Cybersecurity in Autonomous Systems: Evaluating the performance of hardened ROS

David Mañanes, Francisco Javier Rodríguez Lera, Jesús Balsa and Vicente Matellán,

***Abstract*—As robotic systems spread, cybersecurity emerges as major concern. Currently most research autonomous systems are built using the ROS framework, along with other commercial software. ROS is a distributed framework where nodes publish information that other nodes consume. This model simplifies data communication but poses a major threat because a malicious process could easily interfere the communications, read private messages or even supersede nodes. In this paper we propose that ROS communications should be ciphered. We also measure how this ciphering affects its performance. We have used three different ciphering techniques: DES, AES and RSA. We have evaluated the performance of the system, both from the computing and the communications points of view. Preliminary results show that symmetric ciphers using private keys impose significant delays.**

***Index Terms*—autonomous systems, cybersecurity, robotics, performance, cyber-physical systems, ciphers**

I. Introduction

IT is possible to find driverless cars in the streets, autonomous vacuum cleaners in our homes, museum guides, hotel assistants, etc. These cyber-physical systems, as any computer-based system, can suffer different types of attacks, and the need of cybersecurity [7] is required.

ROS (Robotic Operating System) [8] has become the most popular framework for developing robotic applications. It started in the research environment, but currently most of manufacturers of commercial platforms use ROS as the *de facto* standard for building robotic software. For example, object-manipulation robots like Baxter (by Rethink robotics) [2] or service robots as our RB1 (by Robotnik) are ROS based platforms.

Our research group is developing an assistant robot [5] for the elderly. When we initiated experiments involving potential users, caregivers have asked us about the security of our robot and about the privacy of its communications [1]. When an assistant robot carrying a camera is deployed in a home, the access to the camera should be secured; even more when the robot is managing medical information. We have developed all our software for the autonomous behaviour of the robots using ROS, so we need to consider its security.

Some works have been sketched in this area, as for instance in [3].

All authors are with the Robotics Group (http://robotica.unileon.es) and the Research Institute on Applied Sciences to Cybersecurity (http://riasc.unileon.es) at Universidad de León (Spain).
Corresponding author: vicente.matellan@unileon.es

A. Security assessment

There are three basic vulnerabilities threatening any computer system: availability (data interruption), confidentiality (data interception) and integrity (data modification). Other authors also add two more [9]: authenticity and non-repudiation (data fabrication from a non-trusted origin).

The concepts can be easily translated to industrial control applications [4] or to a robotic environment, due to the distributed and extensive network use approach presented by most extended robotics frameworks (Yarp, ROS, ROBOCOMP). On one hand, the robot is deployed in a home environment. It provides information to users and also carry out behaviours in order to fulfil required tasks. During this time, the robot perceive information from its sensors and update its behaviours. On the other hand, an attacker can attempt to make the robot or its network resources unavailable, this is a denial-of-service attack. The attacker can try to modify data messages in order to change robot behaviour, and also intercept the robot information about the environment and about the users personal data. Finally the attacker can simulate a sensor and generate new data available to the robot. Figure 1 summarizes graphically these vulnerabilities in a robotic environment. In this research we focus on confidentiality of data sent to and from the robot. Our approach consists in encrypt data transmitted by ROS nodes, without change ROS messages nor ROS functions to send the data.

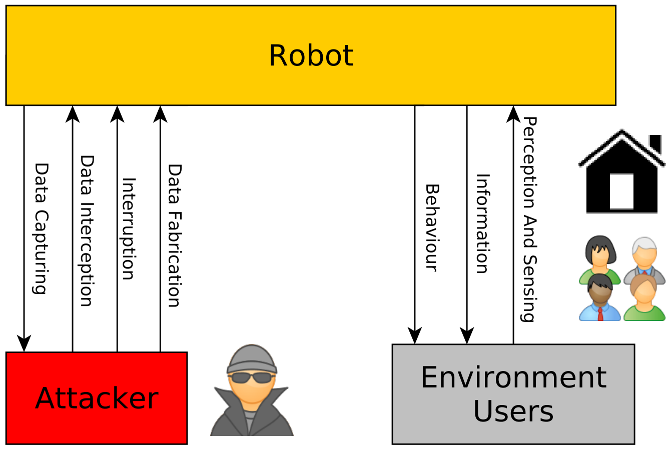

Fig. 1. Conceptual model of the security attacks.

B. ROS overview

ROS provides specific libraries for robotics as well as classical operating system services such as hardware abstraction (for sensors and actuators), low-level device control, and inter-process communication. Inter-process communication is based on a graph architecture where computation takes place in ROS processes named nodes. These nodes can receive and send messages, but no security was considered in the communication mechanism.

ROS framework is basically a message-passing distributed system. Its architecture is based on processes that publish *messages* to *topics*. For instance, a process (*node*) can be in charge of accessing a sensor, performing the information processing, and publishing it as an information structure on a named topic. Another process can *subscribe* to this topic, that is, read its information. Then the process can make a decision about the movement of the robot. Next, this node will publish the commands in another topic to send them to the motors. ROS nodes can be running in the same computer or in different computers.

This approach is very convenient for developers but can be easily tampered by malicious hackers. For instance, in [6] an experiment involving a ROS-based honeypot is described. The honeypot was a radio model truck with two cameras and a compass as sensors, and it was controlled from a remote ROS node written in Javascript and hosted in a remote enterprise grade web server. Vulnerabilities described in the paper comprise plain-text communications, unprotected TCP ports and unencrypted data storage.

The first step to solve some of these problems is to secure the communication channels by using an encryption mechanism. But how does encryption impact on the performance of a robotic system? This is the goal of this paper, characterize and evaluate different alternatives to secure ROS communication system and measure their performance.

The paper is organized as follows: Next section describes the testbed we have designed to measure the performance of the encrypted ROS system. Third section evaluates the data obtained in the experiments, and last section presents some conclusions obtained and further work.

II. Testbed description

Conventional ROS environment is composed by at least one ROS Master and some clients. ROS Master is the key element in the ROS system. It runs as a nameservice and manages registration information of topics and services used by ROS nodes.

When a node wants to stablish a connection with a topic, it communicates with Master to advise its registration information. After this step, the node gets more information about other registered nodes and can stablish new connections appropriately. The Master is updated in real time about clients information, and at the same time callbacks to those related nodes.

Figure 2 provides the graphical representation of this situation. In this case, two clients are connected to a topic. It is transparent to ROS Master this situation, The standard process starts with a basic sensor node (BSN). We have defined as BSN those ROS nodes in charge of to publish the information from the sensor.

We have prepared a testbed with two laptopts.

We have installed ROS Jade in two computers connected through a wired Ethernet 10/100 switch (model XXXX). In the first computer we have connected a Xtion camera and a Hokuyo laser. In the second computer have run a node visualizing the information from the sensors. Figure **??** shows this environment.

In this first approach we have changed the info published by the node, and instead of publish the information in a clean way, we publish the information encrypted. We use the 3DES algorithm that provides a Triple DES (3DES) references the Triple Data Encryption Algorithm (TDEA or Triple DEA). It is a symmetric-key block cipher that applies the Data Encryption Standard (DES) cipher algorithm three times to each data block. It is standardized by NIST in the Recommendation for the Triple Data Encryption Algorithm Block Cipher (NIST Special Publication 800-67).

DES algorithm has a fixed data block size of 8 bytes. Its keys are 128 (Option 1) or 192 bits (Option 2) long. However, 1 out of 8 bits is used for redundancy and do not contribute to security. The effective key length is respectively 112 or 168 bits.

3DES consists of the concatenation of three simple DES ciphers.

The plaintext is first DES-encrypted with K1, then decrypted with K2, and finally encrypted again with K3. The ciphertext is decrypted in the reverse manner.

The 192 bit key is a bundle of three 64 bit independent subkeys: K1, K2, and K3.

The 128 bit key is split into K1 and K2, whereas K1=K3.

It is important that all subkeys are different, otherwise 3DES would degrade to single DES.

3DES is cryptographically secure, although it is slower than AES algorithm.

The decorator pattern can be used to extend (decorate) the functionality of a certain object statically, or in some cases at run-time, independently of other instances of the same class, provided some groundwork is done at design time.

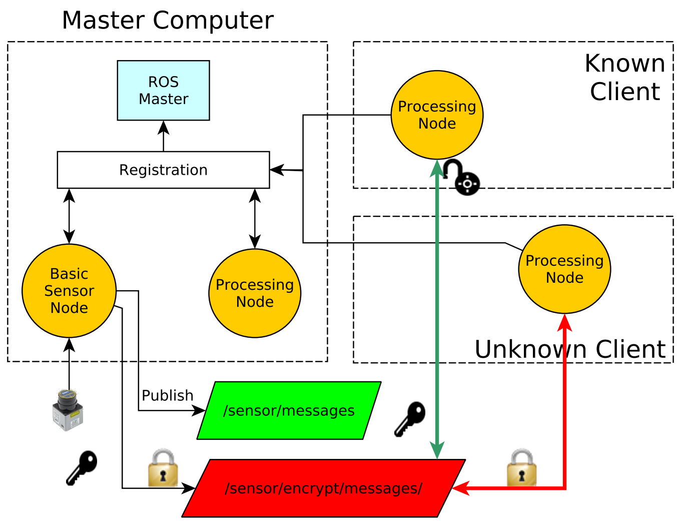

Fig. 2. Scheme of the scenario used for testbed.

A. *Robotic testbed*

We want to evaluate whether ciphering communications would affect the performance of ROS. In the second experiment we changed the first computer for a RB1 robot and the XXX switch by a wireless one. This robot was also running ROS Jade.

III. Experimental Measurements

To illustrate the described approach, we present an ad-hoc implementation of an encryption system to change part of the message used for transmit or receive robot sensors information. We maintanin in each case

The same measurements were made in the second environment to see if the use of wireless systems and a real robot would have any influence.

We have addded a funtion to our program in order to measure the time spent in each encrypt and decrypt call. The function is a python method presented as a decorator pattern The decorator pattern is used here to extend the functionality of encrypt/decrypt at run-time.

```
def fn_timer(function):
          @wraps(function)
          def function_timer(*args, **kwargs):
              v_time_0 = time.time()
              result = function(*args, **kwargs)
              v_time_1 = time.time()
              return result
          return function_timer
```

A. *Text Messages*

In this test we have used a modified version of talker/listener tutorial proposed by ROS. ROS statistical data shows the simple publish/subscribe example sending 11 string messages (*std_msgs/String*) by second of "Hello wold" plus a timestamp with 440 byts of traffic. In this manner, we want to analyse the behaviour of the system using bigger messages: T1, 262144 bytes; T2, 524288 bytes; and T3, 1048576 bytes

The measurments of behaviour of ROS traffic presents with simple plain messages *delivered_msgs* 10 of T1 generate . When T2 messages are sent the system runs with 4194368 bytes of traffic and finally when it is working in T3 messages just send 4194336 bytes and a total of 4 messages. There is no dropped messages by subscription.

We want to determine the duration of execution of our talker. We run the Linux *time* command to measure the total CPU time consumed by the ROS talker process. Firstly we analyzed T1, that presents in a time window of 32.884 seconds a user time of 1.364 seconds and a sys time of 0.044s. It represents a CPU time of 1.408 seconds. Secondly we analyzed T2, that presents in a time window of 33.749 seconds a user time of 2.508 seconds and a sys time of 0.164s. It represents a CPU time of 2.672 seconds. Finally we analyzed T3, that presents in a time window of 34.731 seconds a user time of 5.140 seconds and a sys time of 0.120s. It represents a CPU time of 5.260 seconds.

This is totally different when we are working with encrypted text messages of type T1, T2 or T3. Fig 3 presents the main differences. We have repeated the same experiment, but this time calling an encryption method that encrypts from plain text to DES3. It is possible to see that the encrypt process consumes more CPU than the plain process. For instance, T1 type presents running for 34.486 seconds a total of 23.008 of CPU. This means that the total CPU time increases almost 62%. This is even higher in T2 and T3 types with almost the 98% of real execution time.

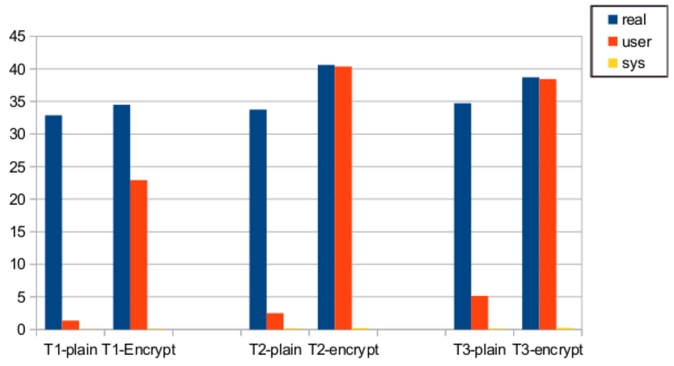

Fig. 3. Time of CPU spent on a simple talker/listener test.

[REESCRIBIR BIEN ESTE PÁRRAFO]. If we focus on the encryption/decryption process, we can see that the time needed for the encryption function shows in 1040 messages a minimal time of 0.059246 seconds and a max time to encrypt the string of 0.683408 seconds with an average of 0.063858 seconds and standard deviation of 0.000388.

This time increases with T2 and T3 types. T2 type needs 0.123383 seconds in average with standard deviation of 0.000040 seconds and T3 0.247303 with a standard deviation of 0.000137.

B. *Camera*

1) Process evaluation: We have used a Logitech, Inc. QuickCam Pro 9000 webcam.

There are three nodes involved in our system: usb_cam, encrypted_node and decrypted_node. The first and the sencond run in the master computer, the decrypted one in a separate computer.

There are three topics involved:

To analyse the behaviour of the system under encrypt/decrypt circumstances we use an Intel i7 computer with 16 GB RAM. The ROS master system has 234 process running by default in a Ubuntu 14.04, the client is running 242 process in the same operative system.

In this case we measure the time needed to encrypt or decrypt the ROS message (*sensor_msgs/Image.msg*). We just encrypt or decrypt the field data of *uint8[]* that is treated as a string. To evaluate the frame rate, our nodes present this information during the execution in the visualization frame .

This experiment was running for about 18 hours sending in total 971790 frames. The minimal time to encrypt the images was 0.001309 seconds, the max time 0.026909 seconds with a mean of 0.010948 seconds and a standard deviation of 0.000004 seconds. The mode value in this phase was 0.010571 seconds wich was reapeated 415 times.

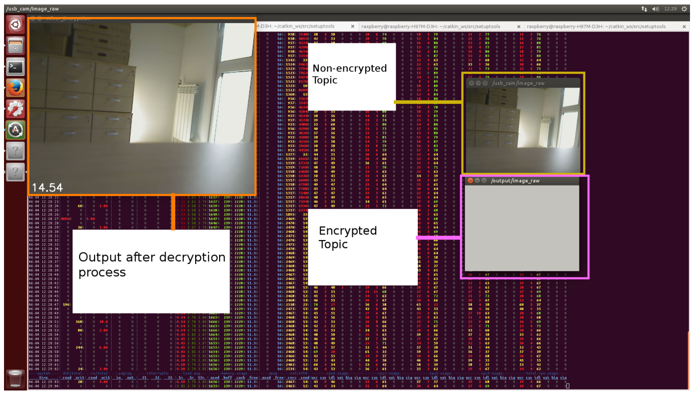

Fig. 4. Screenshot taken during image encryption test.

The minimal time to decrypt the images was 0.001288 seconds, the max time was 0.039130 seconds with a mean of 0.008828 seconds and a standard deviation of 0.000003 seconds. The mode value in this phase was 0.008183 seconds wich was reapeated 593 times.

Figure 4 presents the situation in the client node. We have used the ROS node *image_view* to visualize the images from topics. There is an non-encrypted topic that we have maintained in order to validate delays. There is an encrypted topic that is not possible to visualize using the released image_view ROS node. Finally there is a frame that presents the image after decrypt the topic using our own decrypt node.

The FPS in the encrypter machine is 14.56 and the FPS in the decryption machine is 14.54.

IV. Discussion

Fig 5 outlines the behaviour of the talker cyphering data. Left part of the figure presents the box plot of the three types of message. The right part of the element present the number of repetition associated to a time to each encryption process.

We observe that the delay induced by encryption does not reduce the standard frame rate.

The figure 6 presents the box-plot of the encryption decryption data analyzed and the histograma.

V. Conclusion and Further Work

This article has introduced the use of encrypted information through ROS communication system. We have evaluated the impact of this added feature from two points of view: CPU consuming and network traffic.

We have shown that our approach avoids security problems related with the plain-text publish/subscribe paradigm used by ROS. However, the overhead of CPU performance and communication load should also be considered in distributed architectures that need to work on real time.

As we pointed out in the introduction, securing communications is just one dimension in the cybersecurity of autonomous systems. If we want to see these machines working in our homes we need to secure navigation abilities and interaction mechanisms, to avoid manipulated or malicious behaviours and make robots reliable assistants for every person at home.

Acknowledgment

The authors would like to thank the Spanish Ministry of Economy and Competitiveness for the partial support to this work under grant DPI2013-40534-R and to the Spanish National Institue of CyberSecurity (INCIBE) under grand Adenda21 ULE-INCIBE.

References

[1] Tamara Denning, Cynthia Matuszek, Karl Koscher, Joshua R. Smith, and Tadayoshi Kohno. A spotlight on security and privacy risks with future household robots: Attacks and lessons. In *Proceedings of the 11th International Conference on Ubiquitous Computing*, 2009.

[2] Conor Fitzgerald. Developing baxter. In *Technologies for Practical Robot Applications (TePRA), 2013 IEEE International Conference on*, pages 1–6. IEEE, 2013.

[3] Jérémie Guiochet. Hazard analysis of humanrobot interactions with HAZOPUML. *Safety Science*, 84:225–237, 2016.

[4] Peter Huitsing, Rodrigo Chandia, Mauricio Papa, and Sujeet Shenoi. Attack taxonomies for the modbus protocols. *International Journal of Critical Infrastructure Protection*, 1:37–44, 2008.

[5] Francisco Martn, Jos Mateos, Francisco Javier Lera, Pablo Bustos, and Vicente Matelln. A robotic platform for domestic applications. In *XV Workshop of Physical Agents*, 2014.

[6] Jarrod McClean, Christopher Stull, Charles Farrar, and David Mascareñas. A preliminary cyber-physical security assessment of the Robot Operating System (ROS). *SPIE Defense, Security, and Sensing*, 8741:874110, 2013.

[7] Santiago Morante, Juan G Victores, and Carlos Balaguer. Cryptobotics: Why robots need cyber safety. *Frontiers in Robotics and AI*, 2(23):1–4, 2015.

[8] Morgan Quigley, Ken Conley, Brian P. Gerkey, Josh Faust, Tully Foote, Jeremy Leibs, Rob Wheeler, and Andrew Y. Ng. Ros: an open-source robot operating system. In *ICRA Workshop on Open Source Software*, 2009.

[9] Y Sattarova Feruza and Tao-hoon Kim. It security review: Privacy, protection, access control, assurance and system security. *International Journal of Multimedia and Ubiquitous Engineering*, 2(2):17–31, 2007.

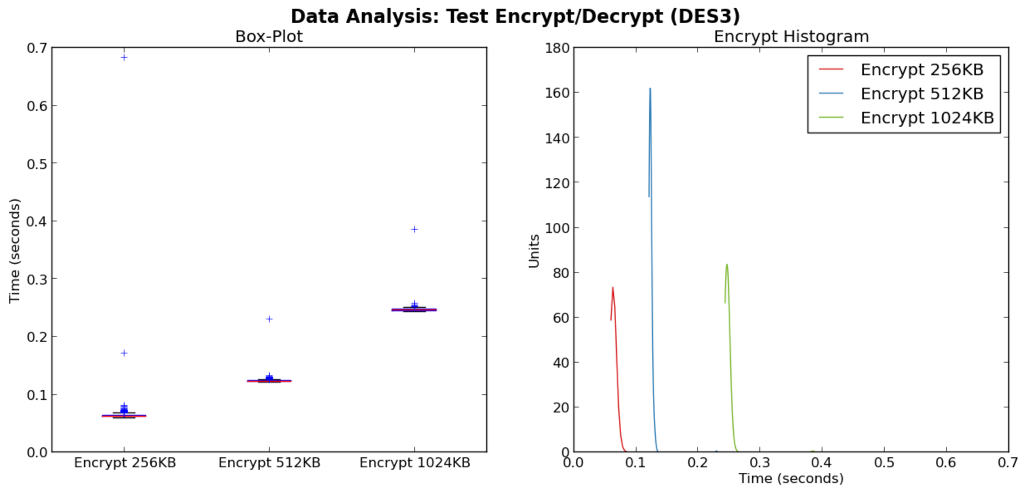

Fig. 5. Time spent on each call to encryption/decryption function.

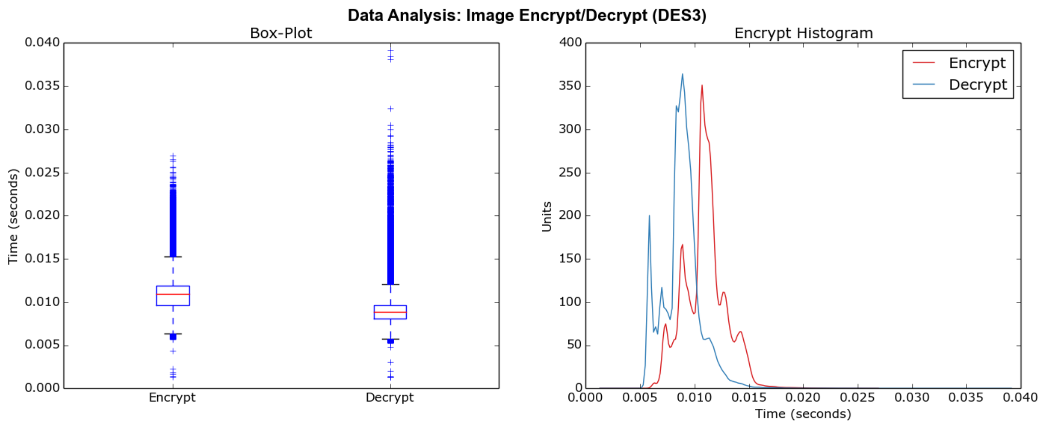

Fig. 6. Time spent on each call to encryption/decryption function.