Security for the Robot Operating System

Bernhard Dieber[a,*], Benjamin Breiling[a], Sebastian Taurer[a], Severin Kacianka[b], Stefan Rass[c], Peter Schartner[c]

[a] *JOANNEUM RESEARCH*
Institute for Robotics and Mechatronics
Lakeside B08
9020 Klagenfurt, Austria
[b] *Technical University of Munich*
Software Engineering Group
Munich, Germany
[c] *Alpen-Adria Universität Klagenfurt*
Institute of Applied Informatics
Klagenfurt, Austria

Abstract

Future robotic systems will be situated in highly networked environments where they communicate with industrial control systems, cloud services or other systems at remote locations. In this trend of strong digitization of industrial systems (also sometimes referred to as Industry 4.0), cyber attacks are an increasing threat to the integrity of the robotic systems at the core of this new development. It is expected, that the Robot Operating System (ROS) will play an important role in robotics outside of pure research-oriented scenarios. ROS however has significant security issues which need to be addressed before such products should reach mass markets. In this paper we present the most common vulnerabilities of ROS, attack vectors to exploit those and several approaches to secure ROS and similar systems. We show how to secure ROS on an application level and describe a solution which is integrated directly into the ROS core. Our proposed solution has been implemented and tested with recent versions of ROS, and adds security to all communication channels without being invasive to the system kernel itself.

Keywords: Robotics, ROS, Industry 4.0, Security
2010 MSC: 68M14, 93C85, 68T40, 68N99, 94A60, 94A62

*Corresponding author
Email addresses: bernhard.dieber@joanneum.at (Bernhard Dieber), benjamin.breiling@joanneum.at (Benjamin Breiling), sebastian.taurer@joanneum.at (Sebastian Taurer), kacianka@in.tum.de (Severin Kacianka), stefan.rass@aau.at (Stefan Rass), peter.schartner@aau.at (Peter Schartner)

1. Introduction

The shift towards industry 4.0 implies stronger automatization and hence increased relevance and use of robots. An Industrial Control System (ICS) is inherently distributed and connects a potentially huge number of sensors and actors, whose orchestration exhibits complex dynamics that are typically fragile and hence vulnerable to attacks. This new technology and the tight integration and interconnection of components leads to new vulnerabilities and thus a heightened focus on the security concerns. The particular evolution of the Robot Operating System (ROS) happened under time-pressure of industry[1], which has assigned a secondary (if not ternary) role to security.

In ICS, the traditional information security triad of Confidentiality, Integrity and Availability (Confidentiality, Integrity, Availability (CIA)) will often be assigned different priorities [1, 2]. When thinking of user data in information systems, it is usually more important to keep them confidential than available. Thus, taking a service offline in response to an attack is often a reasonable reaction[2]. In information systems, lost data can often be restored from backups. However, actions taken by an ICS (e.g., an actuator damaging its surroundings) cannot simply be undone by "restoring from the last known good backup". To give two examples, if a drone or a hydro dam is under attack, simply shutting them down is not a valid response. Above all other considerations, such systems' basic functionality must be available until they reach a safe state (e.g., the drone has landed or the water reservoir held back by the dam was drained).

The development of ICS has been paralleled by the attack strategies adapting themselves to the increasing complexity of the victim systems. The resulting highly complex and extremely well coordinated attacks known as Advanced Persistent Threats (APTs) [3, 4] dramatically demonstrate the need for security to become an intrinsic part of the design of a distributed system, rather than a subsequent add-on that is considered once the availability goal has been reached.

In fact, even though safety beats security in terms of priority, there is no reliable safety without security. Just imagine the safety precautions of a robot to depend on reliable sensor data. Injecting malicious packets to mimic some dangerous situation that forces a robot to react can already cause harm by itself. Blocking messages can be equally dangerous, if the dropout message was a notification of a human being in the robot's way. These two examples (among more to follow in Section 3) already exhibit security as a necessity for safety, and indirectly, also for availability, since accidents typically induce temporal shutdowns of the production system (and hence economic damages).

This article discusses a series of proposed improvements to ROS, which have been motivated by the reported vulnerabilities and insufficiency of ROS regarding security [5, 6, 7, 8]. Specifically, we will develop our discussion along the following skeleton: we let a brief survey of related work motivate our work by

[1] see also the efforts of ROS-Industrial, `www.rosindustrial.org`

[2] although there are exceptions: taking a stock exchange offline can have wide ranging repercussions.

showing the recognition and interest of the community (industrial and scientific) to the problem of securing ROS. In section 3, we provide an analysis of ROS vulnerabilities along with a detailed description on how an attacker would perform a manipulation of a ROS application. We then review prior work concerning a security architecture to harden ROS on the application level (OSI layer 7), and to harden the ROS core (in OSI layer 4), described in sections 4 and 5.

To validate the efficacy of the security added to ROS, we use section 6 to compare the behavior of a "plain" ROS and a hardened version thereof against a fixed attack pattern. That is, we describe a practical testbed including a dedicated tool for penetration testing ROS where messages are injected to see how easy the system can be manipulated, unless cryptographic precautions are implemented. In fact, their implementation is neither heavy weight nor difficult, and our findings are that even a cryptographic light-weight armory can do quite well already.

Cryptography, however, is insufficient by itself and depends on proper and especially simple key-management. Most legacy systems, up to ones under construction today, are designed to be repaired quickly and easily. For example, if a module needs to be replaced, the service staff should not be required to do more than unplug the malfunctioning module and replace it with the new one. If cryptography comes into play, keys are stored all over the system, and replacing a module with a new one requires refreshing the keys inside the whole system. A proper key management is far from trivial and can make cryptography cumbersome (if not infeasible) to apply.

Our proposed security enhancements for *usable key-management* (see section 7.1) therefore heavily rely on smart cards and tamper proof (sub-)modules to handle the key management transparently for the engineers. This is to the end of designing modules so that they, upon replacement, are capable of automatically registering themselves with the system, while the required level of authenticity, genuineness and secure logging are all assured.

The latter is a particularly important matter of forensic investigation in case of system failures (in the worst case, involving harm to humans). This leads to *accountability* as an independent requirement beyond CIA (or better "AIC" in industrial systems), and must be considered separately. We discuss the matter as part of the outlook and follow up work to this article (in section 7.2).

Our Contribution. Our work proposes a security architecture on the application layer, along with a practical implementation and validation thereof. Since plain ROS offers only limited native security, and changes to the operating system (i.e., below the application layer) are usually nontrivial and expensive, the question of how much security can be added "on top" arises. The main message and novelty of this work is the finding that (first) ROS *can* be hardened on the application layer to a wide extent, and that only a relatively thin layer of (cryptographic) security implemented below layer 7 already thwarts many known attacks (at the appeal of using existing "off-the-shelf" mechanisms and technology). The contribution of this work is complemented by the *verification* of the additional security, by demonstrating the newly gained resilience of our

hardened ROS against attacks reported in the related literature, and a demarcation of the line between what can be done on the application layer and which security aspects must be rooted deeper or even outside the operating system. This discussion constitutes the outlook in section 7. We believe this matter to be an important one besides purely technical aspects, and hope to stipulate related research along these lines, motivated by the new findings in this paper.

2. Related work

The typical divide and combination between proactive and reactive security measures is clearly biased strongly towards preventive actions to preserve safety in our context. While there has been considerable progress on intrusion detection in ICS [9, 10, 11], these measures inevitably come with false-negative rates that may be unacceptable for human safety. The strong assurances provided by cryptographic techniques must, however, be considered carefully while avoiding ad hoc solutions with yet (un)known weaknesses wherever possible [12]. The need for security in this context has long been recognized [13, 14, 15] but respective (re-)developments of the related systems did not happen until recently [16, 17, 18]: the most notable of such developments is ROS2 (http://ros2.org), which is supposed to build upon the Data Distribution System (DDS) specified by the Object Management Group (OMG) [19, 20]. ROS2 will support multiple DDS implementations for a user to choose from. The DDS-specified interface is wrapped in a thin, feature-lean layer by ROS2 which does not support all configuration mechanisms of DDS. Thus, features like QoS or security might be left out. In addition, the ROS2 default DDS implementation recently changed to Fast RTPS[3] which only implements the RTPS transport layer [21] of DDS. Security features of DDS, however, are specified on higher layers which are then not used by ROS2 at all. To bridge this gap and towards using best-practices, we propose to stick with well-established cryptographic primitives to secure the communication within ROS to the extent possible without inducing too much overhead for key management.

Lera et al. [22] introduce an application layer security scheme for the "topics" used with ROS' publish/subscribe model, where an encrypting node is added to the network. In this approach the encrypting node subscribes to the plaintext topic, encrypts the received messages and publishes them as a seperate topic. The encryption key is stored at the ROS-Master and is distributed when a legitimate client as well as the encrypting node register at the master. The subscribers read only from the encrypted topic and decrypt the messages with the previously received key. It is also mentioned that the keys should be securely transferred using asymmetric cryptography. Still, this method does not prevent nodes from subscribing to plaintext topics (thus partly missing the confidentiality goal), and has no mandatory integrity protection on encrypted messages. Thus, subscribers may unknowingly fall victim to manipulated data. Finally, in

[3] `http://www.eprosima.com/index.php/products-all/eprosima-fast-rtps`

this approach the–in industrial context–most important security goal availability is not discussed at all. Apart from that, there is also no mechanism which ensures that a node is trustworthy (genuine), i.e., using asymmetric cryptography alone is not sufficient for secure key distribution. These drawbacks indicate that simply encrypting messages is not enough to assure secure communication between ROS nodes regarding confidentiality, integrity and authenticity.

Another application-level approach has recently been presented in [23] where the authors use special ROS services for authentication before the usage of certain resources (e.g., a database). This provides an additional layer of isolation around critical application parts. The authors, however, state that communication in ROS is secured underneath using SSH which in fact is not the case. Thus, their authentication scheme using usernames and passwords can very easily be broken by an attacker.

In Section 4, we describe an application layer approach based on [24] which ensures confidentiality as well as integrity and authenticity of ROS messages. Securing ROS communication exclusively on the application layer also means that ROS itself is not modified, which still leaves several weaknesses in its architecture unconsidered. Therefore, a transport layer approach, which requires modification of ROS, has to be taken into account as well.

The ROSRV project [25] is a very interesting approach to safety (primarily; security comes as a second goal) of ROS-based applications. It uses runtime verification of pre-defined models of the applications to detect and prevent actions which do not conform with the expected application behavior. The ROSRV master acts as a transparent proxy to the ROS master and checks each XML Remote Procedure Call (XMLRPC) call towards the master for its legitimacy before letting it pass through. One drawback, however, is that ROSRV performs the authentication of nodes based on IP addresses. Thus, an attacker who has access to a host running a ROS node can circumvent that. In addition, as shown in later sections, the assumptions that an attacker has to present itself as legitimate ROS node does not hold since the XMLRPC API gives enough alternative attack surfaces.

The Open Source Robotics Foundation (OSRF) has started to implement security for ROS in the SROS project[4] [26]. In SROS, the TCP communication channels are secured by using TLS to establish node-to-node connections within the ROS network. Further, policies are used to provide an access control mechanism for topics, services, parameters and XMLRPC calls. The generation and distribution of certificates and keys is handled by a keyserver node in a preceding initialization phase. A drawback of this approach is the need of a secure environment during the initialization phase, which cannot be presumed in general. Aside from that, SROS is implemented only for Python and not at all for C++-based nodes and communication via UDP is also not considered. Due to these issues, we describe a concept for secure communication channels between C++-based nodes using TCP as well as UDP communication via TLS

[4] `http://wiki.ros.org/sros`

and DTLS in section 5 based on [27].
Recently, an approach to secure the ROS core on the internet layer has been released[5]. IPSec is used to secure the communication channels. While this is more transparent for users, it is less flexible than the certificate-based approach. In addition, access control to topics is done using plaintext files instead of certificates which makes a manipulation not detectable.
A similar approach to ours is also followed in the Open Platform Communication - Unified Architecture (OPC-UA) [28], and most strongly relates to our work, with the main difference of our approach being intentionally less complex to be easier to implement since OPC-UA additionally supports a client-server paradigm and SOAP communication (while ROS communicates data only in binary form) with dedicated security models and also includes lots of flexibility and functionality that may not be fully needed in a robot system. A design (like ours) that strives for the minimal necessary functionality may thus reduce the risk of security exploits using unneeded functions.
A whole line of apparently independent yet related research also looks at the publish/subscribe model [29] not for ROS, but rather in other contexts like the internet [30]. While the requirements there are inherently different and expectedly more complex than for industrial manufacturing processes, the lessons learned on attack patterns and behaviors are relevant to our study here too. Especially so, since the need for penetration testing tools tailored to the special settings of ICS is demanding, yet there appears to be no intensive development of related software. An independent contribution of this work is a step towards satisfying this need.

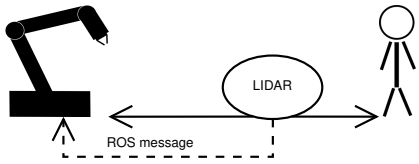
3. Security issues in the Robot Operating System

We address several possible attack vectors on a ROS-application:

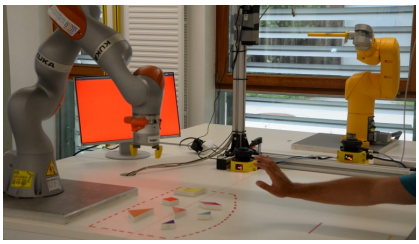
- Unauthorized Publishing (Injections)
- Unauthorized Data Access
- Denial of Service (DoS) attacks on specific ROS nodes.

As an example, we consider the following application (see figure 1): a collaborative robot is working alongside humans to perform a certain manipulation action (e.g., pick-and-place, assembly, etc.). Whenever a human comes close to the robot, it is supposed to slow down or even stop if the human is too close. The speed control of the robot is subscribing to the topic "safety/human_detection" which for the sake of simplicity just publishes a value in the set $\{0, 1, 2\}$ where 0 means no human is present, 1 means humans are in the collaborative zone (work with reduced speed) and 2 causes the robot to stop since a human is too close. A module like this might be realized using safety LIDARs or similar methods. An attacker who is aiming to cause harm to people near a robot (e.g., to

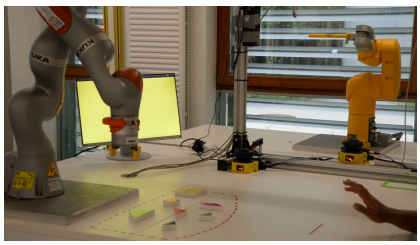
[5] `https://sri-csl.github.io/secure_ros/`



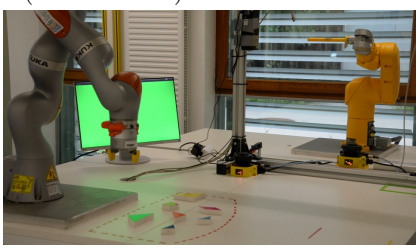
(a) Logical view of the example.



(b) The robot stops when a LIDAR sensor sends the message "safety/human_detection" with the values 2 (indicated red).



(c) A human in the collaborative zone causes the robot to run at reduced speed (indicated yellow).



(d) When no human is nearby, the robot runs at full speed (indicated green).

Figure 1: In this example application, the robot (the KUKA iiwa on the left) speed is influenced by safety LIDARs which detect the distance of a human to the robot (the LIDARs are the yellow cuboids between two robots). A visualization in the background indicates the current safety zone using colors.

damage the reputation of the company) is trying to inject false safety readings to pretend towards the robot that no person is nearby.

An attacker naturally does not want to be detected during the attack. Thus, from an attacker's point of view, it should be hard to spot the changes which are made to the system. If an attacker has such an additional stealth goal, in ROS this means that the attacker should not be visible in the ROS graph and if another node is sabotaged this should ideally also not be represented there. The term *ROS graph* means the connections between nodes constructed via publish/subscribe relations, i.e. if subscriber S subscribes topic `"sample/foo"` from publisher P, there is a connection between S and P in the ROS graph. This graph is constructed from the nodes known to the ROS master and specific information from the nodes on their communication connections.

3.1. Unauthorized Publishing (Injections)

A node in (plain) ROS may publish data for an arbitrary topic without prior authorization. This may be misused to inject data or commands into an application in order to disturb its operation. As an example, a robot might receive fake movement commands causing unpredictable motion that may harm

nearby persons or damage equipment (a video showing this behavior can be found in the supplemental material of [24]). Also, false sensor data might be injected into the system e.g., to fake a normal system state after a manipulation or to provoke a certain reaction of the robot.

In our example, the attacker pretends to be the publisher of the topic *"safety/human_detection"* to cause the robot to run at full speed despite a human being nearby. In this simple case, the fake publisher will always send the value 0 in disregard of the real sensor readings. This attack can also easily be extended to redundant sensor nodes where multiple ROS nodes run independently to provide safety information. Thus, it is recommended to implement redundant safety systems over different communication mechanisms, e.g., one node may communicate using ROS, another one might use Ethercat or a different communication medium.

3.2. Unauthorized Data Access

Every node in ROS may subscribe to every topic within the application. After that, it will receive any data that is published for this topic. This data can contain business-critical information or may be used to reverse-engineer a production process. This attack is especially hard to discover since a node itself may have no outgoing ROS communication.

In the example presented above, a malicious node may want to listen to the "safety/human_detection" topic to be informed when a human is nearby in order to trigger a certain action. However, it shall not be visible in the ROS graph, that there is an additional subscriber present.

3.3. Denial of Service

DoS attacks by publishing a large number of fake data can easily be launched in ROS. The subscriber of this message type will be flooded with bogus messages. This leads to a high processing load on all nodes and potentially to the inability of performing meaningful processing. Since there is no control over which node may publish what data, every node in the network may be used to publish data for a topic which a target node is subscribed to. Later, this can be used for a targeted DoS attack on that node.

In ROS, a node can be shut down using a simple XMLRPC call but doing this would violate the attacker's stealth goal since that node then disappears from the ROS graph. A different approach is to temporarily prevent that node from doing meaningful work by publishing a huge amount of data to it. As we will see in section 3.4, an attacker cannot only inject false data but can also prevent the subscriber from receiving the real data. This also enables person-in-the-middle attacks since a malicious node may act as a subscriber to a publisher and as a publisher to a subscriber and transparently record and manipulate the data flow between those two nodes.

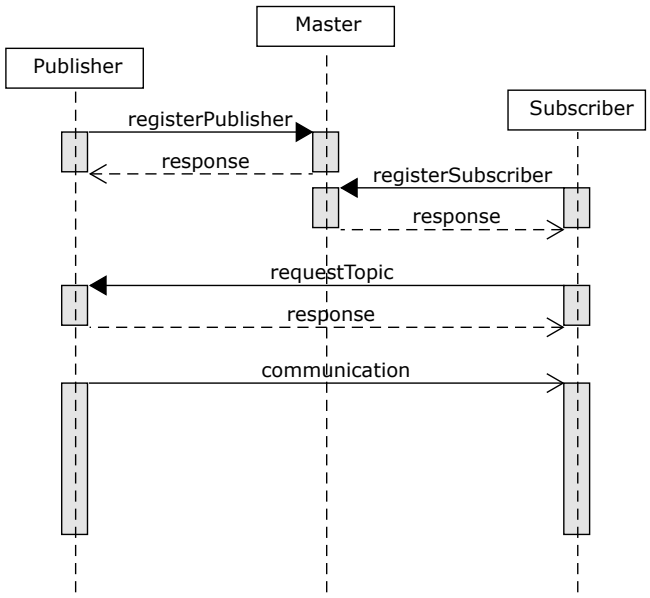


Figure 2: A sequence diagram showing the XMLRPC calls performed for registering publishers and subscribers as well as the XMLRPC communication handshake between two nodes.

3.4. Attacking ROS

In this section we show how an attacker can publish (faked) messages in a running ROS application.

To this end, we first look at (with the help of figure 2) how the default sequence of launching a ROS application works. At first, the ROS master must be started. Then publishers can register themselves at the master. To do this, a publisher calls the procedure `registerPublisher` at the master via XMLRPC with the topic names which the publisher is advertising, the XMLRPC URI information later needed by subscribers, as well as topic names and types as parameters. After receiving the response, the publisher is registered. Similarly, subscribers can register at the master by calling the XMLRPC procedure `registerSubscriber`. The response contains various parameters that enable the subscriber to contact the right publisher of each requested topic. With the XMLRPC call `requestTopic` to the publisher, the subscriber provides a list of desired protocols for the following communication. The publisher returns the selected protocol along with any additional parameters required for establishing a connection. Now, the subscriber is able to establish the channel (TCP or UDP) for receiving the published topic messages from the publisher. Figure 3 shows the communication between a publisher and subscriber after the initial XMLRPC call. In the TCP case, some header information is exchanged before actual topic data is sent by the publisher. In case of UDP communication, the publisher directly sends topic data without prior handshaking.

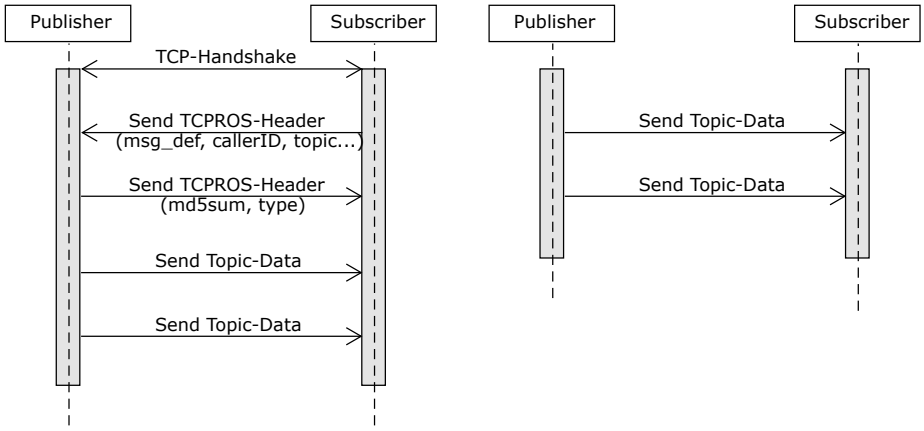


Figure 3: The communication workflow between two nodes after the XMLRPC handshake. The left diagram shows TCP communication, UDP is shown on the right side.

When consuming a service, no prior XMLRPC call is necessary and ROS does not support UDP-based communication with a service since consuming a service is a synchronous action.

Based on the knowledge of the data-flow, an attacker can now develop a workflow to misuse the communication channels in ROS in order to inject false data. The attack goal is to separate a subscriber from all its input publishers, without the subscriber, the publishers or the ROS core noticing that. Note that an attack with the intent to eavesdrop data has a very similar workflow and requires even less effort. Furthermore, we stress that the attack sequence we describe does not make use of any ROS tools (like `rostopic`) and thus needs no access to a ROS-enabled network node. We assume, that an attacker has already breached network security (e.g., firewall) to attack the application itself.

As a first step, the attacker needs to find the URI of the ROS master (if it cannot be read from the ROS_MASTER_URI environment variable). To do so, typically an Address Resolution Protocol (ARP) scan is performed to identify the hosts in the network. Then the master can be found by trying the ROS default port on each host until the master is found. If the master is configured to use a different network port, a full portscan on all nodes must be performed (which might already be detected by security systems and is thus the last measure taken by an attacker [31]). Figure 4 shows a sequence diagram of the attack on ROS. Once the master URI is known, we can use the XMLRPC API to reconstruct the ROS graph, i.e., a model that shows connections between nodes based on publish/subscribe relationships.

The attacker is now able to call the XMLRPC procedure `getSystemState` at the master. The master will return information about the registered publishers, subscribers and services. After extracting the name of the subscriber which should receive the (faked) messages, the attacker is able to send the XMLRPC

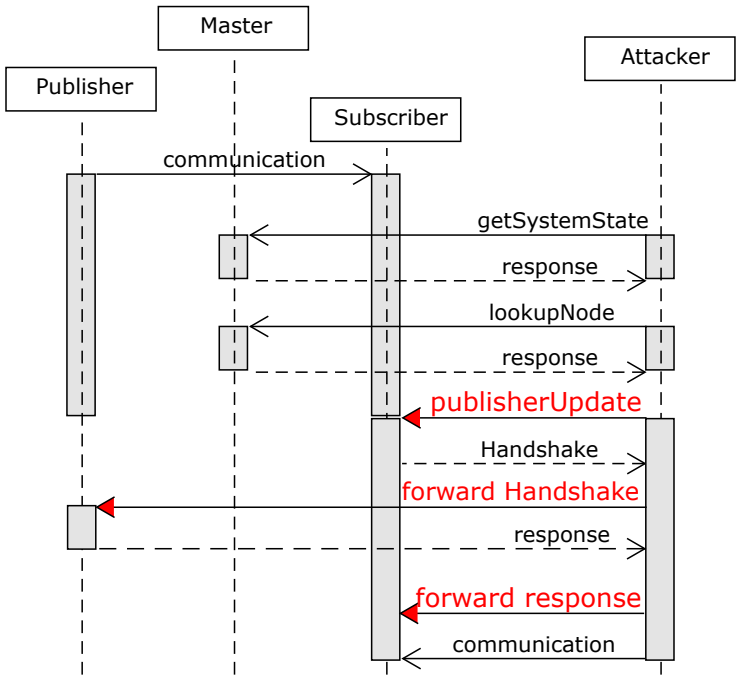


Figure 4: A sequence diagram of the communication flow when attacking a ROS application. The procedure calls marked in red shows the most critical points where the subscriber is isolated from its publisher using the `publisherUpdate` call.

message `lookupNode`, with the name of the node as parameter, to the master. The master will return the URI of the subscriber. With this URI, the attacker is able to send the XMLRPC message `publisherUpdate` to the subscriber. The parameters of this message are the topic and a list of the new publishers to the topic. The name of the topic is known from the information about the subscribers and the list of the new publishers contains the (fake) publisher created by the attacker. While the original publisher is still active and visible in the ROS graph, its data will not reach the subscriber anymore. Note that this attack is reversible, thus, after being done, the attacker can just send another `publisherUpdate` containing the original publishers.

3.4.1. Attack analysis

We now go through the individual steps of the attack and analyze what weaknesses they exploit. First, the XMLRPC API of ROS is not secured with authentication or authorization. Thus, an attacker can use the calls to the master to retrieve the ROS graph as well as communication parameters for each node and topic information. Second, a subscriber does not check if the `publisherUpdate` procedure is called by a valid client. It would be an improvement to just check that the calling host is indeed the host of the ROS master. However, a proper authentication and authorization should be preferred. Third,

a subscriber does not inform the master of a changed set of publishers. In a case, where the subscriber is isolated from its valid publishers, it will no longer be represented in the ROS graph. The attacker can also not be seen there for the same reason.

Note that an eavesdropping attack has a similar structure as an injection, thus the description here also holds true for this case. One additional form of attack would include the manipulation of parameters since it is possible to make targeted change to the parameters a specific node uses (from the ROS parameter server).

3.4.2. Hardening through configuration

Some obstacles for an attacker can already be constructed in the application configuration. First, changing the network port of the ROS master is the most obvious step. This forces an attacker to run a port scan on a broad port range in the network to find the master which can be detected by security systems.

In addition, shutting down the ROS master after the application is initialized can make it very hard for an attacker to interfere. However, this also drastically reduces the robustness of an application since a restarting (or late-starting) node cannot join the graph again. Further, the parameter server is no longer accessible to the ROS nodes.

4. Hardening ROS on the application level

A first approach to securing ROS can be realized at the application level. We introduce a dedicated Authentication Server (AS) which keeps track which ROS node may subscribe to or publish to a topic. In addition, the AS manages the authentication of nodes and generates the topic-specific encryption keys. We assume the AS to be subject of strong physical and logical access control, i.e., it can be mounted and running in a high(er) security domain. The respective security precautions are relatively stronger, but focused on a single component. This buys us security for a large and highly distributed system at the cost of strong security at only one point (the AS).

The publish/subscribe paradigm can be reformulated as a broadcasting communication pattern. Many publishers of a certain topic may be passing data to multiple subscribers. Thus, methods from broadcast encryption may be applied to the problem where this broadcasting needs to be done in a secure and authentic way. While it is not too difficult to establish the necessary cryptographic operations, those nevertheless need to be incorporated at every step of the process. For this reason, we divide the architecture description into phases according to the lifecycle of a publisher and subscriber, and describe the relevant cryptographic operations per phase.

To start, let us assume that every possible message being transmitted by a publisher can be classified to fall into one out of a finite number N of topics. Let us reference these topics by indices $i \in \{1, 2, \ldots, N\}$ in the following. We will enforce a publisher to specify the topic (or several topics) from which messages

are to be expected. This specification is done once during the registration, and then kept fixed for the lifetime of the publisher. Every other topic from the publisher will be rejected by the AS. The relevant details of the registration process are expanded in the next section.

Hereafter, we write $E(m, k)$ to mean the encryption (symmetric or asymmetric) of a message m under a key k. For asymmetric cryptography, we write pk, sk to mean the public and private key of an entity. For digital signatures, let $\texttt{sign}(m, sk)$ be the signature function taking a message m and private key sk to output a signature s. That signature s can be verified by a function $\texttt{verify}(s, pk) \in \{\texttt{true}, \texttt{false}\}$ that takes the public verification key as an additional input to the signature, and outputs either `true` or `false`, depending on whether or not the signature was cryptographically valid. In the following we let our description be abstract, yet emphasize that possible cryptographic schemes are AES for symmetric encryption, and RSA to handle asymmetric matters. The symbol $x \| y$ means the concatenation of the data items x and y in a way so that x and y can both be recovered uniquely from the compound representation $x \| y$. Usually, this will be a humble string concatenation, with a proper separator symbol.

4.1. Registration of a new ROS node

A ROS node starting up has to authenticate itself to the AS before it can join the ROS graph. To do this, it first runs a (digital signature based) challenge-response authentication with the AS to certify itself as a legitimate new node. The set of nodes in the application is assumed to be known at configuration time and is presented to the AS as list of certificates. Note that working with a pre-known set of nodes (i.e., a fixed communication graph) is also the approach of SROS[6]. Specifically, suppose each known (trusted) source S is known to the authentication server as a cryptographic (X.509) certificate, containing a public signature verification key pk_S. Assume that a node N, affiliated to a trusted source S, wants to register itself in the AS, then it can only do so upon successful completion of the steps detailed in Figure 5.

After the successful authentication of the new node to the AS, the same procedure is done from the AS to the node to avoid a person-in-the-middle situation. In such an attack, acting as proxy, an attacker could trick the AS or the node to send their data through this proxy resulting in it having full access to the data stream.

After this procedure, node-type specific communication with the AS follows. For a subscriber, the AS sends the subscriber a (digitally signed) list of public signature verification keys related to publishers of the message topics that the subscriber has registered for. The rationale is that every publisher is obliged to digitally sign its messages for authenticity, since a subscriber will drop messages under the following circumstances:

[6] http://wiki.ros.org/sros

1. Along with the registration request, N submits a public key certificate[a] $Z = (N, pk_N, S, \texttt{sign}(N \| pk_N \| S, sk_S))$ to the AS. In particular, the certificate Z thus tells the AS who the source S of the new node is.
2. The AS looks up the authentically stored public signature verification key of S, and verifies the certificate by checking if $\texttt{verify}(Z, pk_S) \stackrel{?}{=} \texttt{true}$. Iff so, then it sends a random number r to N, which N digitally signs with its secret key sk_N that belongs to the public signature verification key pk_N. The new node candidate N then replies by sending the signature $sig = \texttt{sign}(r, sk_N)$ back to the AS.
3. As before, the AS takes the (authenticated) public key pk_N to verify that the signature on r is correct. That is, it accepts the new node N iff $\texttt{verify}(sig, pk_N) = \texttt{true}$.

[a]Note that the content of the certificate is intentionally restricted to only the relevant contents; the real certificates would have a much richer and complex structure.

Figure 5: Simple Certificate Based Challenge-Response Authentication.

- the digital signature is missing or invalid
- the digital signature does not come from a previously known publisher. Note that looking up the signature verification key in the list given by the AS means that the AS has taken care of the identity check of the publisher previously. Thus, the subscriber's trust in the publisher is based on its trust in the AS (to have properly completed the authentication), and the trust in the digital signature.

For publishers, the AS needs to handle the distribution of encryption keys for topics. Assume that the new publisher P has registered to send messages under topic i (where i identifies some message topic). For each such message topic, the AS maintains an individual session key K_i. Every publisher that registers for messages of topic i is given the respective session key(s) K_i (along the authentication), under which it can encrypt its data and publish it to all subscribers. The subscriber, upon its registration for the same topic i, gets the same session key K_i from the AS.

If publisher P wants to broadcast a data item in an authenticated fashion, it completes the following tasks:

1. it attaches its identity P to the data item m (belonging to topic i) and encrypts $P \| m$ under the session key K_i into a ciphertext $c = E(P \| m, K_i)$.
2. it digitally signs the data item under its private signature key sk_P, thus getting a signature value $s = \texttt{sign}(c, sk_P)$.
3. it attaches the topic i to the compound packet and broadcasts the tuple (i, c, s) to all subscribers.

Upon reception of a digitally signed message $M = (i, c, s)$, a subscriber parses M and completes the following steps:

1. it deciphers c using the known secret key K_i to retrieve the sender's identity P and the data item m (the correct session key K_i is indicated by the first entry in M). Note that this step is only possible if the subscriber has previously registered for messages of that particular topic i (the key K_i was given during the registration); if not, then M can be dropped immediately before any decryption attempts.
2. it verifies the digital signature s using the respective public key pk_P of the identity obtained in the first step (this spares the subscriber to work through the entire list of potential publishers for that message item).
3. it accepts the data item m if and only if m deciphered correctly under K_i, and the digital signature s on $E(P\|m, K_i)$ has been verified correctly.

4.2. Discussion of application-level security

With the presented application-level approach for ROS security, we can already tackle some of the security vulnerabilities. We can prevent unauthorized nodes from publishing and subscribing and thus from injecting false data and from eavesdropping. This is achieved by topic-specific encryption keys which are only handed out to authorized application modules.

Still some insufficiencies persist which cannot be handled on the application level alone. First, only the message content is encrypted, not the message headers. This still allows for frequency analysis of certain message types. Second, the application-level approach cannot prevent nodes from joining the ROS graph or from publishing (even though meaningless) messages to the application which still enables Denial of Service (DoS) attacks on specific nodes. Third, this approach cannot ultimately regulate a node's subscription to arbitrary topics. Thus, all messages of a certain topic will be delivered to it. Our approach only ensures that this subscriber cannot read the message contents without the proper decryption key. Finally, the application-level approach requires nodes to be rewritten to incorporate the security protocol. This is realistic for small applications which do not need external ROS modules from other developers. Still, a more transparent solution is required for more complex applications.

In the next section, we present such an approach which describes the modification of the ROS source code itself to make the changes transparent to existing and new nodes.

5. Securing the ROS communication channel

While the application-level approach already mitigates many security risks in ROS, it also requires each ROS node to implement the security functions thus requiring all modules to be recompiled. To make the security function transparent to application-level modules, we need to modify the ROS communication itself. To overcome the limitations that a purely application-level approach inherently has, we present a modified version of the ROS communication between

nodes which has been extended with authentication and confidentiality. We focus on the direct communication between two nodes (be it publishers and subscribers or services/actions and their clients) to ensure security already at this lowest level.

5.1. Secure channel design

The following changes to the ROS core have been implemented in ROS Kinetic. Note, that a backport to older versions should be simple since the communication core has not been changed in a longer period of time [7]. The security concept described here works for all types of ROS communication i.e., for publish/subscribe, actions and services.

The basic approach we take to secure the communication is to use Transport Layer Security (TLS) [32] and Datagram Transport Layer Security (DTLS) [33]. In addition we perform authorization for each node on a per-topic basis. (D)TLS works in a similar manner like our application-level protocol presented above. It first performs a handshake using public-key cryptography and then uses symmetric encryption during the communication to ensure data confidentiality. Data integrity is secured by using Message Authentication Codes (MACs). After the (D)TLS handshake is complete, we perform fine-grained authorization of each node to ensure that topics are only accessible by trusted nodes. We use X.509 certificates which encode node identity and public keys along with authorization information (e.g., the topics which may be published or subscribed by this node).

The communication flow between master and ROS nodes has already been described in section 3.4. Our approach is located after the initial XMLRPC call from subscriber to publisher (or at the communication start between client and service/action). After the TLS connection is established, the exchanged certificates are used to determine the authorization of each node for this type of communication. This is done under consideration of the type of communication pattern i.e., a publisher checks if a subscriber is allowed to consume its data, a subscriber checks if the data of a publisher may be processed and a service checks if it may be consumed by another node. If the validation of a node certificate fails, no further communication is performed.

We incorporate this procedure directly into the TCP/UDP channel of ROS which is implemented in the `roscpp` package. For both, TCP and UDP, an initial handshake is performed before headers are exchanged. This TCPROS handshake is used to exchange headers and topic data describing the message definitions, caller IDs and the topic. By performing the TLS handshake first, the TCP channel handshake of the ROS communication protocol is already secured. After that, all communication data is rerouted through the respective encryption and decryption facilities. To perform the TLS handshake, authorization and authentication, X.509 certificates are used.

[7] compare the `connection.cpp` file of ROS on github`https://github.com/ros/ros_comm/commits/lunar-devel/clients/roscpp/src/libros/connection.cpp`

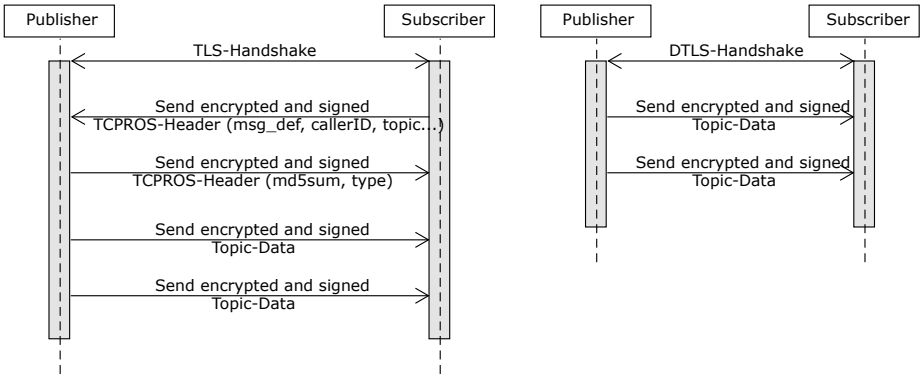


Figure 6: The new, secured communication pattern between two nodes. Again, we show TCP on the left and UDP on the right side. Note, that the UDP communication is now bidirectional and that for both, TCP and UDP, (D)TLS handshakes are performed before any further communication.

For UDP, in the plain ROS implementation the publisher just starts (after the XMLRPC call) by sending a UDP package containing the topic data from the publisher to the subscriber. Afterwards, UDP packages are sent whenever new data is published. Thus, in this implementation, the UDP communication is unidirectional from publisher to subscriber. To secure the channel using DTLS, again a handshake is necessary. This handshake needs bidirectional data exchange for the challenge-response protocol. Thus, the publisher must open an UDP server socket to receive datagrams from the subscriber. To secure already the first UDP data packet from the publisher to the subscriber, the DTLS handshake is performed in parallel to the XMLRPC call from the subscriber to the publisher. The XMLRPC call is synchronized to this handshake and thus, it is not finished before the DTLS handshake is complete. This is a rather invasive change to the default ROS communication protocol but it is necessary to provide secure UDP communication. Afterwards, the dataflow is unidirectional from publisher to subscriber again.

Figure 6 shows the new communication flow between in the secure implementation for TCP and UDP. Comparing this to the original flow shown in figure 3 it can be seen that (D)TLS handshakes are performed first to also secure header exchange. In case of UDP communication, also an additional bidirectional communication had to be introduced to perform the handshake.

Any future communication between the two nodes for this specific topic is then secured using the (D)TLS channel. Thus, all data which is exchanged is encrypted and signed resulting in a confidential and trusted communication between the nodes.

5.2. Discussion of the secure communication channel

The secure communication channel ensures confidentiality and integrity of messages and increases the availability by minimizing the attack surface for

DoS attacks. It can transparently be used with existing nodes without the need for recompilation. Nodes need valid certificates to perform the (D)TLS handshake at the beginning of their communication. The absence of a certificate prevents an attacker from contacting any other node. Performance evaluations in comparison with the unmodified ROS communication core can be found in [27].

This approach however, does not secure the XMLRPC API of ROS. Thus, an attacker can still retrieve the node graph, send `publisherUpdates` and shutdown nodes. In ROS the master-side of that API is implemented in Python, thus the approach to only modify `roscpp` cannot cover this part. In combination with the SROS and/or ROSRV[25] projects however, it is possible to achieve security at both levels and in addition incorporate python nodes. Showing a successful demonstration of this combination is part of our future work.

6. Penetration testing

Penetration testing [34] is a very popular method to test the vulnerability of a system to certain attacks. In this section we present steps to penetrationtest a ROS system and what the resulting consequences are for applications with and without security. Note that we do not describe a penetration test for the network security but just for the ROS application itself. Also, note that we do not test the vulnerabilities in the XMLRPC API, since at the moment, this can only be secured using SROS or ROSRV.

To facilitate penetration testing of ROS applications, we have developed RosPenTo, a tool for (semi-)automated penetration testing specifically for ROS. It exploits the inconsistencies and missing authentication and authorization in the XMLRPC-API of ROS. It works analogous to the flow described in 4. RosPenTo can be used in automatic mode e.g., for use in course of continuous integration tasks, or manually with a user interface to explore and manipulate a running application. With RosPenTo it is easy to show how to ROS applications can be manipulated without that attack being recognized by others. It allows for the isolation of publishers and subscribers and also for the injection of false data.

The validation of the proposed architecture is driven by the attacks described in precursor work and reports on weaknesses of ROS (see Section 3). The respective experimental design was towards reproducing the conditions under which known weaknesses of ROS were to become exploitable, and then run the attack (as the literature describes it) on the hardened version of ROS as described in this work. The outcome was recorded as a success, if the attack failed.

We show a comparison of effects for attack actions on ROS without security, with application-level security and secure communication channel in table 1. It can be seen that in default ROS an attacker cannot be hindered in the execution of actions. When implementing security on the application level, the attacker may still subscribe or publish but received data cannot be interpreted and sent data will be ignored due to the missing (or not verifiable) signature.

	Effect		
Attacker action	**Unmodified ROS**	**Application-layer security**	**Secure communication channel**
Subscribe	Subscription is performed	Subscription is performed but message contents cannot be interpreted	Without valid certificate, communication is cancelled due to failed (D)TLS handshake.
Publish	Data is received and interpreted	Message is received but will be rejected due to invalid signature	Subscriber will not perform subscription since (D)TLS handshake fails
Consume service	Service can be called normally	Service result cannot be consumed.	(D)TLS handshake fails.
Advertise service	Service is consumed regularly	Result cannot be consumed by other nodes.	(D)TLS handshake fails.
Unauthorized access to topic (with valid certificate)	Access granted	Topic key not transferred by AS.	Communication canceled due to access to unauthorized topic.

Table 1: Comparison of reaction to attacks between unmodified ROS, application-level secured ROS and ROS with secure communication channel.

The communication channel security makes sure that illegitimate nodes are excluded already at the start of the communication (due to their missing valid certificate). In cases where (by whatever means) an attacker can present a valid certificate (e.g., of another node), both security approaches use fine-grained topic access control mechanisms to ensure that only topics which previously have been granted may be subscribed to or published.

7. Outlook

7.1. Usable key management

When securing an IT system, the process that establishes cryptographic keys whenever they are needed is crucial for the overall security. Hence we describe essential steps in the so called key management process in the remainder of this section. These steps include

- the enrollment of new ROS components,

- the de-registration of (old or broken) components, and
- the remote and on-site maintenance

Additionally to these steps we will briefly discuss the prerequisites and a simple technique to reduce the memory requirements especially for public keys.

7.1.1. Prerequisites

During manufacturing, each components is provided an X.509 certificate, containing a unique device identifier (DID) (subject name in Figure 7), a public key, and other data, and the corresponding private key. Ideally, the private key is stored in a trusted device (e.g., a trusted platform module (TPM)-like module). Note that for security reasons, separate functions (or groups of functions) should be protected by separate keys (or key pairs in case of asymmetric cryptography).

7.1.2. Enrollment and of new Components

For enrollment, the component sends a standardized and timestamped registration request to the Master. The Master verifies the certificate of the new component and checks the signature of the enrollment request with the public key of the component. After validating the request, the Master checks if the DID is on the list of valid devices. If so, the Master opens a TLS session with the new component. Ideally, a security suite providing perfect forward security should be used (e.g. ECDHE-ECDSA-AES256-GCM-SHA384), i.e. that the public key of the component is only used for authenticating parameters of the key agreement (ECDHE in the example), but not for encryption of a session key. As soon as the TLS session is established, the Master replaces all pre-installed keys (aka transport keys) and certificates by new ones.

Note that a malicious manufacturer – depending on the key exchange mechanism (e.g. whenever RSA is used for authentication and key-exchange) – that can log all messages transmitted during the replacement of the transport keys can still get hold of the new keys (and certificates). The only way to inhibit this attack is to run the key replacement protocol in a closed environment.

7.1.3. De-registration of Components

For the de-registration of a component, its DID and its certificate serial number are simply removed from the list of valid devices stored at the Master node (and cached at nodes with sufficient memory). From this moment on, no other component will accept messages from the de-registered device, because the check of the corresponding certificate will fail.

7.1.4. Maintenance of Components

If the configuration of a device has to be changed, this can easily be done remotely over a TLS-secured channel. If the communication link to the device is broken, or a technician has to interact directly with a component for any reason, the following protocol for offline authentication can be used. Similar to the authentication triplets (see [35, 36]) used during the authentication of a GSM

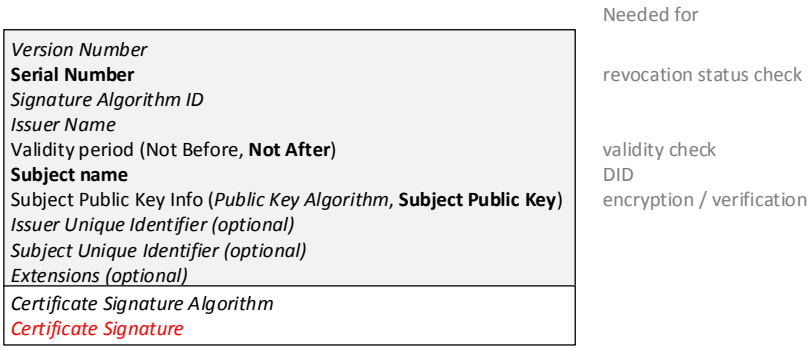


Figure 7: Elements of an X.509 certificate.

cellphone against the (untrusted) base station, the technician (respectively his maintenance device) only gets a challenge, a response, and a session key. The maintenance device does not get a secret key, normally used during the challenge response protocol run. To authenticate, the maintenance device sends the challenge and the response to the component. The component verifies that the response is based on the challenge and the locally stored authentication key (or in case of asymmetric cryptography the public verification key). After successful verification, the component derives the session key and hence a secured channel between the component and the maintenance device can be established. If secured means encrypted and authenticated, the scheme can easily be extended by a derived or provided authentication key.

7.1.5. Memory saving Storage of Public Keys

A classical X.509 certificate most commonly contains the data depicted in Figure 7: After verifying a certificate the first time, unnecessary information (formatted in italic above) can be dismissed, especially the certificate signature. This reduces the memory demand to approximately 50%. Since the certificate signature does no longer protect the authenticity of the data, the authenticity has to be ensured by other means (like internal storage of a trusted device).

7.2. Accountability

A system is accountable if it can give evidence of its actions and this evidence can be understood by a (human) investigator. It allows the investigator to pinpoint the cause of a problem or the deviation from a (security) protocol. Evidence obtained from such an *accountable system* can be used to understand and improve the system, for instance by helping to trace bugs, or to pursue legal action against either the manufacturer of a system or an attacker, for example by providing forensic evidence. Additionally, while an *accountability mechanism* does not help to prevent unwanted events, it helps to detect them, analyze the root cause and develop measures to prevent future occurrences of this unwanted event.

In addition to the reversal of the CIA-triad elaborated on in the introduction, ICSs are often resource-constrained systems that are not capable of performing advanced encryption methods, they may not be connected to the internet and thus cannot receive security updates or be informed about revoked certificates. They are often required to support emergency overrides ("break the glass policy") and need to be easy to repair and replace. For example, an insulin pump needs to support manual overrides in case of life threatening situations and its power supply must be exchangeable without any, often time-consuming, authentication during an emergency. Such requirements are impossible to fulfill with just the classical CIA-trinity. With an accountability infrastructure in place it is, however, possible to allow overrides by, for example, swiping a badge. The action is logged and the proportionality of the measure can be judged after the critical situation is over.

Another challenge is the rise of highly automated and even fully autonomous ICSs. Such systems make decisions based on their sensor input and no longer have a human in the loop. It is impossible to cover all potential corner cases of a system's operation environment and thus such systems need mechanisms to reconstruct their decision process, analyze an accident and, if necessary, assign blame for unwanted events. A widely discussed class of such systems are, for example, autonomous cars and their accidents.

All these constraints cannot be fulfilled with traditional information security measures alone [37] and require new solutions (e.g., by continuously evaluating the current "security level" of a system [38]). To cover the attack surface left by traditional measures, we will extend ROS with facilities to support accountability. We are also considering the possibility to reduce some traditional measures (e.g., authentication) and cover the shortfall with accountability. Forgoing some measures can in some cases have tangible benefits. For instance, forgoing encryption can yield significant performance gains and skipping the authentication of service personal or the integrity checks of replacement parts can make system maintenance faster, easier, more reliable and ultimately cheaper.

A prerequisite for accountability is a secure and redundant logging infrastructure. To fill this gap, we are currently working on extending ROS with a block chain based logging mechanism that ensures the authenticity, tamper evidence and availability of the log messages, for instance by distributing the logs over all ROS nodes.

The other problem is to define meaningful log messages, develop ways to (semi-)automatically reason about them and to find the right trade-off between log volume and log completeness. Ideally, such a system should only log a minimum of relevant events. This trade-off is especially important from a privacy and data protection point of view (see for example the new EU regulation on privacy protection [39]). In many jurisdictions not everything that can be recorded may be legal to record. Especially when recording personal information, it is often necessary to prove the relevance of the data. This is one of the rare cases, where the needs of engineers and privacy advocates align: both want to log only a minimal amount of information. Engineers work with resource constrained devices and often have too much data to process and the cardinal rule

of data protection is to log only minimal data. We investigate these problems in an interdisciplinary project together with legal scholars and work towards guidelines and best practices for law abiding, yet sufficiently powerful logging mechanisms for ROS.

8. Conclusion

In this paper we have given an overview of security issues in ROS and have presented approaches to solve them. We have shown how easy an attacker can make use of weaknesses in the ROS design. Our application-level approach secures small ROS applications without the need to dig deep into the ROS sourcecode. A transparent solution to secure `roscpp`-based nodes has been presented. Here, we modify the sourcecode of ROS to establish (D)TLS channels ensuring authentication, authorization and confidentiality of information exchange between nodes. In combination with e.g., SROS or ROSRV, an even more comprehensive security solution can be built.

In future work we will concentrate –as presented above– on usable key management and accountability. Aside from that, we will work on integrating our secure channel with SROS and extend it with further security functions such as signed logging and hardware-based cryptography.

Acknowledgements

The work reported in this article has been supported by the Austrian Ministry for Transport, Innovation and Technology (bmvit) within the project framework Collaborative Robotics and by the Munich Center for Internet Research (MCIR).

References

[1] F. Pasqualetti, F. Dörfler, F. Bullo, Attack detection and identification in cyber-physical systems, IEEE Transactions on Automatic Control 58 (11) (2013) 2715–2729. doi:10.1109/TAC.2013.2266831.

[2] J. Weiss, Industrial Control System (ICS) Cyber Security for Water and Wastewater Systems, Springer International Publishing, Cham, 2014, pp. 87–105. doi:10.1007/978-3-319-01092-2_3.
URL http://dx.doi.org/10.1007/978-3-319-01092-2_3

[3] S. Karnouskos, Stuxnet worm impact on industrial cyber-physical system security, in: 37th Annual Conference of the IEEE Industrial Electronics Society (IECON 2011), 2011, pp. 4490–4494.

[4] N. Nelson, The impact of dragonfly malware on industrial control systems, Tech. rep., SANS Institute (2016).

[5] J. McClean, C. Stull, C. Farrar, D. Mascareas, A preliminary cyber-physical security assessment of the robot operating system (ros), in: Proc. SPIE, Vol. 8741, 2013, pp. 874110–874110–8. doi:10.1117/12.2016189.
URL http://dx.doi.org/10.1117/12.2016189

[6] M. Cheminod, L. Durante, A. Valenzano, Review of security issues in industrial networks, Industrial Informatics, IEEE Transactions on 9 (1) (2013) 277–293. doi:10.1109/TII.2012.2198666.

[7] B. Miller, D. Rowe, A Survey of SCADA and Critical Infrastructure Incidents, in: Proceedings of the 1st Annual Conference on Research in Information Technology, RIIT '12, ACM, New York, NY, USA, 2012, pp. 51–56. doi:10.1145/2380790.2380805.
URL http://doi.acm.org/10.1145/2380790.2380805

[8] E. Byres, P. E. Dr, D. Hoffman, The myths and facts behind cyber security risks for industrial control systems, in: In Proc. of VDE Kongress, 2004.

[9] S. Shin, T. Kwon, G.-Y. Jo, Y. Park, H. Rhy, An experimental study of hierarchical intrusion detection for wireless industrial sensor networks, Industrial Informatics, IEEE Transactions on 6 (4) (2010) 744–757. doi:10.1109/TII.2010.2051556.

[10] L. Maglaras, J. Jiang, Intrusion detection in scada systems using machine learning techniques, in: Science and Information Conference (SAI), 2014, 2014, pp. 626–631. doi:10.1109/SAI.2014.6918252.

[11] P. Fairley, Cybersecurity at u.s. utilities due for an upgrade: Tech to detect intrusions into industrial control systems will be mandatory [news], IEEE Spectrum 53 (5) (2016) 11–13. doi:10.1109/MSPEC.2016.7459104.

[12] R. Toris, C. Shue, S. Chernova, Message authentication codes for secure remote non-native client connections to ROS enabled robots, in: IEEE International Conference on Technologies for Practical Robot Applications (TePRA), 2014, pp. 1–6. doi:10.1109/TePRA.2014.6869141.

[13] W. Adi, Mechatronic security and robot authentication, in: Bio-inspired Learning and Intelligent Systems for Security, 2009. BLISS '09. Symposium on, Institute of Electrical and Electronics Engineers (IEEE), 2009, pp. 77–82. doi:10.1109/BLISS.2009.30.

[14] D. Dzung, M. Naedele, T. von Hoff, M. Crevatin, Security for industrial communication systems, Proceedings of the IEEE 93 (6) (2005) 1152–1177. doi:10.1109/JPROC.2005.849714.

[15] C. Wang, A. Carzaniga, D. Evans, A. Wolf, Security issues and requirements for internet-scale publish-subscribe systems, in: System Sciences, 2002. HICSS. Proceedings of the 35th Annual Hawaii International Conference on, 2002, pp. 3940–3947. doi:10.1109/HICSS.2002.994531.

[16] C. Esposito, M. Ciampi, On security in publish/subscribe services: A survey, IEEE Communications Surveys Tutorials 17 (2) (2015) 966–997. doi:10.1109/COMST.2014.2364616.

[17] A. V. Uzunov, A survey of security solutions for distributed publish/subscribe systems, Comput. Secur. 61 (2016) 94–129.

[18] A. V. Uzunov, E. B. Fernandez, K. Falkner, Security solution frames and security patterns for authorization in distributed, collaborative systems, Comput. Secur. 55 (2015) 193–234. doi:10.1016/j.cose.2015.08.003.

[19] Object Management Group, Data Distribution Service, v1.4, version 1.4 (2014).

[20] Object Management Group, DDS Security Specification, version 1.0 Beta2 (2016).

[21] Object Management Group, Real-Time Publish-Subscribe Protocol (RTPS) DDS Interoperability Wire Protocol, version 2.2 (2016).

[22] F. J. R. Lera, J. Balsa, F. Casado, C. Fernández, F. M. Rico, V. Matellán, Cybersecurity in autonomous systems: Evaluating the performance of hardening ros, in: Workshop on physical Agents, 2016, p. 47.

[23] R. Dczi, F. Kis, B. St, V. Pser, G. Kronreif, E. Jsvai, M. Kozlovszky, Increasing ros 1.x communication security for medical surgery robot, in: 2016 IEEE International Conference on Systems, Man, and Cybernetics (SMC), 2016, pp. 004444–004449. doi:10.1109/SMC.2016.7844931.

[24] B. Dieber, S. Kacianka, S. Rass, P. Schartner, Application-level security for ROS-based applications, in: 2016 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS), 2016, pp. 4477–4482. doi:10.1109/IROS.2016.7759659.

[25] J. Huang, C. Erdogan, Y. Zhang, B. Moore, Q. Luo, A. Sundaresan, G. Rosu, ROSRV: Runtime Verification for Robots, Springer International Publishing, Cham, 2014, pp. 247–254. doi:10.1007/978-3-319-11164-3_20.
URL http://dx.doi.org/10.1007/978-3-319-11164-3_20

[26] R. White, H. I. Christensen, M. Quigley, SROS: Securing ROS over the wire, in the graph, and through the kernel, in: HUMANOIDS 16 Workshop Towards Humanoid Robots OS, 2016.

[27] B. Breiling, B. Dieber, P. Schartner, Secure communication for the robot operating system, in: To appear in Proceedings of the 11th IEEE International Systems Conference, 2017.

[28] OPC Foundation, Unified Architecture Part 2: Security Model, version 1.03 (2015).

[29] S. Tarkoma, Publish - Subscribe Systems: Design and Principles, 1st Edition, Wiley series in communications networking & distributed systems, Wiley, s.l., 2012. doi:10.1002/9781118354261.
URL http://search.ebscohost.com/login.aspx?direct=true&scope=site&db=nlebk&db=nlabk&AN=465217

[30] K. Visala, D. Lagutin, S. Tarkoma, Security design for an inter-domain publish/subscribe architecture, in: The Future Internet, Vol. 6656 of Lecture Notes in Computer Science, Springer Berlin Heidelberg, Berlin, Heidelberg, 2011, pp. 167–176. doi:10.1007/978-3-642-20898-0{_}12.

[31] M. H. Bhuyan, D. Bhattacharyya, J. Kalita, Surveying port scans and their detection methodologies, The Computer Journal 54 (10) (2011) 1565. arXiv:/oup/backfile/Content_public/Journal/comjnl/54/10/10.1093/comjnl/bxr035/2/bxr035.pdf, doi:10.1093/comjnl/bxr035.
URL +http://dx.doi.org/10.1093/comjnl/bxr035

[32] T. Dierks, E. Rescorla, The Transport Layer Security (TLS) Protocol Version 1.2, RFC 5246 (Proposed Standard), updated by RFCs 5746, 5878, 6176, 7465, 7507, 7568, 7627, 7685, 7905, 7919 (Aug. 2008).
URL http://www.ietf.org/rfc/rfc5246.txt

[33] E. Rescorla, N. Modadugu, Datagram Transport Layer Security Version 1.2, RFC 6347 (Proposed Standard), updated by RFCs 7507, 7905 (Aug. 2012).
URL http://www.ietf.org/rfc/rfc6347.txt

[34] B. Arkin, S. Stender, G. McGraw, Software penetration testing, IEEE Security Privacy 3 (1) (2005) 84–87. doi:10.1109/MSP.2005.23.

[35] ETSI TS 100 929 V8.6.0 (2008-01) Technical Specification: Digital cellular telecommunications system (Phase 2+); Security-related network functions (3GPP TS 03.20 version 8.6.0 Release 1999) (2008).

[36] ETSI TS 133 102 V8.6.0 (2010-04) Technical Specification: Universal Mobile Telecommunications System (UMTS); LTE; 3G security; Security architecture (3GPP TS 33.102 version 8.6.0 Release 8) (2008).

[37] A. Cardenas, S. Amin, B. Sinopoli, A. Giani, A. Perrig, S. Sastry, Challenges for securing cyber physical systems, in: Workshop on Future Directions in Cyber-physical Systems Security, DHS, 2009.
URL http://chess.eecs.berkeley.edu/pubs/601.html

[38] W. Knowles, D. Prince, D. Hutchison, J. F. P. Disso, K. Jones, A survey of cyber security management in industrial control systems, International Journal of Critical Infrastructure Protection 9 (2015) 52 – 80. doi:http://dx.doi.org/10.1016/j.ijcip.2015.02.002.
URL http://www.sciencedirect.com/science/article/pii/S1874548215000207

[39] European Parliament and Council, Council regulation (EU) no 2016/679, http://eur-lex.europa.eu/legal-content/EN/TXT/?uri=uriserv:OJ.L_.2016.119.01.0001.01.ENG&toc=OJ:L:2016:119:TOC (2016).

Acronyms

AS Authentication Server

ROS Robot Operating System

TPM trusted platform module

MAC Message Authentication Code

DoS Denial of Service

DDS Data Distribution System

ICS Industrial Control System

APT Advanced Persistent Threat

CIA Confidentiality, Integrity, Availability

OMG Object Management Group

OPC-UA Open Platform Communication - Unified Architecture

XMLRPC XML Remote Procedure Call

ARP Address Resolution Protocol

DID unique device identifier

Authors

Bernhard Dieber is heading the research group 'Robotic Systems' at the Institute for Robotics and Mechatronics of JOANNEUM RESEARCH. He received his Masters degree in applied computer science and PhD in information technology from the Alpen-Adria Universität Klagenfurt. His research interests include robotics software, security and dependability of robotic systems, visual sensor networks and middleware.

Benjamin Breiling is a Junior Researcher at the Robotic Systems group at the Institute of Robotics and Mechatronics at JOANNEUM RESEARCH. He received his Masters degree from Alpen-Adria Universitt Klagenfurt. His research interests include security architectures, robotic security and security of middleware systems.

Sebastian Taurer is a Research Intern at the Robotic Systems group at the Institute of Robotics and Mechatronics at JOANNEUM RESEARCH. His research interests include penetration testing of robotic security, security architectures and ROS security.

Severin Kacianka received his Master's degree from the Alpen-Adria-University Klagenfurt in 2014. In his thesis he worked on video streaming for Unmanned Aerial Vehicle networks. In 2015 he joined the Chair of Software Engineering at the Technical University of Munich as a PhD student. In his work he focuses on the security and accountability of Cyber-Phyiscal Systems.

Stefan Rass graduated with a double master degree in mathematics and computer science from the Alpen-Adria Universitt Klagenfurt (AAU) in 2005. He received a PhD degree in mathematics in 2009, and habilitated on applied computer science and system security in 2014. His research interests include applied system security, as well as complexity theory, statistics, decision theory and game-theory. He authored numerous papers related to security and applied statistics and decision theory in security. Closely related to the project is his (co-authored) book Cryptography for Security and Privacy in Cloud Computing, published by Artech House. He participated in various nationally and internationally funded research projects. Currently, he is an associate professor at the AAU, teaching courses on theoretical computer science, complexity theory, security and cryptography.

Peter Schartner received the master's degree in Telematics from the Technical University of Graz in 1997 with a focus on information security and the Ph.D. degree in computer science from the Alpen-Adria-Universitt Klagenfurt in 2001 with a focus on security tokens. He participated in various nationally and internationally funded research projects. He is currently an Associate Professor with the System Security Research Group, Alpen-Adria-Universitt Klagenfurt, where he is involved in theoretical computer science, algorithms and data structures, security, and cryptography. He is also a Lecturer with the Trier University of Applied Sciences. His research interests include applied system security, key management, security infrastructures, and applications for security tokens, especially smartcards.