Enhancing security in ROS

Gianluca Caiazza[1], Ruffin White[2] and Agostino Cortesi[1]

[1] Ca' Foscari University, Venice, Italy
[2] UC San Diego, San Diego, USA

Abstract. In recent years, we observed a growth of cybersecurity threats, especially due to the ubiquitous of connected and autonomous devices commonly defined as Internet of Things (IoT). These devices, designed to handle basic operations, commonly lacks security measurements. In this paper we want to tackle how we could, by design, apply static and dynamic security solutions for those devices and define security measurements without degrading overall the performance.

Keywords: Security, IoT, x.509 Certificate, Encryption

1 Introduction

With the spread of connected and smart devices we observed a tremendous grown on the amount of personal data that are stored and processed every day. Considering the sensitive nature of this information there's a widespread suspicion concerning the way in which these information flows into the infrastructures. Additionally, with the increase of smart cities and connected environments this critically is going to enlarge. In particular, in these environments we can identify two possible group of threats: physical and logical. In the first group we found simple physical attack as shooting down the device or capture it with a net; as well as more complex one as radio sniffing, tampering, dossing, etc. In the authors opinion this kind of vulnerability should be addressed from the hardware/firmware point of view, since it will result in a waste of effort tackle them from the application-level. Indeed, our focus is on the logical approach of the problem, on the data-centric analysis of the infrastructure. So, in the light of this, its easy to understand the importance of supporting the key property of computer security: confidentiality, integrity, authenticity, non-reputation, availability. By enforcing these simple concepts, we are able to develop countermeasures against literature attacks as: eavesdropping, modification, impersonation, repetition. However, since we are working with IoT devices the way in which these properties are enforced is not trivial. In fact, we need to consider the intrinsic limitations of the devices, either from the power consumption point of view and the actual computational power available. Additionally, since we want to design a solution that could be applied to a wider scope of devices its safe to assume that we want to keep as real-time performance as possible.

2 Security Enhancements

In order to develop our solution we worked on a widespread open-source middleware software for robotic implementation: Robot Operation System (ROS) [1]. It provides the services of an operating system, including: hardware abstraction, low-level device control, implementation of commonly-used functionality, message-passing between processes, and package management. Still, from our point of view, its important to notice that it doesn't implement any security measurements at all.

Intuitively, the easiest way to ensure secrecy between two (or more) agents is by encryption. Therefore, by means of TLS/SSL and the usage of x.509 certificate we can easily enforce confidentiality and authenticity. In the particular case of ROS, this improvements has been implemented with SROS [2][19] a set of security enhancements that aims to secure ROS. By leveraging on SROS, our goal is to extend the new security features and improve them with some fine statical mechanisms: (1) define an exhaustive standard for security logging the API operations; (2) define a new profile syntax standard for the definition of policy file; (3) provide new method for the automatic generation of the aforementioned certificate. Still, been aware of the limited resources on the devices we need to carefully evaluate each choice in order to keep the solution as lightweight as possible. Along this line, we design our enhancements as static offline mechanisms which results are simply applied to the devices. In detail, apart from the logging mechanisms - that adds a negligible overhead on the device - the other solutions aims to granularly define access control for the agents. In addition, we discuss about two different approaches for the definition of the relations between the certificates and policy profiles.

Our goal is to keep the resource usage under control. In detail, by leveraging on the statical approach defined by SROS (that embeds the access control policy as extensions of the x.509 certificate), and by exploiting Park *et al.* work on x.509 extensions [3][4], we propose two different architectures for IoT network as well: user-pull and server-pull.

By combining our proposed static improvements with the usage of smart certificates we can easily enhance the way in which users defines how agents communicate in the network.

Contrary to all the related works, to the best of our knowledge this is the first research that focuses on the automatic definition of embedded policy profiles in a trusted network and actively prevents, at application level, the disclosure of sensitive information and blocks unauthorized agents by applying a-priori access control model in addition to library functions security enhancements.

3 Technical Overview

In order to better understand how we develop the proposed solution, it's important to have a grasp of the framework that has been selected. In this section we will briefly evaluate ROS and the general concept behind SROS.

3.1 Robot Operating System

ROS implements a peer-to-peer network, namely 'graph', in which the processes (agents) can communicate at runtime via publish/subscribe [5] pattern. From our point of view, the graph itself is the key concept that we need to exploit. The basic Computation Graph components of ROS are nodes, Master, parameter server, messages, services, and topics, all of which provide data to the graph in different ways. Let's see in detail each component:

- Nodes: Nodes are the basic processes that perform computation. ROS is designed to be modular at a fine-grained scale; usually a robot control system comprises many nodes. For example, one node controls a camera, one node controls the temperature sensors and so on.
- Master: The ROS Master provides name registration and lookup to the rest of the Computation Graph. In practice, the master is a DNS server for nodes. Without the Master, nodes would not be able to find each other, exchange messages, or invoke services.
- Parameter Server: The Parameter Server allows data to be stored by key in a central location. Even thou we need to consider this as an independent component, currently it's part of the *Master*.
- Messages: Nodes communicate with each other by means of messages. A message is simply a data structure, comprising typed fields.
- Topics: Messages are routed via a transport system with publish/subscribe mechanism. A node sends out a message by publishing it to a given topic. The topic is a name that is used to identify the content of the message. A node that is interested in a certain kind of data will subscribe to the appropriate topic. It's important to notice that there may be multiple concurrent publishers and subscribers for a single topic, and a single node may publish and/or subscribe to multiple topics. Furthermore, publishers and subscribers are not aware of each other's existence.
- Services: The topics publish/subscribe model is a very flexible communication paradigm, but its `many-to-many`, for *one-way* transport it's more appropriate a request/reply interactions. This kind of communications are done via services, which are defined by a pair of message structures: one for the request and one for the reply. A providing node offers a service under a name and a client uses the service by sending the request message and awaiting the reply.

That said, we can easily translate the ROS's structure into a more general IoT network, in which the nodes represents the single device and the master the default gateway; indeed, this is the structure that is currently in the market (e.g. Apple HomeKit, Samsung Smart-home, Google Things Solutions, etc).

3.2 SROS: Secure ROS

As mention, SROS is a set of security enhancements for ROS that aims to secure ROS API and ecosystem by means of native TLS/SSL support for all IP/socket level communication. In addition, with the usage of x.509 certificates, they defined chains of trust by means of a certificate authority (namely keyserver), namespace node restrictions and permitted roles, as well as user-space tooling to auto generate node key pairs, audit ROS networks, and construct/train access control policies. Furthermore, they defined AppArmor profile library templates, that allow users to harden or quarantine ROS based processes running at linux OS kernel level.

That said, we can summarize that SROS is intended to secure ROS across three main fronts:

1. Transport Encryption: with the usage of TLS and x.509 PKI for authenticity and integrity
2. Access Control: restrict node's scope of access within the ROS graph to only what is necessary leveraging on definable namespace globbing
3. Process Profiles: restrict the application (file, device, signal, and networking access) thanks to AppArmor profile component library for ROS

4 Access Control Policy Generation

The access control policy profile should not be confused with that of SROS's AppArmor profile library or the profiles they provide using features from Linux security modules. Those help users provide Mandatory Accesses Control (MAC) for ROS nodes on the runtime processes level of the hosting operating system. The policy profiles we are discussing below are the ones related to enabling access control for ROS nodes in the ROS graph network level.

Generally speaking, in these profiles we store the communication topology between agents, specifying the operation that are allowed or denied to the device. In order to build a proper profile, we can proceed in two different ways: manually defined the rules or automatically extract them from meaningful log.

As representative of the second group, we have the aforementioned `AppArmor`, a proactively software layer that protects the operating system and applications from external or internal threats, by enforcing good behaviour and preventing even unknown application flaws from being exploited. In detail, AppArmor defines a set of rules for the selected application based on the operation that has been *exercised* by the user during a training phase.

4.1 Security logging

Along the same line, we want to implement a similar mechanism also in our solution. First of all, we need to specify a suitable way of acquiring security logs. In order to do so, we need to identify the key components and communication mechanisms (e.g. API) that are used in the application. The new logging system that we are going to define, will be more structured and informative in regards of the required operation and resource access.

Therefore, leveraging on the well known unix logging system, we specify three different execution mode: *audit*, *complain* and *enforce*. In the first mode, we log all the operation that are executed by the application without applying any constraint. Once the user have exercised the application as necessary, he can move to complain mode. In this case, we prompt to the log all the operations that violates the rules that has been defined in the previous mode. This is a crucial phase, in which we can verify if the rules that we previously define works as expecting or are too strict or naive with respect of the desired access control. Then, if the defined profile works as wanted, it will be enough to enforce them by applying the namesake mode.

All in all, in this phase, is important to enhance the behavior of the application's APIs such that we can easily log the kind of resource that the agent want to access. In the case of ROS, we have extract three macro-group of resources: topic, service and slave operation. In detail, in topic we found the APIs that allow a node to register himself as a publisher or as a subscriber. Along the same line, we have the service APIs, that operates likewise the topic once; lastly we have all the operation that are executed by simple slave node. In general, we can make a parallelism with this API and the hierarchy in an IoT network such that slave operations define the IoT device API, whilst topic and service represent the communication mechanisms (gateway API).

4.2 Policy Profile Syntax

As previously said, we took a good deal from the AppArmor policy definition such as it's globbing syntax. Our goal is to define a more applicable syntax intended to encode policy rules, definition, and relationships in our trusted network. Our intuition, is to leverage upon the namespace resource organization a good deal to define the profiles.

In ROS, we define a domain as a simple root '\'. In it, we address a node simply recalling its position in the hierarchy. When nodes are integrated into a larger system, they can be pushed down into a suitable namespace that defines their functionality. For example, one could take two robot namely `foo` and `bar` and merge them into the same domain with 'foo' and 'bar' subgraphs. Therefore, if both devices had a node named '*camera*', they would not conflict since they will be addressed respectively as: \foo\camera and \bar\camera.

It's important to notice that ROS supports several methods to address a resource that could be either: *base, relative, global* and *private*; although, for the sake of our policy profile we always extend them in a plain version that explicit all the chain relation of the node. In this way, we can define policy profile in an agnostic way in regards of the underling implementation.

Since we are able to unambiguously address an agent, we can specify the necessary rules for each one. We can suppose that several devices share a set of common rules, either because they are necessary for a feature (e.g. logging), or because the node has a *role* in the application, i.e. is an administrative device. In order to simplify the management of these roles, we introduce the concept of `include`. As for other programming languages, the goal of using include statement is to add - at compile time - a set of predefined piece of code, in this case a collection of rules for the specific device. These rules simply follow the same structure of the one that are defined in the policy profile.

Intuitively, we can sort out different resource twofold, by explicitly define the kind of resource (i.e. topic, service, parameter) and by defining resource specific masks for permissions. Let see below an example of node profile:

```
/namespace
{
#include role
resource /scope masks
}
```

As mention before, one of the advantages of using this kind of notation for addressing resource and agent is the usage of globbing syntax. In detail, it's possible to define regexp formulas in rules and profiles scope. Therefore, if we want to specify that an agent is allowed of interact with all the first level camera of the other agents, it will be enough to specify the rule: **\camera.

As a rule of thumb, we define regular expressions with the following syntax:

- * : represent any number and any characters in the current namespace
- ** : represent any number and any characters including the definition of sub-namespaces
- ? : represent a single character or number
- [abc] : represent the the single character `a`, `b`, or `c`
- [a-c] : represent all the character in between `a` and `c`
- {ab,cd} : expands the string to match the expressions `ab` and `cd`

Additionally we introduce the possibility of specifying `deny` rules. In fact, if we want to single out a resource from a wider regexp, it will be enough to define a specific rules for the resource (or resources via regexp) as showed below:

```
....
deny resource /bar/foo1 masks,
resource /bar/foo* masks,
....
```

5 X.509 Certificate: Distribution Architecture

In this section we evaluate different approaches for the definition of the relations between the certificates and policy profiles. We discuss about two different architectures: user-pull and server-pull.

As the name suggests, these models are respectively user-based and host-based. In particular, the purpose of these architecture is to discuss about the static and dynamic solution either for certificate distribution and attribute verification.

5.1 User-pull Architecture

As the name suggests, in this mode the user pulls the attributes certificate bundle from the server and stores it locally. Then it uses the certificate for the authentication phase with the other agents in the graph. This family of solutions exploits the problem of the authentication leveraging on the integrity services offered by the certificates by design. In fact, the certificates are issued by the Certificate Authorities (CA) which are `trusted` entities in the system. However, in addition to the trivial implementation, in which we store the profiles as extensions of the x.509 certificate, we want to introduce additional types of certificate that are defined in the x.509 standard: *Identity certificates* (ID) and *Attributes certificates* (AC)[6].

While X.509 public-key certificates bind an identity and a public key, an attributes certificate (AC) doesn't contain a public key but may contain attributes that specify group membership, role, security clearance, or other authorization information associated with the AC holder. The reason why we should use AC for authentication information in place of PKC extension is manly for two reasons: authorization often does not have the same lifetime as the identity and the public key. So, when we store authorization information in a PKC extension, the general result is that we are shortening the PKC useful lifetime. Furthermore, the CA that issues the PKC is not usually authoritative for the authorization information. In fact, it represents a threat for the system that should be avoided. There are several ways in which we can bound the authorization with the identity certificate, below we presents three different approaches: *monolithic, autonomic* and *chained signatures.*

Monolithic

This is the easiest solution in which we consider only one certificate authority in charge of both identity certificate and attribute. In the monolithic approach we create a certificate that holds both the identity and attributes information; trivially, this is implemented by using x.509 and the extension fields. The resulting certificate *tightly coupled* the identity information and the attributes with a single signature. This means that in order to change an information in an existing certificate we need to revoke the previous and issue a new updated one. However, the management of this particular solution is simplified in comparison of the other, since we need to trust only one CA. Considering that all the certificate information are verified by the only CA's signature and therefore there is only one CRL that needs to be checked.

However, as previously said, this approach has several drawbacks. First, multiple CA's are not supported. We can't revoke a certificate if we aren't the issuer CA and there is the possibility of issuing multiple certificate with different attributes for the same agent. Secondly, due to the design of the solution we are not able to maintain different life-time for multiple attributes; in fact, all the attributes share the lifetime of the PKC. All in all, monolithic approach is the most favourable solution when we are limited in terms of resources or we are looking for a statical solution, even though we sacrifice flexibility in maintenance.

Autonomic

In this case we introduce the concepts of multiple CAs and we differentiate between identity and attribute certificates. With this approach we want to define a *loosely coupled* binding between the ID certificate and the AC. This particular solution allows the existence of multiple ID certificates per agent provided that there is a injective function from the certificates to the agent; this means that we will never have more than one agent that corresponds to an ID certificate. As such we can bind each AC with a different set of information from the ID as: subject's name, public-key, certificate serial number, etc.

Depending on the chosen set of information that has been selected we can modify the certificate issuing a new one and still maintain the correlation between the AC and ID as long as the binder information has not been changed. As example, if we insert in the ID certificate the unique serial number of the agent and we bound the attribute certificate based on that, we can change the other information such as lifetime, serial number, subject's name, while the link between the certificates holds. However, since we moved from a static solution to a dynamic one, we should be extra careful about the new threats as the choice of the information set. In fact, even though we have an injective function from the certificates to the agent there aren't constraint on the information that are stored in the certificates. It means that if we accidentally choose a common set of information, it may happen that the same attribute certificate can be used by unauthorized agents that share the set of information with the authorized agent.

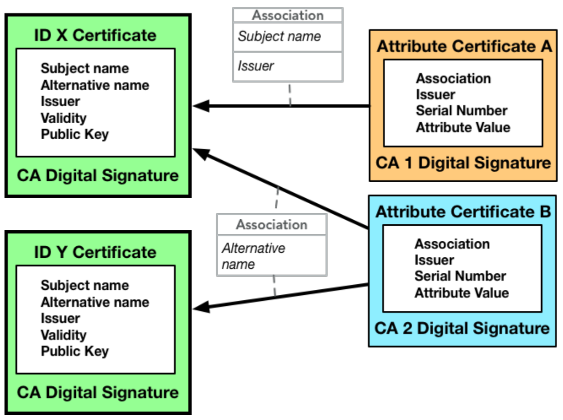

Fig. 1. Autonomic certificate

Chained-Signature

With this technique we want to take the security features guaranteed by the monolithic solution and part of the flexibility of the autonomic approach and create a new hybrid solution. As in autonomic an agent can have multiple ID certificates issued by multiple CAs. However, instead of binding the attribute certificate with an arbitrary set of information, we bound on the digital signature of the corresponding ID certificate. In fact, if the information in the referenced ID certificate are changed (a new certificate is issue), the digital signature *should* change as well. Under the assumption that we are using a suitable signature algorithm, when we issue a new ID certificate (with a different signature) the link between the two certificates is broken and then the attribute certificate becomes automatically useless. One of the advantage of chained signatures is that we don't need to aggregate all the attributes according to the shortest lifetime of the certificate as in monolithic. Furthermore, we introduce a mechanism that allows us to share with other agents only the necessarily information. In fact, with a monolithic certificate all the policy of the agent are available in the PKC, instead with this solution we can share only the necessarily attribute certificate enforcing a new privacy feature and dynamic management of profiles.

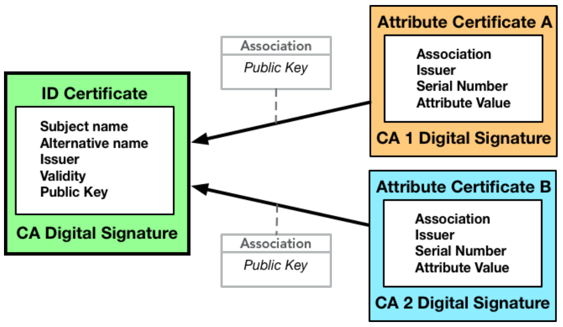

Fig. 2. Chained certificate

5.2 Server-pull Architecture

In server pull model the general idea is to demand the authentication phase to the attribute authority (AA). Our goal is to define a dynamic solution in which apart from the AA no one needs to know the attribute information. However, as in user-pull model, we still need a method to unambiguously identify an agent in the graph; we can achieve this straightforwardly with the usage of the already defined ID certificates.

But instead of issuing attribute certificates and bound them to the ID, we store the attribute policy roles locally in the attribute authority. This particular solution allows us to implement in the AA whenever access control we want (e.g. MAC, DAC, MLS, MCS, RBAC), without modifications to the client (agents) APIs. We design a high level API that permits the agents to retrieve the authorization response from the AA regardless of the chosen access control method.

There are several advantages on this model: first of all, thanks to the usage of AA instead of static certificate we can achieve a dynamic flexible solution that can evolve and change during run-time without additional setup. Secondly, we can maintain the secrecy of the sensitive information about the policy topology inside the attribute authority without compromising the network topology in critical applications.

Still, the solution introduces the problem of the *single point of failure* (SPOF). In practice, considering that all the agents need to query the attribute authority in order to receive a response about the permissions, if it fails the entire system will stop working. In fact, from the attacker point of view it will be enough to tamper the AA to compromise the entire system. In addition, we introduce by design an overhead in the handshaking process; in fact, what was a straightforward local check of the profile in user model, became a remote request to the external attribute authority. However, this problem can be slightly mitigated by resuming previous sessions via `Session Ticket` as introduced in TLS 1.3 [8].

6 Related Work

To the best of our knowledge, the present work is the first research focusing on the discussion of a set of tools and techniques for the automatic definition of security policy profiles in a robotic framework. Although several security threats analysis in the industrial robotic applications have been performed recently, they mainly focus on the threats deriving from the modification of network topology from local to remote connection [9][11][12].

Akerberg *el al.* [10] tackled the problem from the communication channel point of view, proposing a security communication framework for the integration in classical wired industrial networks of wireless nodes, with the implementation of end-to-end integrity and authentication measures by utilizing the black channel concept. On the other hand, Wang *et al.* [13], analyse Publish-Subscribe communication paradigm in a wide-area network. In this particular setup services handle information across distinct authoritative domains and need to manage a large population of publishers and subscribers. In detail, they discuss about the security issues and requirements that arise, distinguishing among those requirements that can be achieved with current technology and those that require novel solutions.

Similarly to the proposed solution, Dieber *et al.* [15] proposed a security architecture intended for use on top of ROS on the application level. By means of an authorization server and the usage of x.509 certificates for authentication, they ensure that only valid registered nodes are authorized to execute some operations. However, their statical architecture is based on the assumption that we have manually generated and distributed the certificates and registered the list of nodes in the authorization server; moreover, they delegate to the user the distribution of the lists of certificates serial number of the nodes that are authorized to query each other.

Additionally, Lera *et al.* [16], presented an interesting analysis on which they proposed that ROS communications should be encrypted. Differently to the previous discussions, in this document they directly evaluate the performance of ROS under encryption. In detail, they used 3DES cyphering algorithm and evaluated the performance both from the computing and the communications point of view. These works are particularly useful in evaluating the performance of ROS under encryption, although it's important to notice that we should prefer algorithms that have dedicated instructions added in hardware in modern CPUs, that help them run substantially faster than software only ciphers implementation. So, depending on the chosen algorithm and key-length we can easily observe some changes in the performance [17][18].

7 Conclusion

We have presented a static procedure to generate and amend policy profiles for IoT robotic devices. In order to determine a suitable solution we analyzed the ROS framework and we discussed about the design of an agnostic solution for the definition and application of an access control policy profile. In the first part, we discussed about the definition of a new standard for security logs, that allows us to reconstruct all the library functions call-back that are not covered by the 'basic' log system, by means of a more accurate and defined structure. Then, we proposed a standard syntax for the definition of the policy profiles based on the well-known AppArmor syntax. Finally, we discussed about static and dynamic solution for certificate and attribute distribution, thus contributing to the scenario depicted in [20]. From our analysis it emerges that one of the biggest threats of robotic network is the lacking of security measure in the communication mechanisms, that needs to be harden with the introduction of access control and encryption mechanisms.

Still, there are a number of issues that are part of our plans for future work as: mitigate the disclosure of sensitive information (i.e an agent profile), improve privacy in access to partially unauthorized resources (e.g. function output custom sanitization); as well as decoupling the cryptographic operations from the authenticity mechanisms by means of middleware implementations as oneM2M [14] or other DDS.

Moreover, we aim to conduct a further analysis in the definition of a real-time system with the addition of cryptographic mechanisms via 'Real-Time Publish Subscribe' (RTPS) protocol for mission-critical implementations.

All in all, we believe that we have given a solid base for the definition of the future security mechanisms for robotic devices that could be easily and securely integrated in big-scale deployments without suffering software limitations. Furthermore, in our opinion the definition of high level solutions as the one that has been proposed in this paper is critical for spread security solutions by reducing the tradeoff between security and usability.

Acknowledgments

Work partially supported by CINI Cybersecurity National Laboratory within the project FilieraSicura, and by the Executive Program 2017-2019 Italia-India within the project "Formal Specification for Secured Software Systems".

References

1. M. Quigley, B. Gerkey, K. Conley, J. Faust, T. Foote, J. Leibs, E. Berger, R. Wheeler, A. Ng: ROS: an open-source Robot Operating System ICRA Workshop on Open Source Software, 2009
2. M. Quigley, R. White, H. I. Christensen: SROS - Securing ROS over the wire, in the graph, and through the kernel, ROSCon 2016
3. Joon S. Park, Ravi Sandhu: Smart Certificates Extending X.509 for Secure Attribute Services on the Web Proceedings of the of 22nd National Information Systems Security Conference (NISSC), 1999, pp. 337- 348
4. Joon S. Park, Ravi Sandhu: Binding Identities and Attributes Using Digitally Signed Certificates, Proceeding ACSAC 00 Proceedings of the 16th Annual Computer Security Applications Conference, Page 120, 2000
5. P. T. Eugster, P. A. Felber, R. Guerraoui, A. M. Kermarrec: The many faces of publish/subscribe, Journal, ACM Computing Surveys (CSUR), Volume 35 Issue 2,Pages 114- 13, 2003
6. S. Farell, R. Housley, S. Turner: "An Internet Attribute Certificate Profile for Authorization, Internet Engineering Task Force (IETF), 2010
7. A. Lenstra , X. Wang, B. de Weger - "Colliding X.509 Certificates", Report EPFL, 2005
8. The Transport Layer Security (TLS) Protocol Version 1.3 resource online: *https://tools.ietf.org/html/draft-ietf-tls-tls13-18*
9. M. Cheminod, L. Durante, A. Valenzano: "Review of security issues in industrial networks", IEEE Transactions on Industrial Informatics, Volume: 9, Issue 1, 2013
10. J. Akerberg, M. Gidlund, T. Lennvall, J. Neander, M. Bjorkman: Efficient integration of secure and safety critical industrial wireless sensor networks, EURASIP Journal on Wireless Communications and Networking, no. 1, pp. 113, 2011.
11. E. Byres, P. E. Dr, D. Hoffman The myths and facts behind cyber security risks for industrial control systems, In Proc. of VDE Kongress, 2004.
12. D. Dzung, M. Naedele, T. von Hoff, M. Crevatin: Security for industrial communication systems, Proceedings of the IEEE, vol. 93, no. 6, pp. 11521177, 2005.
13. C. Wang, A. Carzaniga, D. Evans, A. Wolf: Security issues and require- ments for internet-scale publish-subscribe systems, System Sciences, 2002. HICSS. Proceedings of the 35th Annual Hawaii International Conference on, 2002, pp. 39403947.
14. S. K. Datta, R. P. F. da Costa, C. Bonnet, J. Harri: "oneM2M Architecture Based IoT Framework for Mobile Crowd Sensing in Smart Cities " Networks and Communications (EuCNC), 2016.
15. B. Dieber, S. Kacianka, S. Rass, P. Schartner: "Application-level Security for ROS-based Applications", In Proc. 2016 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS), 2016.
16. F. J. R. Lera, J. Balsa, F. Casado, C. Fernandez, F. M. Rico, V. Matellan: "Cybersecurity in Autonomous Systems: Evaluating the performance of harden- ing ROS" XVII Workshop en Agentes Fsicos, 2016.

17. G. Singh, Supriya : "A Study of Encryption Algorithms (RSA, DES, 3DES and AES) for Information Security" International Journal of Computer Applications (0975 - 8887) Volume 67 No.19, 2013.
18. D. Giry : Bluecrypt cryptographic key length recommendation, resource online: http://www.keylength.com/, October 2016.
19. R. White, G. Caiazza, H. Christensen and A. Cortesi, "SROS1: Using and Developing Secure ROS1 System", in Robot Operating System (ROS): The Complete Reference (Volume 3), Springer, to appear, 2018.
20. A. Cortesi, P. Ferrara and N. Chaki, "Static analysis techniques for robotics software verification". ISR 2013: 1-6.