Secure communication for the Robot Operating System

Benjamin Breiling
JOANNEUM RESEARCH
Institute for Robotics and Mechatronics
Klagenfurt am Wörthersee, Austria
Email: benjamin.breiling@joanneum.at

Bernhard Dieber
JOANNEUM RESEARCH
Institute for Robotics and Mechatronics
Klagenfurt am Wörthersee, Austria
Email: bernhard.dieber@joanneum.at

Peter Schartner
Alpen-Adria-Universität Klagenfurt
Institute of Applied Informatics
Klagenfurt am Wörthersee, Austria
Email: peter.schartner@aau.at

***Abstract*—The boom for robotics technologies in recent years has also empowered a new generation of robotics software. The Robot Operating System (ROS) is one of the most popular frameworks for robotics researchers and makers which is moving towards commercial and industrial use. Security-wise however, ROS is vulnerable to attacks. It is rather easy to inject or eavesdrop data in a ROS application. This opens many different ways to attack a ROS application resulting in data loss, monetary damage or even physical injury. In this paper we present a secure communication channel enabling ROS-nodes to communicate with authenticity and confidentiality. We secure ROS on a peer-to-peer basis in the direct interaction between publishers and subscribers. We describe the implementation changes we have made to the ROS core and assess the overhead introduced by the new security functions.**

I. INTRODUCTION

Industrial networks will most probably become an even more interesting target for cyber attacks in future due to the increasing level of connectivity to outside services and sites. This exposes those networks to attack vectors which–due to air gaps–have not been feasible so far. It has already been shown that industrial systems are prone to cyber attacks [1], [2]. A well known example of an incident where an industrial control system was infected by malware is the Stuxnet worm [3]. Such attacks may be preceded by orchestrated malware infections designated to collect data [4], and/or exploit a lack of awareness on the user's side. While comprehensive standards exist to protect industrial control systems (e.g., as published by NIST [5]), security in industrial control remains a demanding issue. The attack on the Ukrainian power grid in December 2015 also shows that cyber attacks are becoming more sophisticated and are no longer just random acts but are prepared for several month before the actual attack [6].

Cyber attacks on industrial systems may not only be used to steal confidential information, they may also enable the manipulation of robots in their operation. Suppose that a hack on industrial robots causes them to place less than the minimal number of welding points on a car. If so, then the damage caused may range from shortened service intervals, up to lethal mass accidents. In such an attack, the target is not secret information, but causing maximal damage (to human life or at least the reputation of some enterprise).

The coming years will show an increased use of collaborative robot systems where robots are no longer operating behind fences or light curtains but are working hand in hand with humans. A vulnerable collaborative robot system poses a whole new threat because it can also be used to directly harm persons. In addition to industrial use-cases, more robots will be employed in service and home use in the coming years. There, security issues can have different but equally fatal effects ranging from privacy invasion to unsafe or harmful service robots. Typically, in such scenarios the awareness for network security is even lower than in industrial environments.

ROS [7], the Robot Operating System, is increasingly in focus of not only the research community but also steadily moves towards industrial application. The publish/subscribe [8] pattern which ROS implements is very useful to resolve the tight coupling and strong dependencies in a robotic system by providing various forms of decoupling and transparency. Exactly this transparency however, also introduces several security risks which are not sufficiently addressed in ROS [9]. Publishers cannot control the consumption of their data and subscribers cannot easily verify the source and integrity of the received information. This weakness may be used to inject malicious information or for eavesdropping on sensitive data. Network security using firewalls and virus scanners is not sufficient since after a breakin to the network, the application and the exchanged data is exposed to an attacker. Multi-layer security tries to secure a system on multiple levels to raise the hurdles for possible attackers in [10].

In this paper we present a secure communication channel for ROS which handles the communication between two nodes in a secure manner. This channel extends the existing TCP and UDP communication channels with certificate-based authentication, authorization as well as integrity and confidentiality. In addition, nodes which do not trust each other will not exchange data, thus effectively excluding malicious nodes from any communication.

II. RELATED WORK

Security issues in publish/subscribe systems has already been surveilled in [11] and [12]. The security in industrial networks and applications has been an active topic for quite some time now [1], [5], [10], [13]. Recently, an intrusion

978-1-5090-4623-2/17/$31.00 ©2017 IEEE

detection method based on artificial intelligence methods for SCADA systems using special support vector machines has been presented[14]. Shin et al. [15] study various approaches for intrusion detection in wireless industrial networks and propose improvements. In [16] the authors present a method to extend classical wired industrial networks with wireless components while still preserving safety and security. An interesting scheme that – like ours – uses challenge-response authentication to prove a robot's identity has been proposed in [17]. As with [18], cryptographic techniques must be used with care, and ad hoc "hand-crafted" solutions should in general be avoided. The work of [17] is nonetheless interesting in our context, as it clearly demonstrates the recognition of the authentication problem quite a while ago, whereas no solution on the level of the operating system seems to be available until today.

In our previous work we have already shown a concept for application-level security in ROS [19]. However, certain restrictions still apply with this approach e.g., that no publisher or subscriber can be prevented from joining the communication when security is not done within the ROS core. This still enables an attacker to perform node-specific denial of service attacks.

The recent machine-to-machine communication protocol Open Platform Communication–Unified Architecture (OPC-UA) has a security model which is also based on established cryptographic methods [20]. They use similar concepts like the ones described here. The scope of the security model however, is on centralized client-server communication (for the publish/subscribe pattern in OPC-UA no security model has been published at the time of writing).

The ROS community has already begun to address some security issues in the SROS[1] project. Here some similar concepts to what we present are used to secure the communication between nodes. However, it is currently only implemented partially for Python and not at all for C++-based nodes. This reduces the industrial applicability due to the missing code obscurity and the lower performance of Python-based code. In addition, SROS currently only secures TCP communication channels.

ROS2[2], the redesigned version of ROS which is currently under development, will build upon the Data Distribution System (DDS) [21] specified by OMG. DDS also includes security mechanisms which follow similar concepts as presented in this paper [22]. In ROS2 however, DDS is wrapped by a very thin (i.e., feature-lean) layer which does not necessarily allow the use of all DDS features. In addition, not all DDS implementations also include the DDS security features. The default transport implementation wrapped in ROS2 is Fast RTPS[3] which does not implement the whole DDS specification but only the RTPS transport layer [23]. This layer however, does not specify any security by itself (this is supposed to be added by DDS "on-top"). Further, ROS2 is not expected to be released soon and even after that it is not likely to supersede ROS as the dominating robot software framework at once.

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

[2]http://ros2.org

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

III. Architecture of a secure channel

In this section we present the architecture for a secure node-to-node communication channel for ROS and give indications of the implementation we performed. Our architecture relies on establishing trust between communication partners directly instead of letting the ROS master decide which node is allowed to join the application graph. This secures the communication graph on a peer-to-peer level.

A. Possible attacks on ROS-based applications

In this work we address several possible attack vectors on a ROS-application:

- Unauthorized injection of data
- Unauthorized access to data
- Denial of service (DoS) attacks on specific ROS nodes.

1) Unauthorized publishing: A node in 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. Also, false sensor data might be injected to the system e.g., to fake a normal system state after a manipulation or to provoke a certain reaction of the robot.

2) Unauthorized read 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 the node itself has no outgoing ROS communication.

3) Denial of Service: Denial of Service by publishing a large number of fake data can easily be done in ROS. 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. This can then be used for a targeted DoS attack on that node.

B. Requirements

With our secure communication channel we want to realize topic- and node-specific authentication, authorization, data integrity and confidentiality as well as increased availability by reducing the attack surface for DoS. Establishing trust to a specific node is done on a peer-to-peer basis between nodes for each topic separately instead of performing it only once between a node and the master. This design decision already excludes a range of attack vectors on a ROS application. In order to achieve this, we secure the TCP and UDP communication channels between nodes.

Nodes must have a way to verify that another node is trustworthy and authorized for joint communication for a certain topic. After ensuring this, the further communication needs to be confidential and requires data integrity to be assured.

The solution for peer-to-peer security must be transparent to existing nodes and thus, not require rewriting of existing ROS-based applications.

C. Secure channel design

The following description of 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. 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) [24] and Datagram Transport Layer Security (DTLS) [25] and perform fine-grained authorization on a per-topic basis. This is done by i) an initial handshake with mutual authentication and authorization based on certificates and public-key cryptography (RSA), ii) using symmetric encryption (AES-256) to secure the further communication and iii) use Message Authentication Codes (MACs) to ensure data integrity.

When communication between two nodes is initiated in ROS to subscribe to or publish a certain topic, first an XMLRPC call is performed from the subscriber to the publisher. The client node receives the connection information for this call from the ROS master. The XMLRPC call is a first handshake between the two nodes and is performed to exchange information regarding the future communication via TCP or UDP. The nodes verify that the requested topic or service name match and then exchange port information for the TCP or UDP connection. After that–in case of TCP-based communication–a TCP channel is established between those nodes. In the UDP case, the publisher node just transmits UDP packets whenever new data is available.

Note, that when consuming a service, no prior XMLRPC call is necessary and UDP-based communication is not an option.

To secure the communication between nodes, we use TLS for TCP and DTLS for UDP respectively. In both cases, an additional handshake has to be performed to establish a secure channel. For this, the server sends its certificate to the client which is checked for integrity by ensuring that the node name is correct and that the certificate has been signed by a trusted authority. The same operation is done with the client certificate at the server. A challenge-response operation is used to generate keys for future encrypted communication with integrity checks using a MAC.

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). 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. For this we need to modify the *roscpp* package. There, the implementation of TCPROS and UDPROS can be found. In the TCP case, on initially opening a connection to another node, a TLS handshake is performed before the ROS-specific communication. 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, also 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.

For UDP it is necessary to change the paradigm in which ROS performs the communication. The default (insecure) protocol in ROS just starts with sending a UDP package containing the topic data from the server to the client right after the XMLRPC request is finished. However, the ROSUDP communication is always unidirectional. In order to setup a DTLS connection, bidirectional data exchange is necessary to perform the DTLS handshake as described above. Thus, also 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 1 shows the changes in the communication flow between in the secure implementation for TCP and UDP. The new communication is denoted in green while the old messages are shown in red.

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.

IV. Key Management

The management of cryptographic keys and certificates is a crucial part of every security-enabled system. While it is not the main focus of this work we still want to point out how key management for our secure channel can be realized.

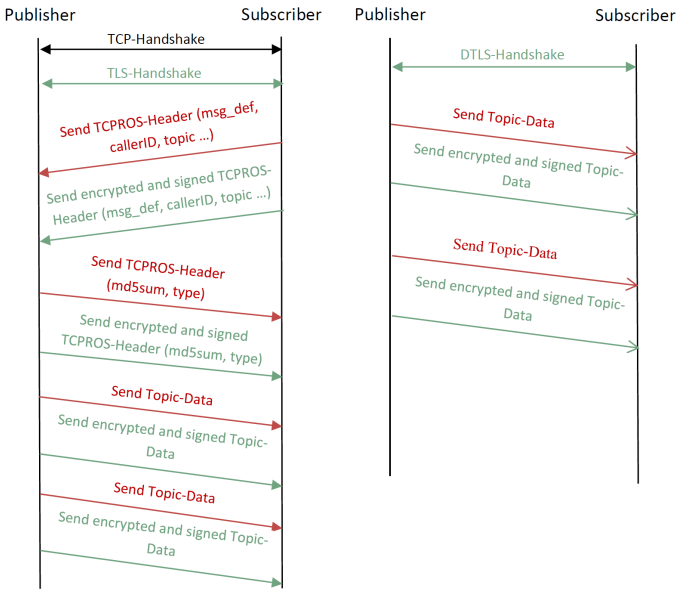

Fig. 1. A sequence diagram showing the changed communication pattern between two nodes. Green indicates the new, secured message flow, red indicates the old flow of communication.

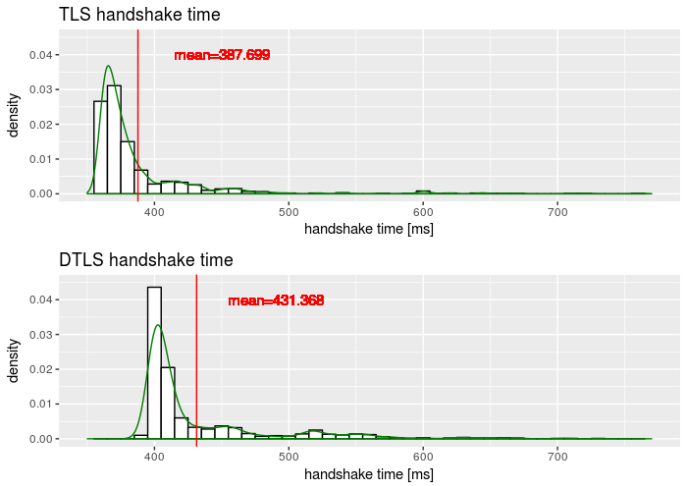

Fig. 2. A histogram of the time required to perform a TLS or DTLS handshake respectively. 1000 runs were measured for each protocol.

A. Device commissioning

We assume that each host device running a ROS node is equipped with a trusted platform module (TPM) which is initialized with a transport key (and the corresponding certificate) by its manufacturer. When a new host device is put into operation, the transport keys and hence the device is first validated using a challenge-response-protocol (to prove the possession of the private key) and the manufacturer's root certificate. After that, the transport keys are replaced with keys, which are only known to the operator of the application. This has to be done via a secure channel (authenticated and encrypted) between the device and the operator's technician. The application keys originate from the operator's own certificate authority. This step is necessary to ensure, that the device is no longer accessible using the manufacturer's keys. This also enables easier combination of hardware by different vendors.

The application keys for each device are at least composed of authentication, communication and key-encryption keys (for periodic updated of the employed keys). For individual functional groups, more keys might be deployed to this device e.g., for reconfiguration.

B. Operation

During operation, more key management actions might be required in case a device is broken and needs to be exchanged or in case of device maintenance.

A defective device must be de-registered and the replacement device has to proof that it is in fact the valid successor of the exchanged device. One way to achieve this is to provide a signed message to the replacement device, which holds the ID of the old device. After verification of the signature, the old device (respectively its ID) is de-registered (the new ID overwrites the old one) and the new device takes over. Note that de-registering without replacement can be achieved analogous, by simply setting the new ID to NULL or any suitable value.

Maintenance of devices can be performed by technicians which, have a tamper-proof key-card to authenticate at the device before it permits maintenance work. All actions should be recorded at the smart card to be able to later proof that no malicious actions have taken place.

In scenarios, where due to management overhead or cost a trusted platform module (TPM) is infeasible we suggest to use a dedicated authority similar as described in [19] which knows the authorized entities within an application and only transmits certificate information in cases where a valid identification has been performed.

V. Evaluation

The main goal of our evaluation is to show the overhead introduced by securing the communication channel in ROS. To do this we measure i) the overhead for the initial handshake, ii) the pure transportation overhead and iii) the overhead in a normal application. All experiments have been performed on a standard laptop running sporting an Intel core i7-4810MQ CPU running four cores with HyperThreading at 2.8GHz with Ubuntu 16.04 as operating system.

A. Handshake overhead

To establish the secure communication channel, a handshake has to be performed first to ensure authenticity and exchange encryption keys. This is only done once per node pair. We measure the time required to perform the handshake to assess the impact on application performance. A long handshake might cause a significant start-up delay for a ROS-based application. In our experiment we perform 1000 handshakes for TCP and UDP each.

The results are shown in figure 2. It can be seen that the majority of results are below 400ms for TCP and below 450ms

for the UDP handshake. Thus, the additional time required for the handshake is acceptably low in most scenarios.

B. Transport overhead

The transportation overhead is the additional CPU load generated by the encryption during message transfer. To assess the overhead on the cost of communication, we use an application which publishes dummy data without any prior processing. Thus, the measured load is merely only generated by communication and encryption. We use data structures of different sizes to also measure the impact of higher payloads.

The following test data structures are used to evaluate the overhead in communication.

1) A simple string transmitted over TCP and UDP
2) Transform messages[4] exchanged over TCP and UDP
3) PointCloud [5] messages exchanged over TCP and UPD

The application was tested with publishing frequencies of 100 and 1000 Hz to see the difference in overhead for higher data rates. Figure 3 shows the results of this evaluation. It can be seen that the communication overhead is in the range of 50 - 100%. Thus, the pure data transmission is more expensive which has to be expected for encryption. It has to be noted however, that the relative overhead in a real application is lower since there, the application itself generates more load (see next section). We also want to note, that the Botan library we used in these experiments was not compiled to use architecture-specific cryptographic instructions. Thus, this overhead can be reduced further by using such an optimization.

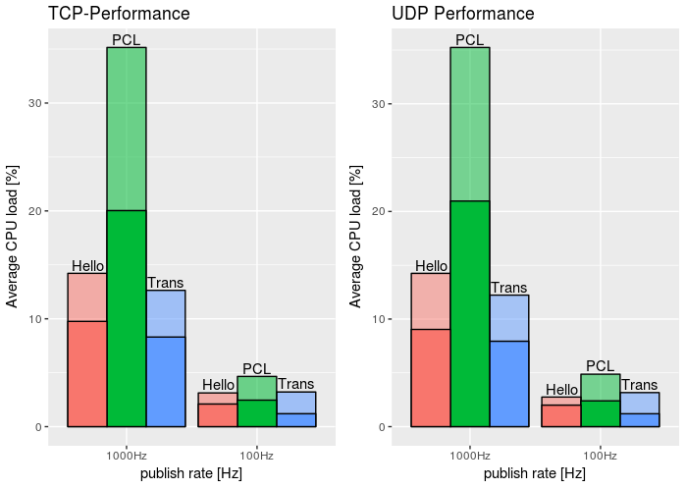

Fig. 3. The communication overhead for different data structures and different publishing frequencies. Each bar shows the processor load for the default (solid color) and added overhead by the secure ROS core (light color). Hello: simple string, PCL: point cloud message, Trans: Transform message.

In addition to load measurements, we also consider the change in payload size. The expected overhead consists of three parts. First, the communication protocol for ROS had to be modified to deal with variable lengths of data, thus 4 bytes for a length value were added. Second, the signature to ensure message integrity is added to the communication. Third, the padding to the AES block size of 128 bits is a variable part of the data overhead. Since the symmetric encryption is block-based, shorter data chunks are padded to the block size which typically results in increased message size. Depending on the original payload size, up to 15 bytes may be added to reach AES block size.

We performed experiments with message sizes from 1 byte to 1000 bytes. The result is an overhead of 58 to 73 bytes where 58 bytes means no padding to the AES block length and 73 bytes include a padding of 15 bytes. Thus, the overhead in data is in an acceptable range especially for larger message sizes. For very small messages, the overhead is indeed significant but it can hardly be reduced without compromises in security.

C. Application overhead

As a final experiment we use a normal application to test the overhead introduced by the secure channel. The application performs arithmetic operations and publishes the results at 100Hz. This experiment evaluates the overhead introduced by the secure channel in relation to the load a normal application generates thus presenting a more realistic scenario than measuring just the transportation overhead. Figure 4 shows then results. As it can be seen, the overhead introducted by the secure channel is only a small fraction of the load the application itself generates. Thus, in realworld applications, the extra ”cost” of security is in the range of a few percent of CPU load.

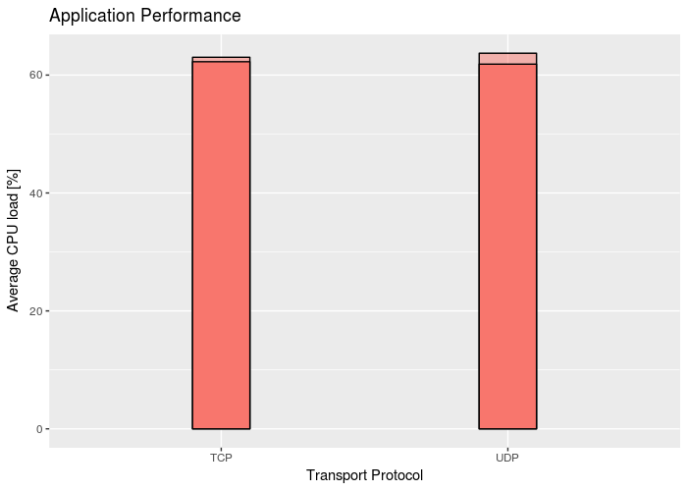

Fig. 4. The application overhead for TCP and UDP.

VI. Discussion and Conclusion

In this work we have presented a concept to secure ROS communication channels using cryptographic methods. The trust in the system is established between nodes on a peer-to-peer topic-specific basis. TLS and DTLS have been included in the ROS core to secure the communication. This required

[4]http://docs.ros.org/kinetic/api/geometry_msgs/html/msg/Transform.html
[5]http://docs.ros.org/kinetic/api/sensor_msgs/html/msg/PointCloud.html

adding an additional handshake step for TCP and a modified communication paradigm for the UDP channel to enable a DTLS handshake to take place. Our solution provides fine-grained control over permissions to publish, subscribe or consume data.

One complex point in each public key infrastructure is handling certificates. This is no different in our approach. As discussed in section IV, proper key management is a complex topic which requires special consideration.

In addition, we do not secure the ROS master itself in our approach. Thus, the master would still transmit information about nodes and topics to a malicious node. This however can also be circumvented with a secure channel and a certificate for the master or using a similar approach to our previous work [19].

Our approach reduces the attack surface for DoS in ROS. However, it has to be said that it is still possible to directly shutdown a node using a special XMLRPC call to that node. Thus, the management layer of ROS also needs special attention in future.

In comparison to using a virtual private network we also authorize nodes for receiving and producing data on a per-topic basis. Thus we can exclude nodes which are not meant to be part of a certain data stream effectively excluding malicious nodes from injecting and eavesdropping data. In contrast to our previous approach [19] we now also encrypt the ROS data structures for messages which makes frequency analyses of e.g., control commands impossible. In addition, it is not necessary in our new approach to recompile nodes in order to benefit from the secure channel.

We have provided our concept and source code to the Open Source Robotics Foundation (OSRF) for consideration to be included in the SROS project.

In future work we will also address the security of ROS2 as well as other industrial and robotic systems. In addition, we will improve our key management techniques and work on securing the ROS management layer.

REFERENCES

[1] M. Cheminod, L. Durante, and A. Valenzano, "Review of security issues in industrial networks," *Industrial Informatics, IEEE Transactions on*, vol. 9, no. 1, pp. 277–293, Feb 2013.

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

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

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

[5] K. Stouffer, V. Pillitteri, S. Lightman, M. Abrams, and A. Hahn, "Guide to industrial control systems (ics) security," National Institute of Standards and Technology, Tech. Rep., 2015, NIST Special Publication 800-82, Revision 2.

[6] 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*, vol. 53, no. 5, pp. 11–13, May 2016.

[7] M. Quigley, K. Conley, B. Gerkey, J. Faust, T. Foote, J. Leibs, R. Wheeler, and A. Y. Ng, "Ros: an open-source robot operating system," in *ICRA workshop on open source software*, vol. 3, no. 3.2, 2009, p. 5.

[8] P. T. Eugster, P. A. Felber, R. Guerraoui, and A.-M. Kermarrec, "The many faces of publish/subscribe," *ACM Comput. Surv.*, vol. 35, no. 2, pp. 114–131, Jun. 2003. [Online]. Available: http://doi.acm.org/10.1145/857076.857078

[9] J. McClean, C. Stull, C. Farrar, and D. Mascareas, "A preliminary cyber-physical security assessment of the robot operating system (ros)," in *Proc. SPIE*, vol. 8741, 2013, pp. 874 110–874 110–8. [Online]. Available: http://dx.doi.org/10.1117/12.2016189

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

[11] C. Wang, A. Carzaniga, D. Evans, and 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*, Jan 2002, pp. 3940–3947.

[12] C. Esposito and M. Ciampi, "On security in publish/subscribe services: A survey," *IEEE Communications Surveys Tutorials*, vol. 17, no. 2, pp. 966–997, 2015.

[13] D. Dzung, M. Naedele, T. von Hoff, and M. Crevatin, "Security for industrial communication systems," *Proceedings of the IEEE*, vol. 93, no. 6, pp. 1152–1177, June 2005.

[14] L. Maglaras and J. Jiang, "Intrusion detection in scada systems using machine learning techniques," in *Science and Information Conference (SAI), 2014*, Aug 2014, pp. 626–631.

[15] S. Shin, T. Kwon, G.-Y. Jo, Y. Park, and H. Rhy, "An experimental study of hierarchical intrusion detection for wireless industrial sensor networks," *Industrial Informatics, IEEE Transactions on*, vol. 6, no. 4, pp. 744–757, Nov 2010.

[16] J. Åkerberg, M. Gidlund, T. Lennvall, J. Neander, and M. Björkman, "Efficient integration of secure and safety critical industrial wireless sensor networks," *EURASIP Journal on Wireless Communications and Networking*, vol. 2011, no. 1, pp. 1–13, 2011. [Online]. Available: http://dx.doi.org/10.1186/1687-1499-2011-100

[17] W. Adi, "Mechatronic security and robot authentication," in *Bio-inspired Learning and Intelligent Systems for Security, 2009. BLISS '09. Symposium on*, Aug 2009, pp. 77–82.

[18] R. Toris, C. Shue, and 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)*, April 2014, pp. 1–6.

[19] B. Dieber, S. Kacianka, S. Rass, and P. Schartner, "Application-level security for ROS-based applications," in *Proceedings of the 2016 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS 2016)*, 2016.

[20] *Unified Architecture Part 2: Security Model*, OPC Foundation, 2015, version 1.03.

[21] *Data Distribution Service, v1.4*, Object Management Group, 2014, version 1.4.

[22] *DDS Security Specification*, Object Management Group, 2016, version 1.0 Beta2.

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

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

[25] E. Rescorla and N. Modadugu, "Datagram Transport Layer Security Version 1.2," RFC 6347 (Proposed Standard), Internet Engineering Task Force, Aug. 2012, updated by RFCs 7507, 7905. [Online]. Available: http://www.ietf.org/rfc/rfc6347.txt

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.