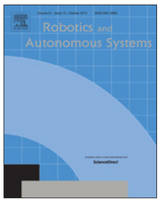
Contents lists available at ScienceDirect

Robotics and Autonomous Systems

journal homepage: www.elsevier.com/locate/robot



Quantitative analysis of security in distributed robotic frameworks

Francisco Martín, Enrique Soriano, José M. Cañas [*]

Universidad Rey Juan Carlos, Spain

HIGHLIGHTS

- Many robotic applications are built with networked software components.
- A threat model has been defined for robotic systems in networked system.
- Experiments with communication middlewares in robotics frameworks are included.
- Security does not significantly decrease the quality of robotic communications.
- Some communication middlewares do not cope with big sensor readings.

ARTICLE INFO

Article history:
Available online 21 November 2017

MSC:
00-01
99-00

Keywords:
Security
Communications
Middleware
Frameworks
Robotics

ABSTRACT

Robotic software frameworks simplify the development of robotic applications. The more powerful ones help to build such applications as a distributed collection of interoperating software nodes. The communications inside those robotic systems are amenable of being attacked and vulnerable to the security threats present on any networked system. With the robots increasingly entering in people's daily lives, like autonomous cars, drones, etc. security on them is a central issue gaining attention. This paper studies several well known communication middleware used by robotic frameworks running on robots with regular computers, and their support for cybersecurity. It analyzes their performance when transmitting regular robotic data of different sizes, with or without security features, on several network settings. The experiments show that security, when available, does not significantly decrease the quality of the robotic data communication in terms of latency and packet loss rate.

1. Introduction

Robots have been traditionally used in industrial scenarios, but in the last years they have matured so much that their use is expanding to homes and streets, with people increasingly trusting them in some part of their life. For instance, robotic vacuum cleaners are now robust appliances in many homes. Commercial drones are now being used for event recording and their use in logistics is being explored. Autoparking capability and autonomous autopilots are included in some car models, driverless cars are already legal for research in several places and autonomous transportation systems are being tested with general public.

On their hardware, robots integrate sensors, actuators and computers. Their intelligence, algorithms and behaviors mainly rely on their software. Nowadays software is more than half of the cost of a robotic system. The robot software, as any other software, is amenable to suffer from security attacks. Many robots have external connectivity, for instance to let the user interact with them (like interacting with the vacuum cleaner through the smartphone). In addition, robots are complex systems and their software rely on robotic frameworks that simplify the development of applications. Most successful robotic frameworks are distributed. Such useful connectivity and distribution open the door to vulnerabilities and robot hacking.

When the target is a robot system, security attacks have logical and physical impact. The consequential damage of a hacked robot is directly commensurate with the amount of trust put into the system and is fully dependent on the capabilities of the robot. Simply hacking a robot to operate slightly out of a specified configuration mode can lead to everything from minor damage to death. Many attack examples have been reported, like the capture of a US Predator military drone (UAV) by the Iranian forces in 2011 [1]. Another relevant example is the remote hacking of a Jeep Cherokee car in 2014 [2].

Typically the robotics community has lived in a "happy naivety" [3]. For instance, the most popular robotic framework does not include any security mechanism. Researchers have discovered multiple common security flaws in mainstream robotic technologies from leading vendors, leaving them wide open to

[*] Corresponding author.
E-mail address: jmplaza@gsyc.es (J.M. Cañas).

https://doi.org/10.1016/j.robot.2017.11.002
0921-8890/© 2017 Elsevier B.V. All rights reserved.

attack. Recently, cybersecurity in robotics is gaining attention [4,5]. There is an increasing number of presentations about robotic systems in cybersecurity conferences (like DEF CON or RSA) and journals (like ACM Transactions on Cyber–Physical Systems (TCPS)), including many domains like rescue robotics [6], teleoperated robots [7,8] or industrial robots [9].

Providing mechanisms to avoid undesired attacks and exploits in robot communication software is becoming increasingly required. Most promising solutions include security mechanisms inside the robot frameworks. Not all the security solutions are suitable for robotic environments. For instance, they have to comply with the general requirement of real-time operation, their computational cost cannot be huge.

The goal of this paper is to quantitatively study the security solutions for communication explored inside distributed robotic frameworks. The performance of different communication middleware like Ice, Fast-RTPS (a DDS implementation) and the ROS transport system are compared when transmitting typical robotic data of different sizes (laser readings, images and point clouds). The experiments have been carried out in several network scenarios: local inside a single machine, an ethernet network and a WiFi network. Data transmission with and without security have been done in the bench tests, and the cost of the secure communications has been measured in terms of latency and data loss rate.

Section 2 presents the utility of the robotic frameworks and some illustrative examples. The security problems in robot software are introduced, classified and commented in Section 3, including three communication choices. Section 4 describes the methodology followed, the experiments performed and the results. Finally some conclusions end the paper.

2. Robotic frameworks

In the last years, several robotic frameworks (SDKs) have appeared. They simplify and speed up the development of robot applications [10,11] . They favor the portability of applications between different robots, and ease the code reusability and integration. Modern robotic frameworks are based on software engineering criteria more than in cognitive issues. Their major achievements are: (i) the hardware abstraction, hiding the complexity of accessing heterogeneous hardware (sensors and actuators) under standard interfaces; (ii) the distribution capabilities, that allow to run complex systems spread over a network of computers; (iii) the multiplatform and multilanguage capabilities, that enable the developer to program and run the software in several computer types, robots and programming languages; and (iv) the existence of big communities around them, that share code, tools and algorithms.

There are several communication mechanisms and choices for providing the distribution capabilities. For instance, the Publish–Subscribe paradigm, the Remote Procedure Call (RPC) paradigm, distributed object-oriented models (like Common Object Request Broker Architecture, CORBA), the use of name servers, the use of central servers vs peer-to-peer communications, the use of specific well known communication middleware like Ice or DDS implementations vs the development of ad-hoc messaging software, etc.

Ice (Internet Communications Engine [12]) is an efficient, open source and object oriented RPC framework that provides SDKs for C++, Python, Java and other languages. It can run on various operating systems, including Linux, Windows, OS X and Android. It implements a proprietary communications protocol, called the Ice protocol, that can run over TCP, TLS, UDP, and WebSocket.

DDS (Data Distribution Service [13]) is a machine-to-machine standard that aims to enable scalable, real-time, dependable, high-performance and interoperable data exchanges using a Publish–Subscribe pattern. Both commercial and open-source software implementations of DDS are available. These include application programming interfaces (APIs) and libraries in Ada, C, C++, Java and other languages.

2.1. Advantages

Frameworks offer a more abstract access to sensors and actuators than the operating systems of simple robots do. The SDK Hardware Abstraction Layer (HAL) deals with low level details accessing to sensors and actuators, releasing the application programmer from that complexity. In addition, it provides high-level, easy-to-use interfaces.

Frameworks also provide a particular software architecture for robot applications, an specific way to organize the programs and deal with code complexity when the robot functionality increases. There are many options here: calling to library functions, reading shared variables, invoking object methods, sending messages via the network to servers, etc. Depending on the programming model, the robot application can be considered an object collection, a set of modules talking through the network, an iterative process calling to functions, etc.

In addition, robotic frameworks usually include simple libraries, tools and common functionality blocks, such as robust techniques for perception or control, localization, safe local navigation, global navigation, social abilities, map building, etc. Libraries shorten the development time and reduce the programming effort needed to code a robotic application as long as the programmers can build it by reusing the common functionality included in the SDK, keeping themselves focused in the specific aspects of their application.

2.2. Ros

The Robot Operating System (ROS) [14,15] is one of the biggest frameworks nowadays. It was founded by Willow Garage as an open source initiative and it is now maintained by Open Source Robotics Foundation. It has a growing user and developer community and its site hosts a great collection of hardware drivers, algorithms and other tools. ROS is a set of software libraries and tools that help to build robot applications (it includes from drivers to state-of-the-art algorithms, and with powerful developer tools simplifies the development of robotics projects). It is multiplatform and multilanguage.

The main idea behind ROS is an easy to use middleware that allows connecting several components (nodes) implementing the robotic behavior in a distributed fashion, over a network of computers using hybrid architecture. ROS is developed under hybrid architecture by message passing, mainly in Publish–Subscribe fashion (*topics*). Message passing of typed messages allows components to share information in a decoupled way. Therefore, the developer does not require to know which component sends a message, and vice versa, the developer does not know which component or components will receive the published messages.

Nodes send and receive messages on *topics*. A *topic* is a data transport system based on a Publish–Subscribe system. One or more nodes are able to publish data to a topic, and one or more nodes can read data on that topic. A topic is typed, the type of data published (the message) is always structured in the same way. A message is a compound data structure. It comprises a combination of primitive types (character strings, Booleans, integers, floating point real numbers, etc.) and messages (a message is a recursive structure). RPC mechanisms (like *services*) are available as well. Resources can be reached through a well defined naming policy and a ROS master.

2.3. Other frameworks

Another important example is ORCA [16], an opensource framework for developing component-based robotic systems. It provides the means for defining and developing the building-blocks which can be pieced together to form arbitrarily complex robotic systems,