the nodes involved must be less strict, posing an additional security threat, and resulting in additional networking overhead (computational requirements and latency).

ROS also does not provide *Quality of Service (QoS)* aspects. Each node is responsible to manage its own communication, similar messages are not compacted, nor there is any effort to reduce network communication. Messages of the same type are potentially candidates for compression, as the high volume of communication hampers the development of time critical applications.

There are no studies concerning ROS performance with a high number of nodes. However, being centralized at the ROS master node, the ROS naming service leads to *scalability* problems [22], not being designed to handle a large volume of requests. This introduces a special bottleneck since the system cannot produce responses in a reasonable time, being exposed to DoS attacks. Also, due to the complete lack of security features, it is unclear whether any analysis has been conducted to look for other security vulnerabilities, such as buffer overflows or remote code execution possibilities within ROS.

According to [23], the availability of ROS source code permits an adaptation to alter it to a multimaster approach, for example, as seen in Reference 24. With this modification, ROS can support a greater number of devices and improve its usability in larger installations for intelligent environments, wireless sensor networks, and/or multi-robot systems. The limitation of every ROS master node to hold the complete namespace in its memory was identified, and a system to apply rules to the namespaces and reducing the memory and bandwidth usage was introduced. Also, the author provided support for the IPv6 protocol in ROS,* and it was shown that the verbosity and complexity of the XML-RPC protocol used lays a heavy burden on distributed nodes in the network. Therefore, extending ROS to support commonly used data formats, such as JSON or Protocol Buffers paired with HTTP, could extend its reach.

When a ROS master node is launched, it opens a port for any network machine to attach to. In turn, these machines may then query the ROS master for critical tasks, such as setting up ROS TCP/UDP connections with other ROS nodes, subscribing to any topic, killing any node in the network, and so on. Consequently, a mischievous entity on the same network as a robot running ROS will most likely have access to all of the robot's data with many avenues of exploitation, such as sending commands, clogging network connections, accessing camera streams, and generally causing bad things to happen.

Recently, important research to overcome some of the security limitations of ROS has been reported. For instance, in Reference 25, the focus has been placed on authentication and authorization

* http://wiki.ros.org/ros_comm6

(AA) in ROS 1.x systems. Before transmitting and receiving data, system entities need to authenticate using a login and password pair, and an authorizer node checks the authentication data, the roles and rights associated with them. The AA node generates all the communication keys to guarantee that communication is always evaluated, whether it comes from a trusted source or not. In Reference 26, the author also analyses the vulnerabilities of ROS and the resulting threats that could be posed by attackers, and propose to standardize security logging formats, a profiling syntax for security policies, and providing new tools to introspect recorded security logs. In Reference 27, a runtime verification framework for robotic applications on top of ROS has been proposed. It provides a way for continuous observation of all communication requests and messages by inserting a monitoring tool into the connection using the man-in-the-middle technique. It detects potentially unwanted messages, enforcing access control policies, such as only enabling certain ROS nodes to publish messages on a certain topic. Additionally, core software development of the ROS 2.0 framework has been started in 2015, and an alpha release for community testing has already been made available at the time of writing this chapter. Efforts have been dedicated to integrate DDS (data distribution service), a standard specification for publish-subscribed communications in real-time and embedded systems, as a transport layer for ROS 2.0. Complying with the DDS Security specification ensures authentication, access control, and data cryptography. Moreover, ROS 2.0 is expected to provide additional security by supporting IPv6 out of the box, making host scanning and identification more challenging for attackers.

## INITIATIVES FOR SECURING ROBOT APPLICATIONS ON ROS

Robot cybersecurity and cybersafety need to be seriously addressed by the robotics community. The field of AI and robotics is expected to face similar security problems to those faced by the computer revolution with the boom of the Internet [2]. Besides the attacks already faced by computer systems nowadays, for example, DoS, eavesdropping, spoofing, tampering, privilege escalation, information disclosure, etc., robots add the physical interaction factor, which increases their susceptibility to attacks. Thus, unauthorized access to a robot may result in disastrous consequences, such as disabling it, causing it to perform incorrectly, damaging itself, damaging its surroundings, or even worse, injuring someone nearby.

Moreover, experiments have shown that it can be particularly challenging in robotics to distinguish a cyber-physical security exploit from an hardware or software bug [21]. Hence, the detection of mischievous hacks may not be straightforward, and exploits might even be disguised as simple bugs, while the system is actually under attack.

ROS is the leading middleware in this field of research, benefiting from a huge community support, regular software releases, widespread use in academia and in some industry sectors, and reaching from small embedded devices up to large-scale service robots. However, in a ROS-based system, an attacker can easily create a node, query the Master about the system state, and send shutdown commands to kill any node, or publish adulterated messages on important topics, for example, erroneous sensor data. Moreover, nodes are uniquely identified by their name, with newly created nodes replacing existing ones with the same name. Thus, an attacker can easily fake a node to publish bogus messages on important topics. For instance, a navigation node in a robot may be killed and replaced by a fake node that misdirects the robot.

Below, we overview five distinct initiatives proposed recently to secure ROS, and in the next section, we run experiments to attest and compare the performance of each approach, mainly focusing on the overhead of communication of these initiatives, when compared to non-secure ROS systems.

## SROS

*SROS*, which stands for *Securing ROS*, has been proposed as an addition to the ROS API and ecosystem to support modern cryptography and security measures in an effort to address existing vulnerabilities [28]. Although still highly experimental and under heavy development,* *SROS* supports "native Transport Layer Security (TLS) for all socket transport within ROS, the use of x.509 certificates permitting chains of trust, definable namespace globbing for ROS node restrictions and permitted roles, as well as convenient user-space tooling to auto generate node key pairs, audit ROS networks, and construct/train access control policies," according to the authors.

In *SROS*, the use of TLS between two communicating applications fosters privacy, authenticated identity and data integrity. *SROS* provides a ROS-independent keyserver at startup to generate and distribute keys and certificates to ROS nodes. The keyserver simplifies the use and development of *SROS* enabled systems for end users, being seamlessly integrated in *SROS*, generating and distributing Public Key Infrastructure (PKI) elements including: asymmetric keys, certificate authorities (CA), and signed certificates, and providing conservative default security configurations.

At the time of writing, *SROS* lacks support for the ROS C++ library (*roscpp*), enabling only nodes coded in Python (*rospy*). To run *SROS*, it was necessary to install it from source and make some initial configurations, namely sourcing the *SROS* setup, running the *SROS* training mode so that the necessary topics/services parameters could be learned during a bootstrap session, and changing some of the keyserver generated configurations.

## ROS-AES-Encryption

A recent study [29] has shown that using ciphered communications in ROS provides minimal overhead of CPU performance and communication load for systems without hard real-time constraints. The work consisted in encrypting data transmitted between ROS processes using the 3DES algorithm, by adding a pair of ROS nodes for the encryption and decryption tasks, and without changing ROS message structures nor ROS standard functions to send the data. The authors coded the nodes in Python (*rospy* library) and evaluated the performance of the system both from the computing and the communication point of view.

Inspired by this, we have developed a very similar approach for evaluation, with two main differences in our design choice for performance reasons: the encryption/decryption nodes were coded in C++ (with *roscpp*) using the Crypto++ library, and we have used the Advanced Encryption Standard (AES) [30] cryptographic algorithm, which is known to be faster than 3DES. The AES algorithm is a symmetric block cipher, which converts data in plaintext to an unintelligible form, that is, ciphertext. The AES algorithm is capable of using cryptographic keys of 128, 192, and 256 bits to encrypt and decrypt data in blocks of 128 bits.

We provide an independent study using this ciphered strategy, which we designated as *ROS-AES-Encryption*. Since *ROS-AES-Encryption* does not change the core packages of ROS, any queries to the ROS master will be satisfied as we are running a standard ROS system. Therefore, encrypting only the communications is not a completely secure solution, as will be seen in Section "Results and Discussion."

## SRI's Secure ROS

*Secure ROS* has been developed by SRI International, and it provides alternative versions of core ROS packages that enable secure communication among ROS nodes [31]. The main goal of *Secure ROS* is to enable secure communication for regular users of ROS. To this end, the authors have integrated the IP extension to security (IPSec) inside ROS. IPSec is used in transport mode, thus encrypting and authenticating the payload of the exchanged messages. In addition, the transport and application layers are always secured by a hash, so they cannot be modified in any way.

The user may specify authorized subscribers and publishers to *topics*, setters and getters to *parameters*, and providers (servers) and requesters (clients) of *services* in a configuration file for the

* http://wiki.ros.org/sros

ROS master at run time. Accordingly, *Secure ROS* will only allow authorized nodes to connect to specific topics, services and parameters listed in the configuration file specified.

Considering the implementation, *Secure ROS* shares some similarities with SROS. However, the installation process is more expeditious, as there are debian builds available. Also, *Secure ROS* supports both *rospy* and *roscpp*, it can be easily set up, being simple and transparent to the common user, which only needs to provide the required configuration file with access rules to each ROS entity. A downside of *Secure ROS* is that it does not provide formal verification means to guarantee that the desired properties conform to the specifications.

## Secure-ROS-Transport

Dieber et al. proposed a ROS security architecture to run at the application level [32]. Using a dedicated authentication server they have enabled secure communication between ROS nodes, resorting to cryptographic methods that ensure data confidentiality and integrity, and avoiding some of the most serious security issues of ROS. However, the proposed security architecture was intended for use on top of ROS. Thus, some key vulnerabilities, which could not be solved at application level were still present, such as arbitrary subscription to data albeit its encryption, and susceptibility for DoS attacks via high publishing frequency of bogus data in topics. As a consequence, the authors proceeded with a modification of the core ROS packages to enhance the security of ROS, as described in References 33 and 34. This led to a secure communication channel, denoted as *secure-ros-transport*, enabling ROS nodes to communicate with authenticity and confidentiality in a peer-to-peer basis. The authors have used TLS for TCP and Datagram TLS (DTLS) for UDP to secure the communications between nodes and the ROS master, by adding an additional handshake step and performing fine-grained authorization on a per-topic basis. This approach reduces the possibility of DoS attacks, and arbitrary subscription/publishing of messages in ROS topics. However, as pointed out by the authors, the ROS master itself is currently not secure, still transmitting information about nodes and topics to any entity that solicits it (*rosnode list*, *rostopic list*, etc.), and allowing a special XMLRPC call to externally kill any running node (*rosnode kill <node>*).

A source installation of *secure-ros-transport* was necessary, as the modifications have been integrated within ROS. For the user this means that there is no need to recompile nodes to benefit from *secure-ros-transport*. It is also noteworthy that the authors did not focus on handling certificates, and they draw attention to the need for proper key management. Also, in the version that we had access to, the Botan Crypto library used (v1.11) was outdated. Although case study data was provided in References 33, mostly focused in the overhead introduced by securing the communication channel, this is further explored in Section "Results and Discussion" of this chapter.

## Rosauth

Focusing essentially on cloud-based solutions, and dealing with the connection of non-native clients with ROS systems, such as ROS-enabled robots, a community effort named *rosbridge* has been gaining increased attention [35]. Driven by the need to consider the critical issue of security in these systems, in Reference 36 a *rosbridge* authentication mechanism has been proposed under the name *rosauth*, aiming to achieve secure authentication for remote, non-native clients in the widely used ROS middleware.

*Rosauth* utilizes web authentication tokens to verify remote clients via an arbitrary external user management system, which is integrated as part of the *rosbridge* protocol. Inspired by message authentication codes (MACs), messages are hashed with a predefined key and a known algorithm, allowing for message authentication. The server stores several keys, and if it receives a message, it compares the received hashed message with the result of the hash function. If it matches, the server will accept the message. Otherwise, it marks it as it came from an untrusted source, and drops it.

The SSL protocol is used to ensure confidentiality, integrity, and authenticity of individual packets. By using certificates issued by trusted certificate authorities, SSL can ensure that external clients know that each ROS system, including the external authenticator, is legitimate. The developed security token schema ensures that only clients which have been authenticated from some trusted external authentication source are allowed access to the ROS system. Despite solving the authentication issue, *rosauth* does not provide authorization levels. Thus, once authenticated, any remote client (as well as any native entity running ROS in the same network) can access the entire ROS system, for example, sending direct commands to robots without restrictions. Therefore, expanding the mechanism developed to encompass authorization levels is a future endeavor of the developers, whom have implemented *rosauth*.

## RESULTS AND DISCUSSION

In this section, the security initiatives described in the previous section have been tested. An Intel i5-4590 (3.30 GHz) CPU, with 8 GB of RAM, and an Ubuntu Linux 16.04 64-bit Operating System running ROS Kinetic Kame has been prepared to run experiments. Results in this section focus on the communication performance of each initiative when transmitting data between a publisher and a subscriber *node* running in the same machine. This allows us to compare the delay in communications, the number of lost messages, the ability to keep up with intended publishing rates, the levels of access from unauthorized nodes inside the ROS network, and to generally assess the trade-off between security and fluidity of each approach.

In each of the results trial, we have defined two types of messages to be published and subscribed by two separate ROS nodes: (i) a “Hello World!” string with a header containing the publishing *timestamp*, a message sequence number and an optional *frame_id* string (which we set to “0”); (ii) a *nav_msgs/Occupancy* grid map illustrated in Figure 20.1, which also includes useful header information. Table 20.1 provides an overview of the experiments conducted. For each tested initiative, the string with header message, consisting of 27 bytes, was published 600,000 times at three different intended publishing rates (1, 10 and 30 kHz). The occupancy grid message, consisting of 343 kB of data, was published 150,000 times, also at three different intended publishing rates (250 Hz, 2.5 and 7.5 kHz).

Henceforth, we designate the “Hello World!” message, simply as *string*, and the occupancy grid message, as *map*. Below, we exemplify a string and a map message published during the experiments (Figure 20.2).

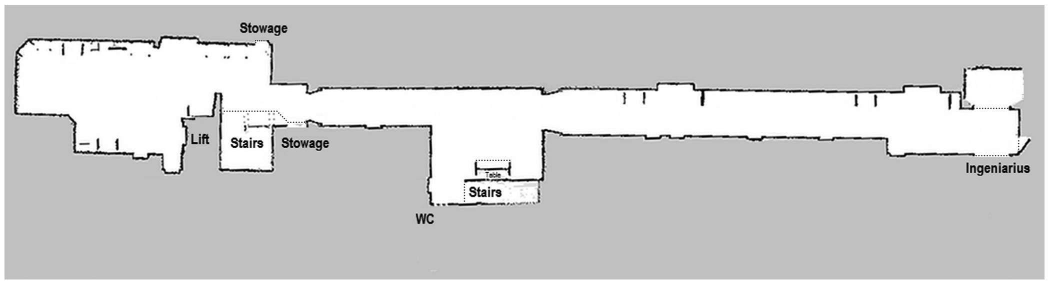

**FIGURE 20.1** Occupancy grid map transmitted during the experiments (1187 × 296, with 0.05 m/cell resolution).

**TABLE 20.1**
**Overview of the Tests Conducted for Each Initiative**

| Message Type | Total # Messages @ Publishing Frequency | | |
|---|---|---|---|
| “Hello World!” String w/Header (27 Bytes) | 600 k @ 1 kHz | 600 k @ 10 kHz | 600 k @ 30 kHz |
| Occupancy Grid Map (343 KBytes) | 150 k @ 250 Hz | 150 k @ 2.5 kHz | 150 k @ 7.5 kHz |

(a)
```
header:
  seq: 1
  stamp:
    secs: 1500660269
    nsecs: 676667209
  frame_id: 0
text: Hello World!
```

(b)
```
header:
  seq: 1
  stamp:
    secs: 1500660269
    nsecs: 676667209
  frame_id: map
info:
  map_load_time:
    secs: 1500654640
    nsecs: 559560440
  resolution: 0.0500000007451
  width: 1187
  height: 296
  origin:
    position:
      x: -29.675
      y: -7.4
      z: 0.0
    orientation:
      x: 0.0
      y: 0.0
      z: 0.0
      w: 1.0
data: [...]     # An int8 array with size: 1187x296=351352
```

**FIGURE 20.2** Fields and format of the ROS messages used in the experiments. (a) String message. (b) Map message.

The publishing rates in the experiments were defined so as to allow an analysis at three different levels: moderate, fast, and almost unbearable rates, thus taking each initiative to the limit. Since ROS is not a real-time system, the intended publishing frequency is not guaranteed, and packet losses are expected, as well. In all experiments, we have defined a queue size of 1 for each publisher and each subscriber. The official "insecure" release of ROS Kinetic Kame was also tested to allow comparison of delays imposed by distinct security initiatives.

Table 20.2 presents the overall results of the string experiments. As can be seen, most approaches have comparable performance with the official ROS release for the transmission of small string messages, with the delay values staying very close to each other, despite the generally superior performance of ROS without any security layer. However, it can be noted that in SROS, the publisher was not able to keep up the intended 30 kHz rate, reaching a maximum of ~21 kHz with a very high packet loss (59.72%).

*Rosauth* is clearly a special case, because the publishing entity is not a native ROS node, but a HTML5/Javascript web client instead. This client connects to ROS via *rosbridge* using websockets, providing a JSON message which is parsed on the ROS side, and then published in the ROS network. In the javascript publishing loop, the *setInterval* function imposes a 4 ms lower limit, which results in a maximum publishing frequency of 250 Hz, as seen in Table 20.2. For this reason, tests with 10 and 30 kHz were skipped for *rosauth*, as the results would be similar to those presented with 1 kHz. Evidently, the above mechanism for transmitting messages from non-native clients into ROS has an impact on the publishing/subscribing delay.

The box plots in Figure 20.3 illustrate the delay in message delivery for each of the tested initiatives in the string experiment at 1 kHz. The average value along 600 k transmissions is represented by a black asterisk. The ends of the blue boxes and the horizontal red line in between, correspond to the first and third quartiles and the median delay values, respectively.

**TABLE 20.2**
**Results for the String Experiments. For Each Line, 600 k Strings of 27 Bytes Were Published and Subscribed (P/S)**

| | Publishing Frequency (Hz) | Real Publishing Frequency (Hz) | Packet Loss (Absolute/%) | Average P/S Delay (ms) | Standard Dev. P/S Delay (ms) | Median P/S Delay (ms) |
|---|---|---|---|---|---|---|
| ROS (C++) | 1,000 | 999.924 | 290 (0.048%) | 0.141 | 0.044 | 0.144 |
| | 10,000 | 9990.432 | 868 (0.145%) | 0.023 | 0.016 | 0.020 |
| | 30,000 | 29951.328 | 122659 (20.443%) | 0.022 | 0.007 | 0.021 |
| SROS (Python) | 1,000 | 999.955 | 85 (0.014%) | 0.131 | 0.047 | 0.117 |
| | 10,000 | 9985.279 | 37068 (6.178%) | 0.061 | 0.024 | 0.059 |
| | 30,000 | 20898.611 | 358330 (59.721%) | 0.079 | 0.098 | 0.083 |
| ROS-AES-Encryption (C++) | 1,000 | 999.992 | 98 (0.016%) | 0.150 | 0.038 | 0.157 |
| | 10,000 | 9992.973 | 882 (0.147%) | 0.025 | 0.012 | 0.023 |
| | 30,000 | 29962.872 | 125337 (20.889%) | 0.023 | 0.006 | 0.022 |
| Secure ROS (C++) | 1,000 | 999.998 | 54 (0.009%) | 0.156 | 0.047 | 0.145 |
| | 10,000 | 9996.984 | 518 (0.086%) | 0.023 | 0.010 | 0.019 |
| | 30,000 | 29929.231 | 156039 (26.006%) | 0.022 | 0.012 | 0.021 |
| Secure-ROS-Transport (Python) | 1,000 | 999.982 | 99 (0.016%) | 0.167 | 0.028 | 0.179 |
| | 10,000 | 9987.499 | 752 (0.125%) | 0.105 | 1.236 | 0.054 |
| | 30,000 | 29683.691 | 7890 (1.315%) | 0.071 | 0.420 | 0.066 |
| Rosauth (HTML5/Javascript) | 1,000 | 249.929 | 76 (0.013%) | 1.024 | 2.589 | 1.006 |

An analysis of the box plots confirms the longer delays occurring in the *rosauth* tests, and similar delays that all other initiatives present in the transmission of smaller messages. Due to the large size of each dataset, some outliers (red crosses) are visible, specially above the upper extremes of each box plot, which shows that occasionally delays are higher than expected, possibly due to computation peaks or network latency.

On the other hand, Table 20.3 presents the overall results of the map experiments. Since we are now transmitting a large message of 343 kB, the results are significantly different when compared to the string experiment. The tests with ROS and *Secure ROS* stand out from the remaining initiatives, being able to keep up with the intended publishing rates, and presenting low percentages of packet loss (<0.2%). This confirms that *Secure ROS* is one of the most promising initiatives proposed to secure ROS without compromising transmission performance.

The *ROS-AES-Encryption* algorithm was only able to publish the map message at ∼186 Hz. Clearly, the encryption of large blocks of data imposes delays in the publishing rate. However, it is interesting that no packet was lost during the experiments with *ROS-AES-Encryption*, which implies that the encryption + publish step always took longer than the subscribe + decryption step. The average delay in delivery of messages for *ROS-AES-Encryption* is within the [5.5, 5.8] ms interval, which is 15–45 times higher than in a regular ROS transmission.

*SROS* and *Secure-ROS-Transport* presented very similar results. Both approaches reached their publishing limit at around 80–82 Hz, dropping several packets (∼48%–53%). Their average delay in the delivery of messages is within the [16.0, 16.3] ms interval, which is 44–129 times higher than in a regular ROS transmission. The fact that the tests of both initiatives were conducted with ROS nodes written with the *rospy* Python library (as opposed to C++ in ROS, Secure ROS and ROS-AES-Encryption) may also have an impact in these results.

Similarly as before, *rosauth* presents the lowest publishing frequency of 1.6 Hz, and an extremely high average delay of 125 seconds (340–990 $\times$ $10^3$ times higher than in a regular ROS transmission), which is the result of an approximately constant growing delay of message delivery along the

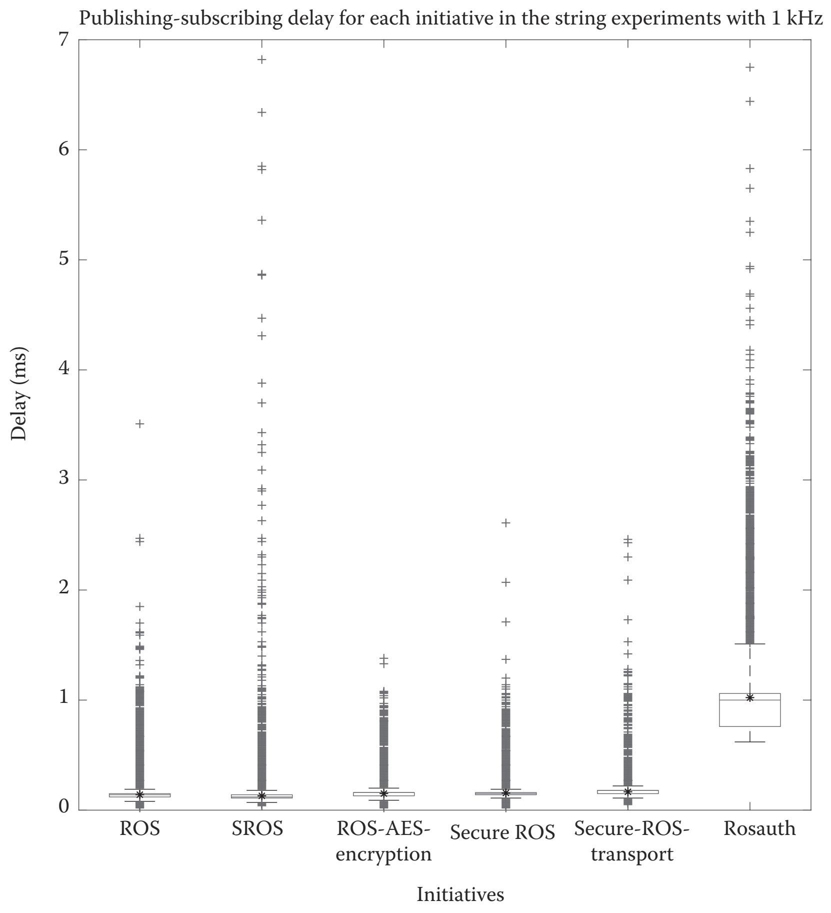

**FIGURE 20.3** Overview of the P/S delays for the String experiments at 1 kHz.

experiment, which starts with delays of ~0.7 s and ends at ~266.2 s. For the same reason as before, tests with 2.5 and 7.5 kHz were skipped for *rosauth*.

The box plots in Figure 20.4 illustrate the delay in message delivery for each of the tested initiatives in the map experiment at 250 Hz. Like before, the average value along 150 k transmissions is represented by a black asterisk. The delays of *rosauth* were plotted separately due to the distinct order of magnitude.

Besides *rosauth*, the average delay is a bit superior to the median for all other initiatives. This means that the delay values are positively skewed, that is, most values are below the average, and as a consequence, outliers are above the upper extremes of each box plot. The box plots confirm the superior performance in message delays of *Secure ROS*, followed by *ROS-AES-Encryption*, and then *SROS* together with *Secure-ROS-Transport*.

Packet transmission delays, losses and upper publishing frequency limits are not the only relevant issues in the analysis of the security initiatives for ROS. We proceed with a qualitative analysis of the security aspects of each initiative, mainly checking the extent at which the access to data is prevented from unauthorized entities with access to the ROS network, where the messages are exchanged.

ROS provides several command-line utilities, which allow to anonymously retrieve the list of topics in use (*rostopic list*), the list of nodes in use (rosnode list), the list of services in use (*rosservice list*), the ability to kill a running node (*rosnode kill <node>*), the ability to display the messages being

**TABLE 20.3**
**Results for the Map Experiments. For Each Line, 150 k Maps of 343 KB Were Published and Subscribed**

| | Publishing Frequency (Hz) | Real Publishing Frequency (Hz) | Packet Loss (Absolute/%) | Average P/S Delay (ms) | Standard Dev. P/S Delay (ms) | Median P/S Delay (ms) |
|---|---|---|---|---|---|---|
| ROS (C++) | 250 | 250.0 | 0 (0.0%) | 0.366 | 0.155 | 0.283 |
| | 2500 | 2499.776 | 71 (0.047%) | 0.171 | 0.065 | 0.151 |
| | 7500 | 7496.412 | 235 (0.157%) | 0.126 | 0.049 | 0.122 |
| SROS (Python) | 250 | 81.801 | 75818 (50.545%) | 16.248 | 3.594 | 15.286 |
| | 2500 | 81.316 | 74376 (49.584%) | 16.293 | 3.593 | 15.342 |
| | 7500 | 81.859 | 73354 (48.903%) | 16.173 | 3.499 | 15.263 |
| ROS-AES-Encryption (C++) | 250 | 186.923 | 0 (0.0%) | 5.516 | 0.182 | 5.451 |
| | 2500 | 186.112 | 0 (0.0%) | 5.544 | 0.194 | 5.468 |
| | 7500 | 178.870 | 0 (0.0%) | 5.768 | 0.288 | 5.661 |
| Secure ROS (C++) | 250 | 250.0 | 1 (0.0006%) | 0.351 | 0.143 | 0.277 |
| | 2500 | 2499.634 | 72 (0.048%) | 0.172 | 0.065 | 0.154 |
| | 7500 | 7496.050 | 184 (0.123%) | 0.126 | 0.041 | 0.122 |
| Secure-ROS-Transport (Python) | 250 | 80.308 | 74855 (49.903%) | 16.096 | 3.678 | 15.096 |
| | 2500 | 79.938 | 78000 (52.0%) | 16.286 | 3.899 | 15.090 |
| | 7500 | 80.275 | 79547 (53.031%) | 16.284 | 3.817 | 15.162 |
| Rosauth (HTML5/Javascript) | 250 | 1.613 | 1 (0.0006%) | 125048.519 | 88037.166 | 133328.971 |

exchanged in a topic (*rostopic echo <topic>*), and much more. Thus, we have run the aforementioned commands from within the ROS network and inspected the access levels granted by each different initiative.

Table 20.4 provides the data returned by each initiative on requests made inside the ROS network by unauthorized nodes. *SROS* proof to be the approach with higher security levels, not replying to any kind of query to the ROS master from unauthorized nodes. *Secure ROS* also presented adequate levels of security, not allowing access to list and view any messages in any ROS topic by unauthorized entities. Furthermore, it also does not allow to kill nodes. Surprisingly, for unknown reasons it is possible to list the nodes and services that are in use by ROS. It should also be noted that *SROS* provides access configuration at node level, while *Secure ROS* provides access configuration at machine level (IP address filtering). Hence, any query ran, for example, by SSH, from within the machine running the secure transmission will return data for *Secure ROS*, but not for *SROS*.

Unlike *SROS* and *Secure ROS*, other initiatives do not secure the ROS master adequately, thus providing inadequate authorization levels. They all allow listing topics, nodes and services from within ROS, as well as killing nodes without authorization. However, when running a *rostopic echo* command, *Secure-ROS-Transport* does not provide access to the messages being exchanged, and the *ROS-AES-Encryption* will display unintelligible ciphered text.

As for *rosauth*, it assumes that the ROS network is trusted and only secures the connection between the non-native client and the ROS interface, which then just forwards the messages insecurely through ROS. So, for the sake of the present analysis on the ROS side, it operates just like any ROS network without security.

In general, considering all the initiatives tested, *Secure ROS* and *SROS* are currently the ones with the most potential for enhancing the security of ROS. *Secure ROS* provides impressive transmission performance with negligible overhead and satisfactorily prevents access to data from unauthorized entities. *SROS* is undoubtedly the most secure initiative tested, providing satisfactory performance on high-throughput transmissions. Nevertheless, considering that they are still under development,

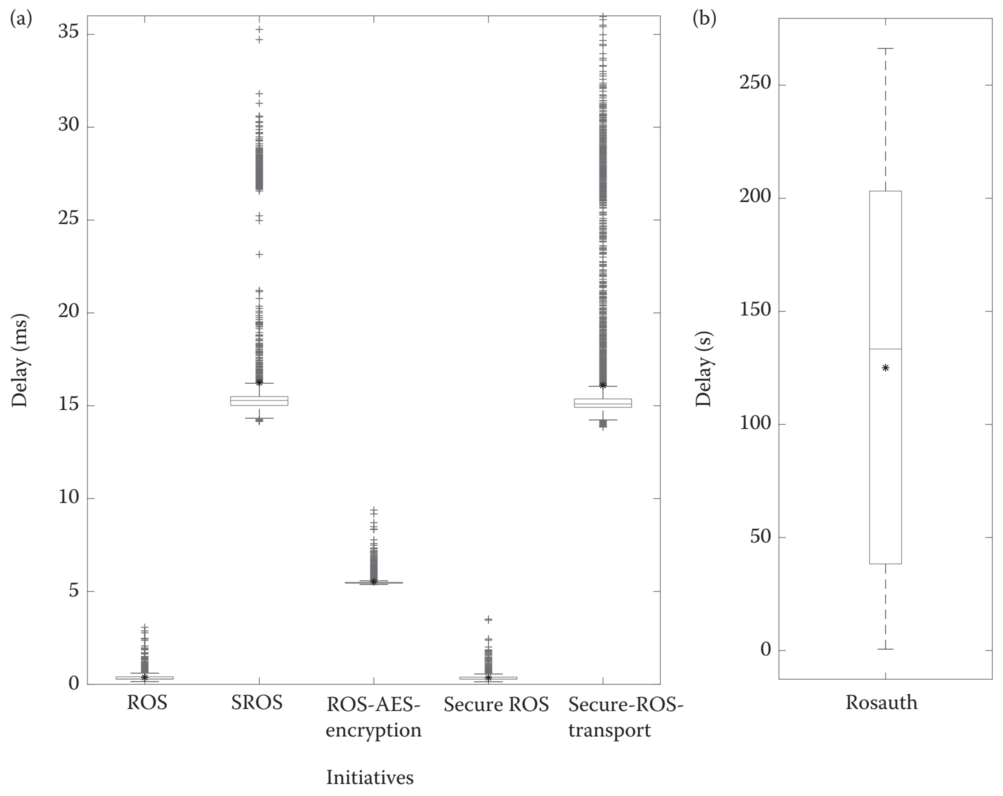

**FIGURE 20.4** Overview of the P/S delays for the map experiments at 250 Hz. (a) Publishing-subscribing delay for each initiative in the map experiments with 250 Hz. (b) Publishing subscribing delay for Rosauth in the map experiment with 250 Hz.

both have room for improvement, *Secure ROS* could prevent unauthorized access to list the nodes and services in use by ROS and allow authorization at node level, while *SROS* could support the ROS C++ *roscpp* library, and provide easier installation and deployment features.

Robot cybersecurity should be addressed at several different levels. In this work, we have focused on the security of the robot operating system (ROS). Besides ensuring the secure communication of ROS components, it is also important to secure other components of the overall robotic system. For instance, the network in which the robot(s) operate should be impenetrable, using WPA2+AES security, SSID hiding, MAC ID filtering, static IP addressing, and any other widely documented

**TABLE 20.4**
**Data Returned by Each Initiative on Requests Made Inside the ROS Network**

| | ROS | SROS | ROS-AES-Encryption | Secure-ROS | Secure-ROS-Transport | Rosauth |
|---|---|---|---|---|---|---|
| rostopic list | ✓ | ✗ | ✓ | ✗ | ✓ | ✓ |
| rosnode list | ✓ | ✗ | ✓ | ✓ | ✓ | ✓ |
| rosservice list | ✓ | ✗ | ✓ | ✓ | ✓ | ✓ |
| rosnode kill | ✓ | ✗ | ✓ | ✗ | ✓ | ✓ |
| rostopic echo | ✓ | ✗ | ✓✗ | ✗ | ✗ | ✓ |

security measures. All accesses from non-native ROS clients should use SSL/HTTPS secure connections and authentication to verify the client's identity. The ROS network can be deployed under a VPN to maintain security and privacy over network communications. Firewall rules should be defined allowing traffic only on specific non-default ports from specific IPs, root login through SSH should be disabled and strong user-level authentication passwords should be enforced, as well as encryption of data storage. Moreover, methods for careful protection and exchange of cryptographic keys, and for maintenance of certificates, mandatory digital signatures and access levels should be used, allowing for their secure storage [37].

## CONCLUSION AND FUTURE WORK

Although it is not a simple task, to generally improve robot cybersecurity several recommendations are important from day one [7], such as: secure software development life cycles, encrypting robot communications, keeping software up to date, granting access only to authorized users, providing methods to restore a robot to a secure factory default state, implementing cybersecurity best practices, educating roboticists and executives for cybersecurity, providing ways for users to give feedback on possible vulnerabilities, and promoting security audits before production. To this end, it is essential to enforce early and preventive secure design principles for robotic applications.

The purpose of this chapter is to identify potential security and privacy risks in the widely used ROS, thus raising awareness toward robot cybersecurity and the need to mature the industry security principles to avoid consequent penetration of insecure robots into the market.

Beyond the in-depth analysis of the literature in robotics data security, several security flaws of the widely adopted ROS framework have been revealed, and an analysis of initiatives to secure robot applications in ROS has been presented. We have also provided general recommendations and security measures at different levels to guide the implementation and deployment of single and multi-robot systems. In the future, we intend to develop a ROS-based commercially exploitable multi-robot system for surveillance of infrastructures encompassing key cybersecurity and privacy measures, as part of the ongoing STOP R&D Project.

## ACKNOWLEDGMENTS

This work was supported by the **S**eguranças robó**T**icos co**OP**erativos (STOP) research project (ref. CENTRO-01-0247-FEDER-017562), co-funded by the "Agência Nacional de Inovação" within the Portugal2020 programme.

The authors thank Bernhard Dieber from Joanneum Research, Institute for Robotics and Mechatronics, for providing the source code of *secure-ros-transport* and to Ruffin White from the University of California, San Diego, for his assistance on setting up *SROS*.

## REFERENCES

1. E. Garcia, M. A. Jimenez, P. G. Santos, M. Armada, "The Evolution of Robotics Research". *IEEE Robotics & Automation Magazine*, 14 (1), pp. 90–103, March 2007.
2. S. Morante, J. G. Victores, C. Balaguer, "Cryptobotics: Why Robots Need Cyber Safety". *Frontiers in Robotics and AI*, 2 (23), pp. 1–4, September 2015.
3. T. Denning, C. Matuszek, K. Koscher, J. R. Smith, T. Kohno, "A Spotlight on Security and Privacy Risks with Future Household Robots: Attacks and Lessons". In *11th International Conference on Ubiquitous Computing (UbiComp 2009)*, Orlando, Florida, USA, September 30–October 3, 2009.
4. N. Nevejans, "European Civil Law Rules in Robotics". Study requested by the European Parliament's Committee on Legal Affairs, *Policy Department C: Citizens' Rights and Constitutional Affairs*, pp. 1–34, October 2016.
5. D. Portugal, M. S. Couceiro, R. P. Rocha, "Applying Bayesian Learning to Multi-Robot Patrol". In *Proceedings of the 2013 International Symposium on Safety, Security and Rescue Robotics (SSRR 2013)*, Linköping, Sweden, October 21–26, 2013.

6. M. Quigley, B. Gerkey, K. Conley, J. Faust, T. Foote, J. Leibs, E. Berger, R. Wheeler, A. Ng, "ROS: An Open-Source Robot Operating System. In *Proceedings of the IEEE International Conference on Robotics and Automation (ICRA 2009), Workshop On Open Source Software*, Kobe, Japan, May 12–17, 2009.
7. C. Cerrudo, L. Apa, "Hacking Robots Before Skynet". In *Cybersecurity Insight*, IOActive Report, Seattle, Washington, USA, 2017. https://media.scmagazine.com/documents/287/hacking-robots-before-skynet_71714.pdf
8. I. Asimov, *I, Robot*. Gnome Press, December 1950. https://en.wikipedia.org/wiki/I,_Robot
9. M. Finnicum, S. T. King, "Building Secure Robot Applications". In *Proc. of the USENIX Workshop on Hot Topics in Security, 20th USENIX Security Symposium*, San Francisco, CA, USA, August 2011.
10. W. Adi, "Mechatronic Security and Robot Authentication". In *IEEE Symposium on Bioinspired Learning and Intelligent Systems for Security (BLISS)*, Edinburgh, Scotland, pp. 77–82, August 20–21, 2009.
11. G. S. Lee, B. Thuraisingham, "Cyberphysical Systems Security Applied to Telesurgical Robotics". *Computer Standards & Interfaces*, 34, pp. 225–229, 2012.
12. S. Yong, D. Lindskog, R. Ruhl, P. Zavarsky, "Risk Mitigation Strategies for Mobile Wi-Fi Robot Toys from Online Pedophiles". In *Proc. of IEEE 3rd Int. Conf. on Privacy, Security, Risk and Trust and IEEE 3rd Int. Conf. on Social Computing*, Boston, MA, USA, October 2011.
13. A. Caiti, V. Calabrò, G. Dini, A. Lo Duca, A. Munafò, "Secure Cooperation of Autonomous Mobile Sensors Using an Underwater Acoustic Network". *Sensors*, 12, pp. 1967–1989, 2012.
14. F. Higgins, A. Tomlinson, K. M. Martin, "Survey on Security Challenges for Swarm Robotics". In *Proc. of the 5th International Conf. on Autonomic and Autonomous Systems*, Valencia, Spain, April 2009.
15. N. P. Hoand, D. Pishva, "A TOR-Based Anonymous Communication Approach to Secure Smart Home Appliances". *ICACT Transactions on Advanced Communications Technology (TACT)*, 3 (5), pp. 517–525, September 2014.
16. T. Bonaci, J. Herron, T. Yusuf, J. Yan, T. Kohno, H. J. Chizeck, "To Make a Robot Secure: An Experimental Analysis of Cyber Security Threats Against Teleoperated Surgical Robotics", *arXiv: 1504:04339*, pp. 1–11, April 2015.
17. J. S. Pleban, R. Band, R. Creutzburg, "Hacking and Securing the AR. Drone 2.0 Quadcopter—Investigations for Improving the Security of a Toy". In *Proc. of IS&T/SPIE Electronic Imaging*. The International Society for Optical Engineering, San Francisco, California, January 2014.
18. B. Friedman, "Value-Sensitive Design". *Interactions*, 3(6), pp. 16–23, ACM, 1996.
19. J. Machado Santos, D. Portugal, R. P. Rocha, "An Evaluation of 2D SLAM Techniques Available in Robot Operating System". In *Proc. of the 2013 International Symposium on Safety, Security and Rescue Robotics (SSRR 2013)*, Linköpping, Sweden, October 21–26, 2013.
20. R. Rusu, S. Cousins, "3D is Here: Point Cloud Library (PCL)". In *Proceeding of the IEEE International Conference on Robotics and Automation (ICRA 2011)*, Shanghai, China, May 2011.
21. J. R. McClean, C. Stull, C. Farrar, D. Mascareñas, "A Preliminary Cyber-Physical Security Assessment of the Robot Operating System (ROS)". In *Proc. of SPIE Defense, Security, and Sensing. The International Society for Optical Engineering*, Baltimore, Maryland, Vol 8741, May 2013.
22. V. Hax, N. Filho, S. Botelho, O. Mendizabal, "ROS as Middleware to Internet of Things". *Journal of Applied Computing Research*, 2(2), pp. 91–97, July–December 2012.
23. T. Schneider, "Distributed Networks Using ROS—Cross-Network Middleware Communication using IPv6". *Diploma Thesis*, Department of Electrical Engineering and Information Technology, Technical University of Munich, Munich, Germany, October 2012.
24. A. Tiderko, F. Hoeller, T. Röling, "The ROS Multimaster Extension for Simplified Deployment of Multi-Robot Systems". *Robot Operating System (ROS), The Complete Reference (Volume 1), Studies in Computational Intelligence*, 625, pp. 629–650, Sprinter 2016.
25. R. Dóczi, F. Kis, B. Sütő, V. Póser, G. Kronreif, E. Jósvai, M. Kozlovszky, "Increasing ROS 1. x Communication Security for Medical Surgery Robot". *In Proc. of the 2016 IEEE International Conference on Systems, Man, and Cybernetics (SMC 2016)*, pp. 4444–4449, Budapest, Hungary, October 2016.
26. G. Caiazza, "Security Enhancements of Robot Operating System". *Master Thesis*, Department of Environmental Sciences, Informatics and Statistics, Universitá Ca'Foscari Venezia, Venezia, Italy, 2016.
27. J. Huang, C. Erdogan, Y. Zhang, B. Moore, Q. Luo, A. Sundaresan, G. Rosu, "ROSRV: Runtime Verification for Robots". In *Runtime Verification (RV 2014)*. chapter Notes in Computer Science, vol 8734. Springer. https://link.springer.com/chapter/10.1007/978-3-319-11164-3_20
28. R. White, H. I. Christensen, M. Quigley, "SROS: Securing ROS Over the Wire, in the Graph, and through the Kernel". In *Humanoids Workshop: Towards Humanoid Robots OS (HUMANOIDS 2016)*, Cancun, Mexico, November 15, 2016.

29. F. Lera, J. Balsa, F. Casado, C. Fernández, F. Rico, V. Matellán, “Cybersecurity in Autonomous Systems: Evaluating the Performance of Hardening ROS”. In *Proc. of the 17th Workshop of Physical Agents (WAF 2016)*, Malaga, Spain, June 16–17, 2016.
30. M. J. Dworkin, E. B. Barker, J. R. Nechvatal, J. Foti, L. E. Bassham, E. Roback, J. F. Dray Jr. “Advanced Encryption Standard (AES)”. *Federal Inf. Process. Stds. (NIST FIPS)*, pp. 1–51, Report 197, November 2001.
31. A. Sundaresan, L. Gerard, M. Kim, “Secure ROS 0.9.2 documentation”. Available at: https://sri-csl.github.io/secure˙ros. July 2017.
32. B. Dieber, S. Kacianka, S. Rass, P. Schartner, “Application-Level Security for ROS-Based Applications”. *In Proc. of the 2016 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS 2016)*, Daejeon, South Korea, October 9–14, 2016.
33. B. Breiling, B. Dieber, P. Schartner, “Secure Communication for the Robot Operating System”. *In Proc. of the 2017 IEEE International Systems Conference (SysCon 2017)*, Montreal, QC, Canada, April 24–27, 2017.
34. B. Dieber, B. Breiling, S. Taurer, S. Kacianka, S. Rass, P. Schartner, “Security for the Robot Operating System”. *Robotics and Autonomous Systems*, 98, pp. 192–203, Elsevier, the Netherlands, December 2017.
35. C. Crick, G. Jay, S. Osentosiki, B. Pitzer, O. C. Jenkins, “Rosbridge: ROS for Non-ROS Users”. In *Proc. of the 15th International Symposium on Robotics Research (ISRR)*, Flagstaff, AZ, USA, August 28–September 1, 2011.
36. R. Toris, C. Shue, S. Chernova, “Message Authentication Codes for Secure Remote Non-Native Client Connections to ROS Enabled Robots”. In *Proc. of the 2014 IEEE International Conference on Technologies for Practical Robot Applications (TePRA)*, Woburn, MA, USA, April 14–15, 2014.
37. D. Portugal, S. Pereira, M. S. Couceiro, “The Role of Security in Human-Robot Shared Environments: *A Case Study in ROS-based Surveillance Robots*”. In *Proc. of the 26th IEEE International Symposium on Robot and Human Interactive Communication (RO-MAN 2017)*, Lisbon, Portugal, August 28–September 1, 2017.