Università
Ca'Foscari
Venezia

Doctor of Philosophy
Programme Computer Sciences

Cycle XXXIII

Final Thesis

Application-level Security for Robotic Networks

SSD: INF/01

Programme Coordinator
Prof. Agostino Cortesi

Supervisor
Prof. Agostino Cortesi

Graduate Student
Gianluca Caiazza
Matriculation Number 840009

Abstract

Cyber-physical systems (CPS) are increasingly deployed as part of the interconnected robotic cyber-infrastructures which are known as the Industrial Internet of Things (IIoT) network. Those pervasive devices are capable of automatizing various tasks and provide novel functionalities in a wide range of applications. However, this growth made the devices a worthwhile target for attackers and cybercriminals as well. The new frontiers of large-scale deployments of connected smart devices, in which we observed a tremendous growth in the amount of stored and processed sensitive data, have maturated into a widespread suspicion concerning the way in which these flow into the infrastructures. How do we make these devices safe? How can we verify their correct operation? Due to the intrinsic limitations of those devices, either from the power consumption point of view and the actual computational power at our disposal, engineering cybersecurity solutions are not trivial. This thesis focuses on discussing and developing security solutions for those networks by analyzing the application of the security pillars of Confidentiality, Integrity, Availability, Privacy, Authenticity and Trustworthiness, Non-Repudiation, Accountability, and Auditability. We provide an overview of the robotic scene and introduce the Robot Operating System (ROS), the framework we adopted as a testbed of our solutions. More in detail, this work discusses (i) novel solutions in the field of authentication and authorization in access control architectures and policy generation, management, and distribution, (ii) vulnerabilities and countermeasures in robotic frameworks, and (iii) novel approaches of network vulnerabilities excavation and accountability. In order to provide agnostic research tools and results, we develop static solutions at the application-level that could exploit prior offline computation power.

The main results of the thesis can be summarized as follows:

1. A state of the art analysis of application-level threats on a general robotic framework and an in-deep review of the attack surface on ROS
2. The formalization of novel approaches to access control architectures distribution, and the dissertation of an advanced policy management tooling we developed in the field of authentication and authorization
3. The definition of a novel network vulnerability excavation tool and discussion on attribute-based encryption to tackle privacy issues
4. Creation of a blockchain-powered software-based black box for a robotic network to address Accountability and Non-Repudiation

The results discussed in this thesis give a solid base for the definition of the future security mechanisms for robotic devices that could be easily and securely integrated into big-scale deployments spreading security solutions by reducing overall the tradeoff between security and usability.

Acknowledgements

Gianluca Caiazza
Venice, February 2021

First and foremost, I would like to express my sincere gratitude to my supervisor Prof. Agostino Cortesi, for his guidance, his support, and his enthusiasm throughout those years. My gratitude goes to the reviewers of this thesis, Prof. Carlo Ferrari, Prof. Fabio Martinelli, and Prof. Corrado Santoro, for their comments and useful suggestions.

I am grateful to the people I had the opportunity to work with throughout this thesis for their invaluable support in broadening my perspectives and improve my scientific background. I would particularly like to thank Dr. Pietro Ferrara for the many suggestions and the time spent reading this work.

I am beholden to my friend and coauthor Ruffin White, to whom I will be forever thankful for the wholehearted dedication and the many sleepless nights spent working together.

Furthermore, I would like to thank all the colleagues at Ca' Foscari. I would particularly like to single out Martina Olliaro for the amusing moments shared while backing each other up.

I also want to thank some friends: particularly Francesco Di Sano, for his never-ending interests, the fruitful discussions and the spur to dive into so many different adventures. Maksym and Francesco, for their friendship and support. Gianmarco, Giacomo and Marco, for the fantastic time spent together. Last but not least, Alessio Ferraresso, Alessio Gravinese, Edoardo and Luchino Magnone, for their friendship and the great memories of the time I spent in Seoul.

Finally, I want to thank my family: my parents Saverio and Concetta, my brother Manlio, my uncle Vincenzo, my Grandparents Nunzio and Giuseppina; for motivating me to reach my objectives and supporting me spiritually not only throughout the writing of this thesis but also during my whole life.

Contents

Introduction

The number of smart connected devices, known as cyber-physical systems (CPS), has grown exponentially in the last few years forming the so-called Internet of Things (IoT) network, a large interconnected cyber-infrastructure of smart robotic devices, sensors, and actuators. Industry 4.0 represents a shift in the future towards the adoption of those ubiquitous connected robotic systems in the so-called Industrial IoT (IIoT) network. We see applications of such devices in various domains as healthcare, energy management, smart cities, intelligent transport systems (ITS), Cooperative-ITS (C-ITS), agriculture, etc. Robots have pervaded those domains alongside Machine Learning (ML) and Artificial Intelligence (AI), defining the so-called 'intelligent' devices networks [52].
To cope with the requirement for custom device-tailored solutions, a significant number of different IoT platforms has emerged [65]. We can easily observe how these are more oriented to the so-called Consumer Internet of Things (CIoT) rather than to Industrial IoT (IIoT) solutions. However, the high growth rate of consumer solutions in addition to the steady development of Industrial applications has rapidly overcome the security measures that were deployed originally for offline isolated *one-purpose* devices [5].
The security requirements for IoT networks are greatly influenced by the purpose they serve. In such cases, as discussed by Dieber *et al.* [33], the information security pillars of Confidentiality, Integrity, and Availability (CIA) that are generally applied to secure information systems assume different priorities. Generally, in IoT robotic networks, we consider a subset of the CIA pillars' scope: confidentiality corresponds to privacy, authenticity coincides with integrity, while availability is usually addressed via hardware redundancy.
The transition from private network machine-to-machine (m2m) connectivity to a global online network, has exposed those devices to threats previously not considered. Moving from a greenfield private network, in which only known and controlled devices lives, to a brownfield heterogeneous network (e.g Internet), raises several security concerns. With the widespread deployment of those devices, cybersecurity becomes pivotal in preventing their exploitation as an attack tool (e.g. Mirai botnet [50]). Unfortunately, cybersecurity has not been highly prioritized during the design and manufacture of IoT mobile robots. In the early days, robotic systems were simply intended as '*physically*' enhanced computers without specific constraints or limitations. Since the goal was to develop the fastest, lightest, and most practical solution for prototyping and deploying products, overall, often the security component and the associated privacy risks were overlooked. As discussed by Clark *et al.* [25], higher priority has been placed on lowering overall development costs and reduce time to market to deliver functionality to

consumers, which resulted in a series of intrinsic technical debts for the entire field that strongly affects the security of the products.
As discussed by Morante *et al.* [66], the emerging security problems that the field of robotics faces now are similar to those presented for computers in the early days of the Internet. In fact, robotic networks are generally subject to classical cyber-physical attacks such as Denial of Service (DoS), eavesdropping, tampering, repetition, spoofing, etc. By looking at OWASP 2018 Top 10 for the Internet of Things[1] as well as for Web applications Top 10, compared in Table 1, we can easily spot the similarities and correlations between the issues in both charts. As we can see, for example, data privacy management, authentication, lack of monitoring, are all shared issues.

Internet of Things	**Web Security**
Weak, Guessable, or Hardcoded Password	Injection (e.g. SQL)
Insecure Network Services	Broken Authentication
Insecure Ecosystem Interfaces	Sensitive Data Exposure
Lack of Secure Update Mechanism	XML External Entities (XXE)
Use of Insecure or Outdated Components	Broken Access Control
Insufficient Privacy Protection	Security Misconfiguration
Insecure Data Transfer and Storage	Cross-Site Scripting (XSS)
Lack of Device Management	Insecure Deserialization
Insecure Default Settings	Components with Known Vulnerabilities
Lack of Physical Hardening	Insufficient Logging & Monitoring

TABLE 1: OWASP Top 10 Internet of Things and Web Service

A sample scenario of those issues can be foreseen from the vulnerabilities discovered in 2017 when 500.000 pacemakers have been recalled by the US Food and Drug Administration (FDA) due to the fears that they could be hacked to run the batteries down or even alter the patient's heartbeat via malicious firmware update [43]. Similar threats are common to other robotic platforms applications such as autonomous cars, drones, assistance medical robots, etc [4]. This is even more critical considering the large scale introduction of robotic applications into daily life coupled with the further adoption of wireless remote access mechanisms. Ensuring security and privacy for the robotic platforms is thus critical, as failures and attacks could have devastating consequences.

Considering the similarities between IoT devices and general Computers, despite being tempting, we can't straightforwardly apply the same mechanisms for cybersecurity. In fact, although robotic and computer networks may seem similar, they are greatly different [23]. In robotic networks, we need to consider different tradeoffs between safety-critical corner cases, and security and real-time constraints that are not present in other systems. We can find a significant number of devices that need to operate in conjunction with each

[1]https://owasp.org/www-pdf-archive/OWASP-IoT-Top-10-2018-final.pdf

other. Therefore, it is necessary to have a good understanding of the robotics system to assess security risks and threats.

Challenges

As discussed by the authors in [52], considering how robotic devices and smart applications are pervading several fields, even becoming essential, the risk associated with cyber-attacks should not be underestimated. In terms of costs and disruptiveness, those represent a huge threat to numerous systems. Therefore, different research areas should be investigated by targeting the following objectives:

- **Confidentiality:** several works have focused on targeting confidentiality in IoT especially for privacy management. In particular, we can identify two major risks associated with unauthorized access to private information (e.g. log, history, IP, etc), and insecure communication over an unencrypted channel by sending plaintext payload outside the device.
- **Availability:** in IoT devices, especially in a critical application, we can achieve availability via hardware redundancy. However, in the case of price constraints or device dimension issues, we can enhance the robustness of the product via monitoring and preventing analysis.
- **Authenticity (Trustworthiness):** one of the most underestimated threats is represented by authenticity. The trustworthiness of robotic devices is paramount to their utilization in safety-critical applications. Through this property, we can guarantee the integrity of the system which is pivotal to leverage other critical system's properties. Attacks in this area usually focus on bypassing authentication mechanisms thus tampering with the components and compromise the system.
- **Accountability (Non-Repudiation):** in this research area, we want to keep agents in the network accountable for their actions. This is critical in applications that require collaboration between peers in conjunction with critical constraints in regards to safety and verification.
- **Auditability:** being able to design solutions that incorporate audit mechanisms to keep the system running and organized is paramount in a highly distributed autonomous network. Especially when multi-tier heterogeneous solutions are deployed, this is not trivial to achieve.

Methodology

In order to discuss our approach, we need to introduce a reference architecture for the IoT analysis. There exist several architecture models in the literature optimized for specific applications [48, 57]. For example, the authors in [92, 81] introduced a five-layer architecture based on Service Oriented Architecture (SOA) to integrate IoT in enterprise services. Again, when analysing smart systems, Computer-Integrated Manufacturing (CIM) model is often used [88]. However, for the sake of our analysis, we can reduce all of them to a basic three-layer architecture, depicted in Figure 1 composed by: Application, Network, and Perception layer.

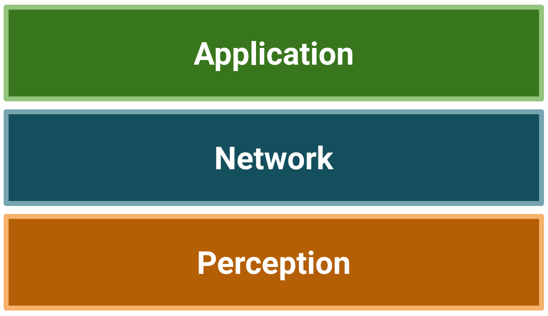


FIGURE 1: Three-layer IoT architecture model

In detail each layer is composed as follows:

- **Perception Layer:** this is the lowest level which connects the hardware components to the device. In this layer, we collect all the environmental values (e.g. temperature, speed, location, video footage, etc) using end-point nodes that are physically connected to sensing technology (e.g. GPS, thermometer sensors, cameras, etc). We can apply some basic cybersecurity solutions such as lightweight encryption for a basic degree of data confidentiality, and key agreement between the components to enforce authentication.
- **Network Layer:** this is the layer that is in charge of managing the connection from the physical perception layer to the application realm. Through various network technologies (e.g. 3/4/5G, Wi-Fi, Bluetooth, Zig-Bee, etc) it manages the information route and storage offering high-level representation (wrappers) of real-world peripherals data to middleware (e.g. cloud, robotic framework, etc). We can apply cybersecurity at this level with the usage of cryptographic key management, intrusion detection systems (IDS), and routing security (e.g. via demilitarized zone (DMZ)).
- **Application Layer:** this layer is in charge of all the application-specific services to the users. It contains the core logic of the system and the high-level protocols that compose the 'smart' side of the device. Here we can apply more complex information security management solutions that are agnostic with respect to the hardware.

The goal of this thesis is to define cybersecurity solutions that can be applied to multi-tier heterogeneous IoT networks. Since each application differs from the other, we want to design agnostic solutions in order to offer mechanisms that can easily be applied and integrated regardless of the underlying devices. This leads us to target the application-layer. Therefore, our research scrutinized this layer and present an in-deep analysis of the core security objectives introduced above focusing on the development of those on the open-source robotic framework Robot Operating System ROS1/ROS2, and on Data Distribution Service (DDS).

In the thesis, we tackle all the challenges highlighted so far, either by exploiting formal methods or using techniques that are nevertheless inspired by rigorous and disciplined principles.

Main contributions

Regarding the research field of mobile IoT networks, we contribute by proposing a range of frameworks to assist developers and auditors in analyzing, developing, and secure highly distributed robotic networks. Our particular approach focuses on the application-layer of the systems by providing agnostic solutions that can be applied to virtually all IoT networks. Our goal is to develop powerful security solutions to reduce the trade-off between security and usability.

By analyzing the state of the art and the threats at the application-layer we identify the major sources of security concerns in those applications. To exemplify the threats, we used Robot Operating System as a testbed and provide an in-deep review of how it works and why it is a good representative of the robotic frameworks. Then, we guide through a pentesting analysis of the framework with the usage of two tools we contribute for the security verification of ROS' applications. Our goal, at this stage, is to demonstrate the false sensation of security given by a firewall or VPN, by showcasing the threats that characterize the field.

In this context, we discuss the pivotal components of access control in a robotic network. In this topic the contribution is twofold. First, we provide an overview of the main properties and principles of authentication schemes. Additionally, we analyze the policy distribution in this kind of network and we contribute two novel approaches for secure policy attestation between peers by leveraging on the identity and attribute certificates defined by the x.509 standard. The second contribution is on access control policies. First, we provide the properties and general understanding of the subject. Next, we introduce our novel framework for automated systematic generation and verification of necessary cryptographic artifacts via a meta-build system layout via workspaces and plugins, namely ComArmor and Keymint. Finally, we experimentally assess the effectiveness of the toolchain by testing the framework to a publish-subscribe network commenting on the results.

We continue along this line of research, by contributing a formal verification tool developed for automatic excavation of vulnerabilities in distributed IoT networks. To showcase our work, we use the framework Data Distribution Service (DDS). We present a threat model and our approach to the reconstruction of the topological heuristic graphs via lazy evaluation. Then, we discuss reachability verification and the complexity of the solution. Finally, we experimentally demonstrate the effectiveness by showcasing a simple application and discuss countermeasures to those threats. Lastly, we discuss the usage of Attribute-Based Encryption (ABE), for secure policy distribution on public communication channels.

About accountability and non-repudiation, we contribute a novel application of an Event Data Recorder (EDR) based upon cryptographic linked integrity proofs, disseminated via distributed ledgers, namely Black Block Recorder (BBR). The approach combines the use of Digital Signature Algorithms (DSA), keyed-hash Message Authentication Codes (HMAC), and Smart Contract (SC) via Distributed Ledger Technology (DLT) to enable tamper-evident logging while considering the limited resources available for mobile robotic deployments. Finally, we perform an in-depth analysis of the software-based solution and experimentally assess the performance of the framework on ROS2 and Hyperledger Sawtooth.

Structure of the Thesis

The thesis is structured as follows:

- Chapter 1 introduces Robot Operating System (ROS) and discuss agnostic application-level attacks to robotic application and countermeasures. Then, we introduce ROS2 and comment on the evolution of the robotic middleware.
- Chapter 2 discusses access control policies and proposes a server-less and a server-centric policy distribution architecture with the usage of x.509 identity and attribute certificate, and a novel meta-built framework for procedurally provision access control policy.
- Chapter 3 discusses vulnerability excavation in distributed network via formal analysis techniques by exploiting passive sniffed discovery payload in DDS.
- Chapter 4 presents our framework for accountability in distributed robotic network with the usage of blockchain technology.

Chapter 1

ROS: What, Why and How

The aim of this chapter is to introduce the main features of the Robot Operating System (ROS) from the Open Source Robotic Foundation (OSRF), the most popular robotic framework [87], either in academia and industry upon which we base our discussions. Since its creation in 2007, ROS has seen a steady increase in global grown. The enormous exposure of the framework, with over 7600 citations of the original paper [75], the blatant metrics [87], and the numerous community-driven events, in addition to several sector-specific consortia such as Ros Industrial (ROS-I), ROS Healthcare (ROS-H), ROS Micro Embedded Systems (micro-ROS), and ROS Hardware, well describe the pivotal position that it has in the market.
However, along with the increased adoption and application outside the academic realm, as evidenced by the multitude of robots available in the market [1], the framework became a worthwhile target for attackers as well. As with other CPS and IoT software solutions, ROS suffers from a lack of cybersecurity features necessary for real-world products outside a controlled environment. Due to the immaturity of standards in the field, this framework gives us a good representation of the products in the robotic scene.

Such as for similar frameworks, the ROS authors' primary goal was to develop software tools that users would need to undertake novel research and development projects. At time of its creation, a lot of effort was spent into defining levels of abstractions that would allow much of the software to be reused elsewhere, and ease the research and development in academia. However, their solution, influenced by the robotic scene at that time, was targeting platforms that do not suit the market anymore. A single robot with workstation-class computational resources at its disposal, no-real time support, excellent network connectivity (mostly wired), in addition to network flexibility - to ease development - is an easy target for an attacker.

The developed paradigm overshot the original target of the PR2 robot[2] and been applied to wheeled robots, industrial arms, legged humanoids, self-driving cars, aerial vehicles, surface vehicles, and many more.

[1]https://robots.ros.org/
[2]https://robots.ieee.org/robots/pr2/

Before we dive more into the *whys* and *hows* this framework gained its popularity, in this chapter, the reader will learn firsthand about the basic structures and low-level mechanisms of ROS; specific vulnerabilities, along with information about how they can be exploited, to manipulate ROS applications. Despite being ROS specific, the following attacks are not limited to the presented framework and allow us to lay the foundation for further discussions in the thesis.

The rest of this chapter is structured as follows:

- **Background:** in this section we introduce formally what ROS is, how it differs from other robotic platforms, and how it works.
- **Concepts:** in this section we discuss in detail how are defined and what purpose each level has inside the ecosystem.
- **ROS API:** based on what we learned in the previous section, here we analyze ROS APIs in greater detail. We present the inside mechanisms behind the abstraction level and how those translate into the actual implementation.
- **Connection:** in this section we specify how connections work in ROS. This is necessary to grasp the general rules behind the connection and obtain the tools to better understand the reasoning behind the attacks.
- **Attacks:** here we present some attacks to ROS applications and discuss their consequences.
- **Countermeasures:** in this section we present some solutions to the previously discussed attacks.
- **ROS 2:** in this section we discuss the evolution of the framework and introduce some key concepts behind ROS2, the next iteration of the framework, and Data Distribution Service (DDS).
- **Conclusions:** in this section we summarize what we have presented in the chapter, and introduce some basic concepts about the next chapters.

The work discussed in this chapter has been published in [32].

1.1 Background

ROS is defined as an open-source, meta-operating system for robots like manipulators, mobile robots, autonomous cars, social robots, humanoids, unmanned aerial vehicles (UAVs), and others. It provides the services and abstractions of an operating system, including standard hardware APIs, low-level device control, implementation of commonly-used functionality, message-passing between processes, and package management. It also provides tools and libraries for obtaining, building, writing, running code across multiple computers.

It implements a peer-to-peer network, namely the *graph*, in which the loosely coupled processes, known as *nodes*, can communicate at run-time via publish/subscribe pattern [34] using the ROS communication infrastructure.

ROS leverages the communication between processes on different styles, including synchronous RPC-style communication over **services**, asynchronous streaming of data over **topics**, and storage of data on the **parameter server**.

Those different communication styles, that we will analyze later on, allow the developers to fit their solutions and code into different containers - similarly to standard code reusing patterns - that can be shared among projects and ease the definition of the aforementioned logical abstractions. Indeed, we can see ROS as a distributed framework of processes that enables them to be individually designed and loosely coupled at run-time. This is possible thanks to the logical combination of three abstract levels, namely: Filesystem, Graph, and Community. This division allows making independent decisions about development and implementation, leaving to developers the ability to integrate different kinds of code.

1.2 Concepts

In this section, we dive more into the aforementioned ROS functionalities levels of concepts: the Filesystem level, the Computation Graph level, and the Community level. As we can see from 1.1, each level provides a different degree of abstraction to the others from the hosting OS. In the following, we dive into detail in each of those levels individually.

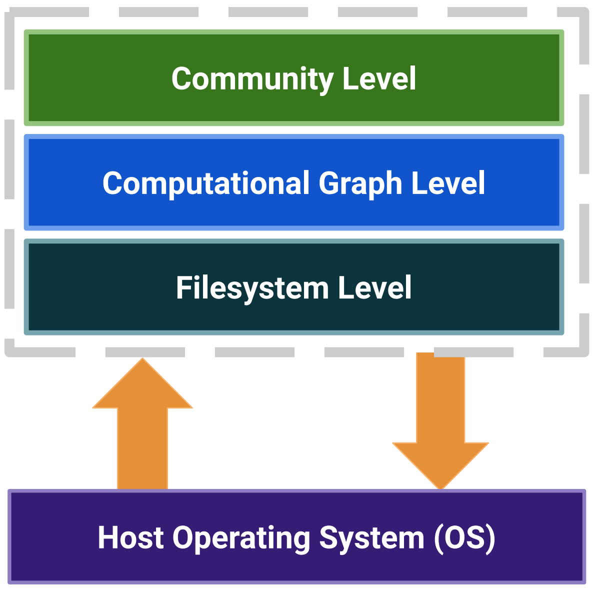


FIGURE 1.1: ROS functionalities level

1.2.1 Filesystem level

ROS develops some abstractions from the underlying operating system (OS). The Filesystem level represents the first tier in which lives the mechanisms to communicate at the application level with the host OS. As the name suggests, filesystem-level covers ROS physical resources that we encounter on the host machine. In Figure 1.2, we see how the filesystem level structures its internal components which define ROS' entities in the system.

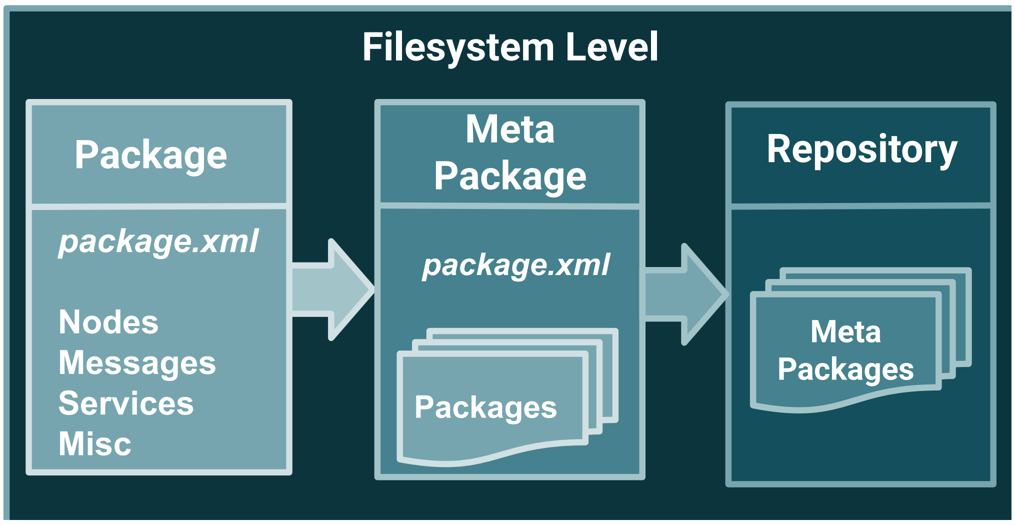


FIGURE 1.2: Filesystem level

In detail it defines:

- **Packages:** Packages are the main unit for organizing software in ROS. A package may contain ROS runtime processes (nodes), a ROS-dependent library, datasets, configuration files, or anything else that is usefully organized together.
- **Metapackages:** specialized Packages which only serve to represent a group of related other packages.
- **Package Manifests:** Manifests (package.xml) provide metadata about a package, including its name, version, description, license information, dependencies, and other meta information like exported packages.
- **Repositories:** A collection of packages that share a common VCS system. Packages which share a VCS share the same version and can be released together. Repositories can also contain only one package.
- **Message types:** Message descriptions, define the data structures for messages sent in ROS.
- **Service types:** Service descriptions, define the request and response data structures for services in ROS.

By means of this mechanisms, we define the components that we find in the Computation Graph. In particular, this layer gives to developers a structured workspace to build upon with the toolchain provided by the framework similar to *Cmake* or *setuptools* for python, namely *catkin*[3] and *colcon*[4].

1.2.2 Computation Graph level

On top of the high-level layer provided by the filesystem level, the computation graph represents the core component of a robot. For the sake of our analysis, this is the most important component in which we specialize as well. This is, in fact, the peer-to-peer network of ROS processes in which all the communications/actions happens. 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 as depicted in Figure 1.3.

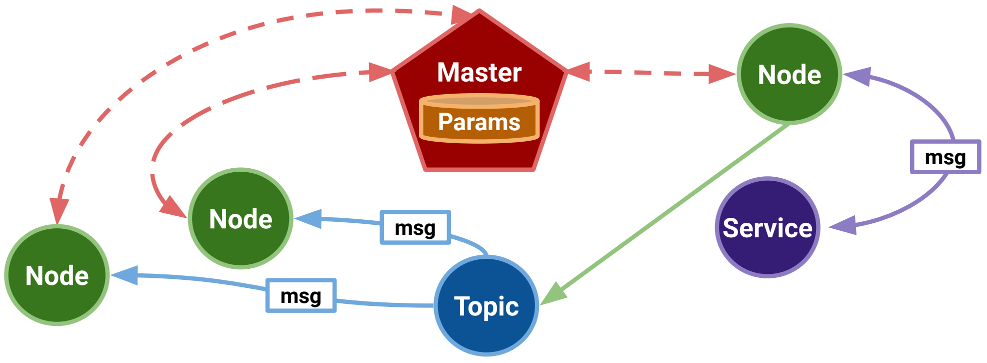


FIGURE 1.3: Computational Graph Level

Let's review 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. We can see see the master as 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 though we need to consider this as a logical independent component, currently it is embedded in the Master.

[3]https://docs.ros.org/en/api/catkin/html/
[4]https://colcon.readthedocs.io/en/released/index.html

- **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 a 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. Additionally, 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-to-one` transport it's more appropriate a request/reply interactions. This kind of communication is 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.
- **Actions:** those are the simplest communication mechanisms in ROS, based on client-server interaction, a client can invoke an action from a server and wait for the results of the specifically provided computation. Actions are usually deployed for external agents or resources.

Communications in the graph are managed by the Master, which acts as a nameservice, to topic and service registration as well as node addresses. Therefore, each node has to communicate with the Master, to receive information about other registered nodes, and establish connections. The Master is also in charge of callbacks in case of network activities, which allows nodes to dynamically create connections as new nodes appear.

In order to establish a connection, a node has to query the Master and retrieve the address of the requested node (if present). Therefore, if a node wants to subscribe to a topic, it needs to send a request connection to the node that publishes that topic and establish a connection over an agreed-upon connection protocol either TCP or UDP. The most common protocol used in a ROS is `TCPROS`, which uses standard TCP/IP sockets.

However, considering the agnostic architecture of ROS, in order to unambiguously address a specific node in the graph we need to introduce the concept of *names*. Names are used for all the ROS's components: nodes, topics, services, and parameters all have names.

In detail, ROS provides a hierarchical naming structure that is used for all resources in a ROS computation graph. These names are very powerful in ROS and central to how larger and more complicated systems are composed in ROS.

Each resource is defined within a `namespace`, which it may share with many other resources. In general, entities (node) can create other resources (topic, service, etc) within their namespace - working by namespace isolation - and they can access resources within or above their own namespace. This encapsulation isolates different portions of the system from accidentally picking the wrong named resource via their Uniform Resource Identifier (URI).

Thanks to this abstraction we are indeed facilitating reusing code, since, from a developer perspective, it is enough to write the nodes that work together as if they are all in the top-level namespace. When these nodes are integrated into a larger system, they can be pushed down into a namespace that defines their collection of code.

In order to exemplify those concepts, we depict in Figure 1.4 a remote drone network in which we identify three different namespaces. To ease readability, nodes' topic have been omitted from the picture and are represented by the blue arrows. In ROS we can distinguish four types of Graph Resource Names: *base*, *relative*, *global*, and *private*.

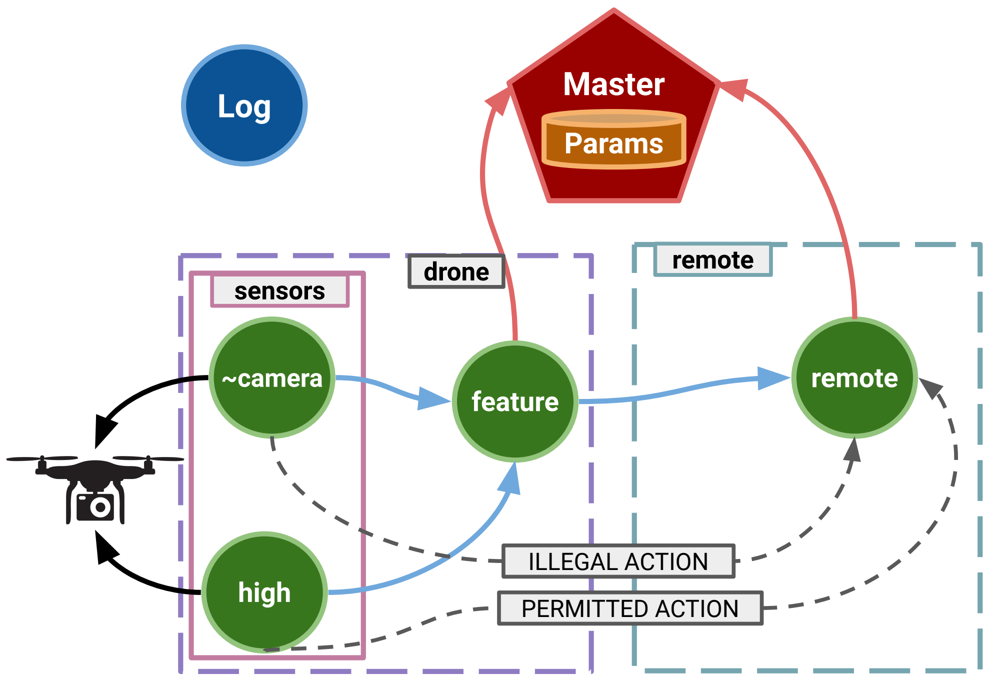


FIGURE 1.4: Remote Drone example network

Let's see below how they are defined:

- **Base:** this is addressed with the root URI "/", similarly to a Unix directory tree root address. In the figure above, in the Base namespace we find the Topic "/Log".
- **Relative:** inside a namespace (e.g. '/drone'), we can address a resource relatively as in Unix systems. Therefore, if we are already in the working namespace, specified earlier, and we have a subtree 'sensors' with a resource 'high' in it that publish the altitude of the drone, we can address it as 'sensors/high'.
- **Global:** from whenever namespace we address unambiguously a resource from the root. Therefore, if we are in namespace 'remote' and we want to access the

altitude sensor from the previous example we can access it at the URI '/drone/sensors/high'.

- **Private:** we can also specify private name that are addressed as relative name but are only visible from the internal namespace via the character '~' (i.e. '~/sensors/-camera/').

As anticipated in the example, by default, the resolution is done relative to the node's namespace. As we can see, from the 'remote' namespace we can access open/public resources as *high*, but not the private one as *camera*.

A developer needs to be careful with the usage of the base; these are global names and should be avoided as much as possible since they limit code portability of the implementation. As example, our drone network, can reuse the already implemented mechanisms `drone` and `remote` by simply swapping the elements in the */drone/sensors/* namespace to accomodate a different model of drone.

However, from an attacker's point of view, nodes in the root are particularly interesting since they offer global access and visibility to the network, therefore offering access to all the resources of the robot.

1.2.3 Community level

Lastly, in this level we find ROS resources that enable separate communities to exchange software and knowledge. It represents ROS's developer community that shares code structure and knowledge for the definition of more complex community-based systems.

These resources include:

- **Distributions:** ROS being a meta-operating system uses a versioning system stacks similar to Linux distributions. It offers straightforward installation mechanisms to a collection of software which is maintained consistent version across a set of scheduled release.

- **Repositories:** as introduced in Section 1.2.1, ROS relies on a federated network of code repositories, a bug ticketing system, where different institutions can develop and release their own robot software components.

- **Documentation:** The ROS Wiki is the main forum for documenting information about ROS. Additionally, tools as Blog, mailing list, ROS forum (discourse) are available for discussing the framework.

1.3 ROS API

To develop a robot system in ROS, we use the framework's meta-operating system API. As discussed, the communication between ROS entities is managed in the computational graph, in which we find a common and established set of subsystem APIs that are used

to string together ROS nodes into an interconnected computational graph. In particular, the API can be logically divided into three main categories: the master API, parameter server API, nodes API where we put particular emphasis on the slave API subset.
In detail, those are implemented via XML-RPC, a stateless HTTP-based remote procedure protocol. The reason why XML-RPC has been chosen, in 2007, instead of other HTTP/HTTPS protocols is mainly because its lightweight, no stateful connection requirements, and wide availability in a variety of programming languages. Still, this choice comes with several drawbacks including verbose encoding of application-level data, which results in greater overhead costs, and more notably the lack of any authenticated encryption or authorized remote execution.
In particular, we addressed these issues for Secure ROS (SROS) [21, 98]. Other research initiative worked towards similar goals [31]; in particular, introducing identification and authorization of clients, to enforce basic access control, which can be achieved using HTTPS security methods but are not supported natively by ROS.

To exemplify how this level works, and guide through the discussion, we leverage a simple publisher-subscriber network [5] depicted in Figure 1.5. The goal is to deploy two entities in the graph: a publisher (*talker*) - which will continually broadcast a message over a channel "*chatter*" located in the root namespace - and a subscriber (*listener*). In the computational graph, we identify two nodes, namely talker and listener, and introduce the API calls we discuss in the following.
The rest of this section details the intended use of the three API categories, additionally foreshadowing the potential vulnerability each called method may exhibit.

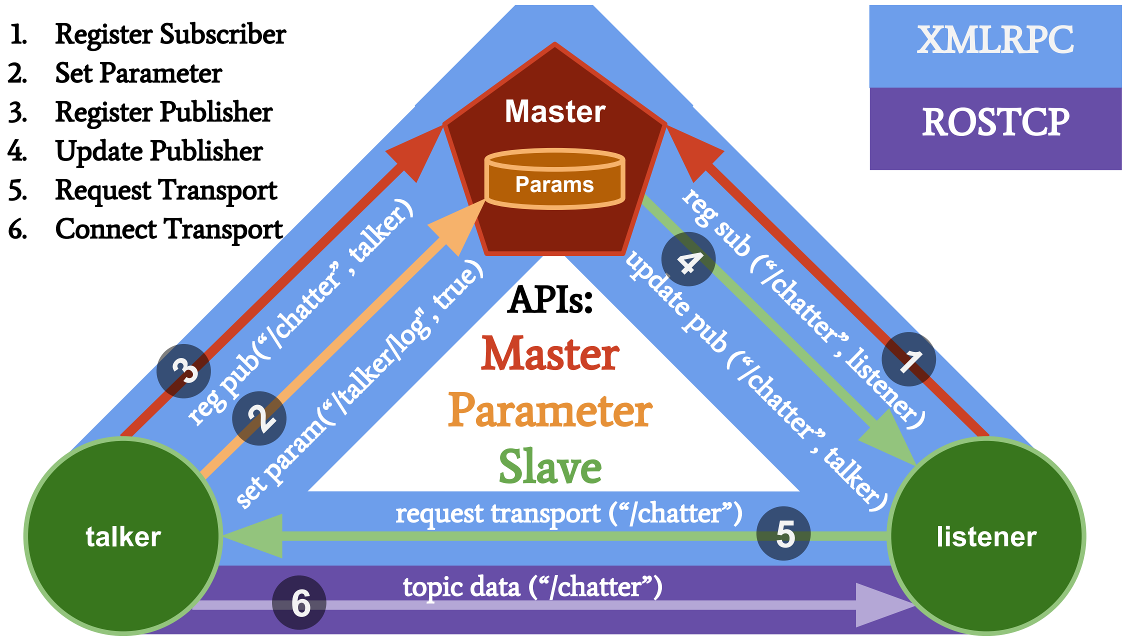


FIGURE 1.5: Publisher-Subscriber network

[5] http://wiki.ros.org/ROS/Tutorials/WritingPublisherSubscriber(python)

1.3.1 Master API

The master is the principal component in ROS. The Master API[6], exposes several functions to the other agents (aka nodes) via a standardized interface to connect to the master (server). In detail, it provides registration API for registering as publishers, as subscribers, and as service providers. Furthermore, it includes the mechanisms that allow the discovery of agents in the namespaces of the network. All in all, we can picture the master as a *trusted* entity in the network that is queried by the nodes via its address (URI). In fact, as said, each entity in the graph is qualified by a URI that corresponds to the `host:port` XML-RPC address of the entity in the server.

In the following we analyze in detail each of those API:

Register/Unregister - Subscriber, Publisher, Service

This group of calls makes use of the caller ID, API URI of the node, and the namespace/datatype for subsystem registration. It is important to notice that identity derives completely via the call parameters provided and it is never necessarily proven, neither through the context of the socket connection or otherwise, enabling trivial spoofing of registration requests.

Lookup - Node, Service

This group of calls is used to lookup the URIs for nodes given a node ID or service given service name, enabling the resolution for the URI location of namespaced nodes and services. Acquiring the URI for a target element in the graph is the starting point for many remote attacks; open oracle access to arbitrary disclosure of this information simplifies this process greatly.

Get - Master State/URI, Topic List/Type

For introspection analysis, the complete internal state of the Master can be retrieved, extracting as result the entire topology of the ROS system, i.e. all current publishers, subscribers, and services. This is used by debugging and live monitoring tools like rqt's node graph visualizer. Deeper topic introspection is also possible and is particularly useful for fingerprinting the system and ascertaining the necessary header information to spoof subsystem connection requests.

1.3.2 Parameter API

Like the Master, the Parameter Server API[7] is implemented via XML-RPC. It stores basic data as scalars, lists, and base64-encoded binary. It can also store dictionaries (i.e. structs), but these have a special meaning; in fact, dictionary-of-dictionary is the structure used

[6]wiki.ros.org/ROS/Master_API
[7]wiki.ros.org/ROS/Parameter Server API

for the representation of the namespace; in detail, each dictionary represents a level in the naming hierarchy (aka namespace), and the other dictionary represents the internal structure of that namespace 'branch'.

Let's consider an example: we want to set the value of a variable *foo* in the namespace */node/bar/*. In terms of internal structure in the parameter server the value `/node/bar` would be a dictionary `{foo:1}` and the value `/node/` would be a dictionary `{bar:{foo:1}}`.

In the following we analyze in detail each of those API:

Set, Get, Delete

These calls allow the caller to read and write parameter values into the key-value parameter storage. All anonymous agents are provided read and write permissions to the parameter database given that no ownership model is enforced, nor are any namespace restrictions retained. For example, every node registers a logging level parameter that may be used to silence or censor log activity.

Has, Search, List

Additional calls provide the ability to inquire into the parameter namespace tree, ranging from checking a target key, recursively searching the namespace hierarchy or a complete dump of instantiated keys. As said for the master, such calls are commonly employed by developer or user interface tools such as rqt's dynamic reconfigure to list node parameters into a front panel display. This is additionally useful for profiling or fingerprinting the purpose and capabilities of system components.

Subscribe, Unsubscribe

To synchronize local node parameters with those stored globally in the parameter server, nodes may subscribe to value change events for a given parameter key. These callbacks are initiated by the parameter server, where a temporary connection to the node's API is created upon each event. Given the socket connection is not continuous, unlike topics or actions, the parameter subsystems as with services are particularly exploitable using isolation attacks as detailed later.

1.3.3 Node API

The Node, in particular, has three types of APIs that expose it to the other agents:

1. **Slave API:** this kind of APIs are used by the node to receive callbacks from the master, and for negotiating connections with other nodes.

2. **Transport protocol API:** namely TCPROS and UDPROS. These methods are used by nodes as communications primitives. For example, if a node wants to establish a topic connection with another, it uses these protocols to negotiate a connection. The

difference between the two protocols is, trivially, the underlying mechanisms: TCP - persistent, stateful TCP/IP socket connections - in TCPROS and UDP in UDPROS.

3. **A command-line API:** lastly, based on the XML-RPC back-end, each node supports command-line remapping arguments, which enables names within a node to be configured at runtime and low-level XML-RPC calls.

As the other entities discussed above, every node has a URI, which corresponds to the `host:port`.

In particular, we are interested in the Slave API[8] which we analyze in detail:

Update - Publisher, Parameter

These methods serve as callbacks for the Master to notify subscribing nodes of changed topic publishers registered or to disseminate modified values of parameter keys. As most nodes merely register for such events, rather than requesting and subsequently parsing the entire system state from the Master API directly, these callbacks are the dominant mechanism for discovery and synchronization of data through the graph.

Request - Topic Transport Info

After a subscriber receives a publisher update callback, it will subsequently request the "topic info" by contacting the new publishers directly to negotiate an established means of transport, e.g. ROSTCP or ROSUDP. This phase also checks to ensure expected data types match via comparing message type checksums from the connection header. A separate socket port is relegated for the actual message transport, thus this handshake may be bypassed if the URI for transport is known a priori.

Get - Bus State/Info, Master URI, Pid, Subscription, Publications

For remote diagnostic purposes, additional system-level calls provide current statistics and meta info on active transport connections, configured Master URI, process identifier of the node on a relative host, as well the node's internal record of its own subscribed and published topics. These calls however reveal much in the way of the local graph topology without necessarily resorting to the Master API.

Shutdown

A particularly powerful call is the shutdown method that can be used to remotely self terminate the node process. This method is used by the Master when resolving node namespace conflicts, i.e nodes with duplicate fully qualified names, by conventionally killing the older node in favor of the newer. However, this method is not restricted to the Master and can be invoked by any client, e.g. from `rosnode kill`, permitting the

[8]wiki.ros.org/ROS/Slave_API

termination of ROS process without requiring the proper POSIX signal permissions in the target host.

1.4 Communications

In this section, we dive a little more into detail on how communication is handled in ROS. In general, when we talk about network communication, TCP is de facto the best choice since it provides a simple reliable communication stream: TCP packets always arrive in order, and lost packets are resent until they arrive. However, while these features are great for reliable wired networks, they become less efficient when the underlying network is a lossy WiFi or GSM connection. In this situation, UDP is more appropriate.

As discussed in the previous section, when we want to establish a connection, we need to negotiate it using the appropriate transport via XML-RPC. The result of the negotiation is that the two nodes are directly connected, with messages streaming from publisher to subscriber via ROS's proprietary implementations of TCPROS and UDPROS. This means that each transport has its own protocol that defines how the message data is exchanged. For example, using TCPROS, the negotiation would involve the publisher giving the subscriber the URI on which to call connect. The subscriber then creates a TCP/IP socket to the specified address and port. The nodes exchange a Connection Header that includes information like the MD5 sum of the message type and the name of the topic, and then the publisher begins sending serialized message data directly over the socket.

By leveraging on the previously discussed APIs, we can summarise the whole negotiating process as a two-step communication between the nodes and the master.

This process opens to some exploitation that leads to vulnerabilities; for example, if we already knew the URI of a node, it is possible to directly connect to a node without querying the Master. Following this overview, in the sub-sections below we will see how the connections are established for topics and services.

1.4.1 Establishing a topic connection

In the next table, the sequence by which two nodes begin exchanging messages in case of topic communication is depicted:

Operation	Protocol
+ Subscriber starts: reads its command-line remapping arguments to resolve which topic name it will use. + Publisher starts: reads its command-line remapping arguments to resolve which topic name it will use.	Remapping Arguments
+ Subscriber registers with the Master + Publisher registers with the Master. + Master informs Subscriber of new Publisher for the wanted topic. + Subscriber contacts Publisher to request a topic connection and negotiate the transport protocol. + Publisher sends Subscriber the settings for the selected transport protocol.	XML-RPC
+ Subscriber connects to Publisher using the selected transport protocol.	TCPROS/UDPROS

TABLE 1.1: Topic Connection

1.4.2 Establishing a service connection

On the other hand, services can be viewed as a simplified version of topics. The difference is that whereas topics can have many publishers, there can only be a single service provider. However, the master accepts as a service provider the most recent node to register with the master. An advantage of using services instead of topics is that the service client doesn't have to be a ROS node but it can also be an ad-hoc external application (i.e. a script).

Table 1.2 depicts the operation on services, according to the corresponding protocols.

Operation	Protocol
+ Service registers with Master	XML-RPC
+ Service client looks up service on the Master + Service client creates TCP/IP to the service + Service client and service exchange a Connection Header + Service client sends serialised request message + Service replies with serialised response message.	TCPROS/UDPROS

TABLE 1.2: Service Connection

1.5 Attacks on ROS

In this section, we explain some basic ways ROS applications can be manipulated by an attacker. In particular, knowing the workflow behind the communication mechanisms allows us to reveal some weaknesses that we demonstrate in practice on two novel software

[32], namely **ROSPenTo** and **Roschaos** which exploit the APIs discussed above at the master, node level, and parameter server. The idea behind those two CLIs is to demonstrate how an unmodified ROS client library can be exploited in case of attacks. The difference between the two is the attacker approach to the framework. In the first case, we pose as an external pentester while in the latter, by leveraging on the trusted nature of the framework - by extending the underlying libraries, e.g. rosgraph - we manipulate the topology and internal state of the computational graph from the host machine.

With no surprise, the kind of attacks that emerge from our analysis is common to several IoT/CPS frameworks. In particular, we can find some classical cybersecurity attacks from the literature such as:

- **Eavesdropping** (or Interception): the attacker "listens" to the flow of data, accessing information and therefore breaking *confidentiality*. Since in ROS a publisher sends everything as plaintext we are not able to avoid and block this kind of attack.

- **Modification**: the attacker intercepts the data and modifies it, breaking *integrity* of information. The attacker can put himself in the middle of the communication between a remote and the robot and change the content of the message

- **Forging or Impersonation**: the attacker introduces new information and lets the destination believe that the information is coming from an honest source. It breaks *authentication*. Similarly to the previous one, by design, ROS allows the presence of multiple publishers and subscribers for the same topic. This is tricky because it allows an attacker to steal the device without any knowledge from the administrators.

- **Repetition** (reply-attack): re-send a message that has been intercepted in the past. In this way, an attacker can build a payload that allows him to do everything he wants. In order to fix this kind of attack, a possible solution is the introduction of an 'approval' time window for the messages.

In the following we will discuss:

- **Inject Attack:** in this Forging attack we want to introduce new information inside the computational graph thus breaking Authentication.

- **Service Isolation:** this represent the first step for Modification attack. In this scenario, we want to isolate a resource from the other agents and get privileged access to it, while also isolate the service from the rest of the network.

- **Man in the middle:** in this classic attack we want to intercept and forge messages without being notices by the users that are communicating in the network by eavesdropping.

- **Parameter Attack:** in this attack we want to modify on the fly variables values to trick the victim with false data via Modification

- **Log and Monitoring Attack:** in this case we want to brake accountability in the network and disrupt the logging and monitoring systems in the robot via Modification

As a side note in addition to our analysis which focused on the computational graph only, more recently, a red team comprehensive analysis of ROS has been done by Alias Robotics [91]. In addition to similar attacks as the one described here, the researchers showed more advanced threats with a fairly simple threat model: they demonstrated how we could execute code remotely on the host machines; how unprotected robot endpoint could be easily used to pivot in the ROS network; exploitation of known vulnerabilities on the target robots. All in all, offering a good overview of the problems that afflict the field also in real-world industrial networks.

1.5.1 Inject Attack

In this base attack, our goal is to inject false data into a running ROS application. This attack would be useful to trick a decision point into performing unsafe action or fraud a monitoring system in autonomous devices. In order to do so, we want to mislead a ROS node into consuming data from a false publisher. In short, we have to isolate the subscriber from the publisher and force the connection to an unauthorized publisher. Interestingly, we can perform the attack without consulting the master, therefore without raising any telltale log in the network traffic. In detail, we leverage the trusted nature of the framework, and by means of the `publisherUpdate` call the attacker to isolate the subscriber and feed the victim with the new URI.

This is possible if we already know the URI of the subscriber and publisher, otherwise, we need to invoke the `getSystemState` to get the list of all the topics, call `lookupNode` to get the URI of the publisher, as well as the `requestTopic` call of the XML-RPC Slave API. Still, doing so would leave some logs in the master history. Therefore, if we want to be unnoticed we have to get the URI in a different manner via scanner tools (i.e. Wireshark).

1.5.2 Service Isolation Attack

Knowing how we can hijack a topic and inject arbitrary data, in this case, we want to tackle services. Similarly to what we have done above, an attacker uses the `getSystemState` to get a list with all the available services and providers, the `lookupService` to extract the URI of the target, and the `unregisterService` method to trick the master into removing the service from the graph. As previously said, if we already know the target URI we can skip directly to the unregistering request.

Once the service has been correctly unregistered, any other client in the network would not be able to retrieve the service server URI, which means that the service is not available anymore for regular ROS nodes in the network. Interestingly, no notice is sent to the service server to confirm the removal, therefore only the attacker will be able to send TCPROS-Header and a subsequent service request to the URI of the service. Still, we have to note that this attack can be detected by calling the `getSystemState` method since it would be clear that the service is not available anymore from the retrieved list.

1.5.3 Man in the middle attack

In order to perform a man in the middle attack, we leverage the capability we earned in the injection attack and the service isolation by exploiting in a similar manner the `actionlib`[9], the library that provides a standardized interface for interfacing with preemptable tasks. In detail, we want to break the action capability. The minimum agents that we have to consider in this scenario are two entities: an action client agent, and an action server. As shown in the picture below, in order to trigger or cancel an action, the client has to send a message respectively to the `goal` topic and `cancel` topic.

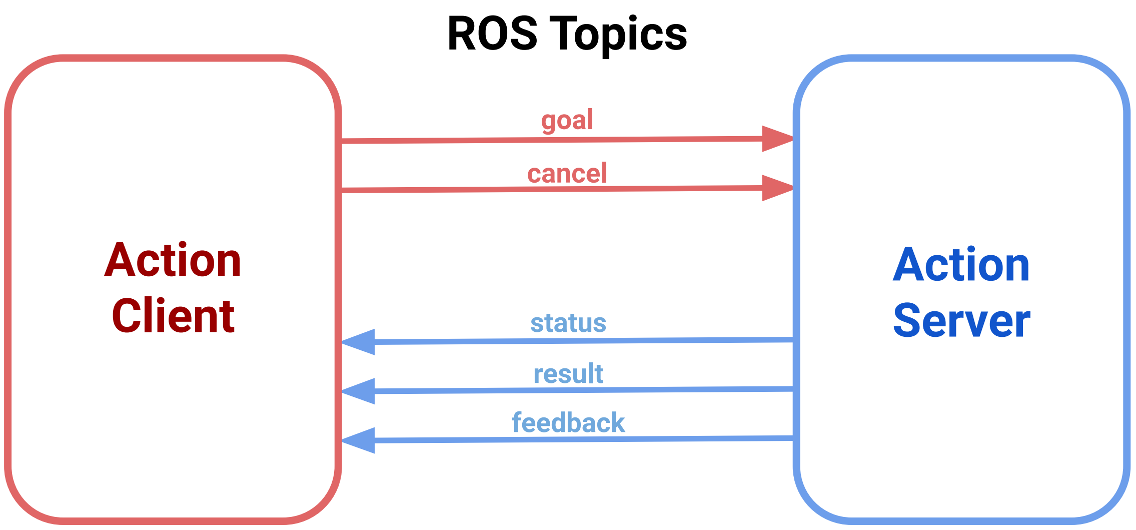


FIGURE 1.6: Action Interface from ROS wiki

Therefore, we can see how the service stack is actually implemented as two topics, namely goal and cancel. In order to attack the server, the malicious agent has to inject its own information into those topics. However, as an attacker, we are not able to prevent the server from sending feedback, result, and status messages to the client. Therefore, the client can still detect the attack.

We can solve the problem by running additional topics by covering: status, result, and feedback as well. Thus, the attacker can publish the messages the client would expect as a reply on its own messages sent on the topics goal and cancel. From the communication point of view, this scenario is just the multiple application of the previously described attack. The main challenge for the attacker is the context-sensitive knowledge required to pretend a reasonable behavior of a client and server in order to remain undetected. As said, this knowledge could be built via network analysis or prior investigation from the attacker. Apart from that, the attack itself is not visible in the ROS graph, and therefore it is as hard to detect as the injection attack.

[9]`http://wiki.ros.org/actionlib/DetailedDescription`

1.5.4 Malicious Parameter Update Attack

In this attack, we want to substitute a specific value in the parameter server with a malicious one. As discussed in 1.3, ROS Parameter API provides the capabilities of storing and accessing variables. In detail, we have the method `getParam` which gets the current parameter value as result, and `subscribeParam` that subscribe the agent to a specific parameter. When an agent subscribes to a variable, it stores the value locally and if the parameter value changes on the parameter server, the server calls the node's `paramUpdate` method; as a result, the local value in the node is updated.

Trivially, this attack exploits, once again, the trustability of the framework and invokes the node's `paramUpdate` method arbitrarily without interfering with the corresponding value on the parameter server.

Still, a more complex attack would involve the composition of some of the previous ones. In this case, if the attacker wants to gain control of the value of a variable in a specific node, it would be enough to:

1. Retrieve the URI of the victim, either via the `lookupNode` method or network scanner tools (i.e. Wireshark)
2. Then, the attacker can send the request of `unsubscribeParam` to the parameter server by crafting the request as coming from the victim, exploiting the fact that no message source verification is enforced in ROS
3. Since the victim is not aware of being unsubscribed, it is still waiting for `paramUpdate`. At this point, it would be enough to send the payload with the arbitrary value to the victim.

The scenario described here can also be applied to multiple nodes subscribing to the same parameter. In the worst case, every node sees a different value of the same parameter at a certain point of time, which can for instance lead to unpredictable behavior in a distributed ROS application.

1.5.5 Log and Monitoring Attack

Lastly, we discuss an attack in the area of logging and monitoring. In order to maintain logs, `roslog` is used to record and disseminate log events during runtime. In detail, we can configure the logging levels or verbosity of internal log handlers in each node process, which then writes the events to disk within the log file directory, as well as aggregates the mover unified topics for remote diagnostics.

As for other attacks, the modus operandi would be similar. We can either unregister the node from publishing on the default logging topic `rosout` or modifying on the fly the configuration to subverted the correct behavior of the system or disrupt an automatic trigger. Additionally, by being more selective in the sabotage actions further down in the data pipeline, we can easily create abnormalities such as delayed sensor data or expired transformations.

1.6 Countermeasures

To complete the review on ROS's attacks, in this section, we discuss some countermeasures. First, we could use `roswtf`[10]. The tool performs a ROS graph analysis and detects attack patterns as the one described above. However, it was developed to diagnose issues and not to be used as an intrusion detection or prevention system (IDS/IPS). In particular, it detects file-system issues and graph problems as unresponsive nodes, missing connections between nodes, or potential machine-configuration issues with `roslaunch`.

Second, some approaches in the past have been proposed to secure ROS. Starting from the work on Secure ROS (SROS) [94] and our work on it [98], an application-layer approach from Dieber *et al.* [31] as well as secure versions of the ROS core itself (such as Breiling *et al.*[19] or SRI's Secure ROS [11]).

Lastly, to overcome the limitation of ROS, ROS2[12] has been released and is currently under development. In particular, it is not susceptible to the attacks that affect the first iteration of the framework and integrates several new technologies and third party pluggable solutions that were not available when ROS1 was created. The underlying DDS communication technology [71] uses a different technique for discovery and works without a master (which is one of the main attack points for ROSPenTo and Roschaos). Besides, it supports security enhancements in the communication channels themselves [70]. Those security enhancements are made available to ROS2 via the SROS2 project[13]. An initial performance evaluation of security in ROS2 has been presented in [49].

1.7 ROS 2

ROS has been developed and is in use since 2007. We already discussed its flaws and one could think that those problems could be tackled and integrated into the existing core ROS code by following the examples given by the security propositions we saw in Section 1.6. However, several long-standing shortcomings and architectural limitations that cannot be rectified in a backwards-compatible manner affects the implementation of safety time-critical components in the framework. Although being considered, there is too much risk associated with changing the current ROS system that is relied upon by so many products and researchers [54]. To keep those devices working, and be unaffected by the new developments, ROS has been split into two parallel sets of packages that can be installed alongside and interoperate (e.g., through message bridges), namely: ROS1 and ROS2.

Through a complete refactoring, ROS2 aims to modernise and improve its core foundation components of distributed processing, anonymous publish/subscribe messaging,

[10]`http://wiki.ros.org/roswtf`
[11]`http://secure-ros.csl.sri.com/`
[12]`http://www.ros2.org/`
[13]`https://github.com/ros2/sros2`

RPC with feedback, etc, with support to a broad scope of projects. The final result will be a new ROS that still embraces the key concepts of ROS1 but will support new kinds of projects.

From the first iteration of ROS, we observe a huge evolution of IoT/robotics devices. This translates into new challenges and requests from developers and industry. By crawling on the developers' forum or by mining the repositories it is easy to spot the new trend of developers:

- **Teams of multiple robots:** although physically possible with the current implementation of ROS it is still tricky to build multi-robots upon the single-master structure
- **Small embedded platforms:** the limits of workstation-class computational resources on board is simply huge. In the IoT era, the focus needs to be put on bare-metal microcontrollers and/or microcomputers with limited performance or power constraints (i.e. raspberry pi)
- **Support for Real-time systems**: since ROS1 was not offering any real-time guarantee
- **Non-ideal networks:** design new solutions to increase the number of network types in which ROS can be deployed
- **Production environments:** develop tools to seamlessly transport research prototype to real-world application with ease
- **Prescribed patterns for building and structuring systems:** create new tools for features such as life cycle management and static configurations for deployment in the market

In addition to the evolution of the devices, the biggest advantage nowadays is that we can count also on a bigger number of open-source framework projects. Back in 2007, the OSRF developers had to develop from scratch the whole systems for discovery, message definition, serialization, and transport. However, with the introduction of software interfaces, it's possible to build a modular solution that allows the interoperability of external software for auxiliary functions that are non-robotics-specific. In practice, they demand to other projects as Zeroconf, Protocol Buffers, ZeroMQ, Redis, WebSockets and in general other Data distribution Service (DDS) implementations (e.g. Fast DDS, RTI Connext, etc), the task of developing auxiliary features.

Furthermore, through these external frameworks, OSRF developers can achieve some remarkable benefits. First of all, they need to maintain less code that is directly developed by other 'groups'; also, they can exploit an indirect advantage by horizontally included software framework that is the possibility of using features that have been introduced in the external framework and also build ad-hoc solutions for each project independently. For example, if the resources of a device are limited and the project doesn't need a certain feature, it's possible to get rid of the unnecessary software layer (downgrade to a 'core' version of ROS) and lightening the solution.

1.7.1 Structure of the middleware

To get a better understanding of how ROS2 works, here we dive more into detail about the structure of a middleware. We said that the new iteration of the framework maintains the same concepts of ROS1; this means that the client communication libraries will maintain the same publish/subscribe mechanisms used by the custom ROS protocols TCPROS and UDPROS.

However, to develop such mechanisms, despite the usage of different middleware solutions (e.g. DDS), we need to introduce an abstraction layer: **Middleware API**. This middleware interface defines the API between the ROS client library and any specific implementation.

In detail, as depicted in Figure 1.7, each implementation of the interface will have a thin *adapter* which maps the generic middleware interface to the specific API of the middleware implementation. However, for the sake of simplicity, apart from the scheme below, we will consider the adapter and the actual middleware implementation as a single component.

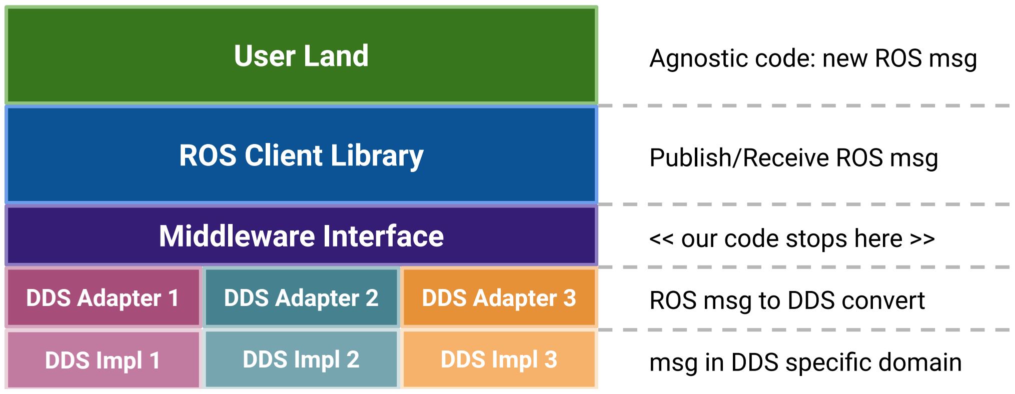


FIGURE 1.7: ROS 2 middleware interface

As such, the final solution will not expose any middleware implementation specifics to the developer that will work agnostically from the underlying middleware. In other terms, all DDS specific APIs and message definitions would be hidden from the user.

1.7.2 Data Distribution Service (DDS)

Thanks to this integration, we can see how DDS actually becomes an implementation detail of ROS2. In fact, thanks to DDS, ROS2 gets rid of ROS1's Master and its related issues, by inherent features as discovery, message definition, message serialization, and publish-subscribe transport.

In detail, the modifications with the integration with DDS are:

- **Discovery:** the entire discovery features previously provided by the Master are completely replaced by the DDS-based discovery system.
- **Publish-Subscribe Transport:** The DDSI-RTPS (DDS-Interoperability Real time Publish Subscribe) protocol replace ROS's TCPROS and UDPROS wire protocols for publish/subscribe.
- **Messages:** to maintain the current ROS' semantic contents that have evolved over years of usage in the robotic community, a message in ROS2 preserves the ROS1 like message definitions and in-memory representation. The idea behind the integration with DDS is of converting a message field-by-field into another object type for each call to publish. Interestingly, experimentation has shown that the cost of this copy is insignificant when compared to the cost of serialization.
- **Services and Actions:** DDS currently does not have a ratified or implemented standard for request-response style RPC which could be used to implement the concept of services in ROS. For now, ROS 2 is implemented on top of DDS' publish-subscribe mechanisms.

Despite being middleware agnostic, ROS2 advanced users still can access the underlying DDS implementation for extreme use cases or when in need of integrate with other existing DDS systems; still, this feature could limit the portability of the implemented solution to a single DDS vendor. To understand why some low-level DDS features can be still access from ROS2, it's important to notice that the OMG defined the DDS specification with several companies that are now the main DDS vendors [14]. Despite being part of a standard protocol, each vendor has specialized on a different level of the protocols. RTI's Connext DDS offers the full implementation of DDS-Security specification [41]; eProsima Fast RTPS provides an implementation with more direct access to the DDS wire protocol RTPS; TwinOaks's CoreDX DDS instead offers minimal implementation for embedded devices, etc.
From our perspective, DDS Security features are perhaps the most interesting addition to the framework. Thanks to the definition of so-called **Service Plugin Interface** (SPI) architecture, DDS can enforce the security features we tried to introduced in SROS but in a completely decentralized fashion.

In detail, the five SPIs are defined as follows:

- **Authentication:** the central authentication plugin is pivotal in the architecture, as it provides different authentication schemes and a Public Key Infrastructure (PKI) on x.509 certificate.

[14]http://dds-directory.omg.org/vendor/list.htm

- **Access control:** this plugin defines and enfoces restristion on resource access in DDS domain. For each participant (agent) it requires two signed XML files: a **Governance** file which specifies the domain to secure, and a **Permissions** file containing the policies of the agent bound to the name of the agent (defined by the Authentication plugin via x.509 certificate)
- **Cryptographic:** this plugin is in charge of all the cryptography-related operations of encryption, decryption, signing, hashing, etc.
- **Logging:** is in charge of logging the entire DDS domain events.
- **Data tagging:** provides the capability of add a tag to data sample in the network

At the time of writing, ROS2 supports only the first three SPIs. Since, as mentioned, not all DDS' vendors are implemented the entire security stack. Still, the security integration of Access Control Policies and Security Enclaves as we discuss in the next chapters in this thesis are critical for IoT networks.

1.7.3 ROS1 vs ROS2 Architecture

Lastly, in this section, we discuss in more detail the differences between ROS1 and ROS2 architectures. ROS2 as a direct evolution of ROS1, evolve the concepts originally designed in ROS1 by leveraging on the abundant off-the-shelf features that third party software provides. In Figure 1.8, we compare the architectures of the two frameworks:

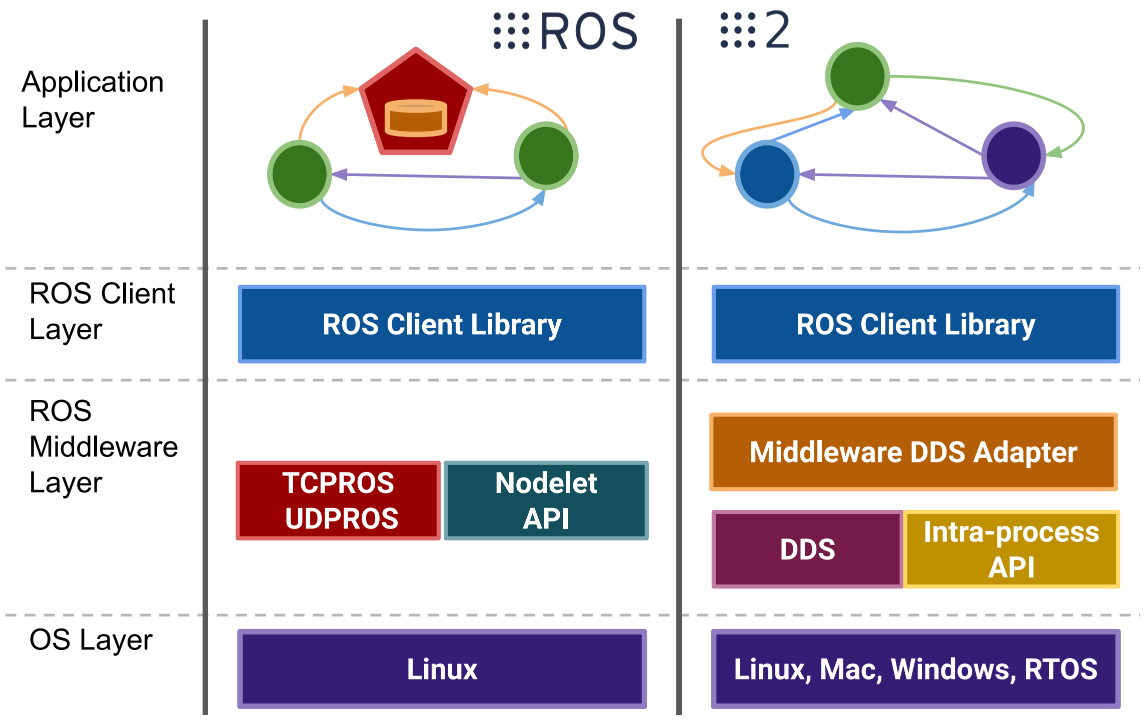


FIGURE 1.8: ROS 1 and ROS 2 Architecture comparison

As we can see, the enhancements affect almost the entire core infrastructure by the mantra of improve without revolutionize the framework:

- **OS Layer:** ROS originally supports only Linux-based OS. ROS2 overcomes this limitation and provides more development platforms to users such as Linux, Mac, Windows, and RTOS. Moreover, those additional operating systems' support, led to the second development, such as ROS' on cloud development (e.g AWS RoboMaker[15]), and numerous new integrated development environments (IDEs)[16].
- **Middleware Layer:** the evolution from the proprietary protocols TCPROS/UDPROS to modern and better-performing solutions, is paramount to solve the technical debt (design/code debt) of the old code base of ROS1. Moreover, the improved intra-process mechanisms in ROS2 are well optimized for modern applications.
- **Client Layer:** to maintain continuity in the development process, and ease interoperability between ROS1 and ROS2 applications, this level is virtually the same across the two ROS' version. Still, it have been rewritten to accommodate the different Middleware Layer.
- **Application Layer:** this layer better represent the huge differences between the two iterations. In the one hand, we have the Master that manages all the connections, on the other, the peer-to-peer network allow to define more secure and agile solutions.

Lastly, in the following table we highlight the differences, and main features, that ROS gained in this new iteration:

Feature	**ROS 1**	**ROS 2**
Number of Robot	Single	Multiple
Target Platform	Workstation-class	Small embedded platforms
Realtime support	No	Yes
Connectivity	Excellent networks	Non-ideal networks
Target Applications	Research and academia	Industry
Prescribed Dev. Patterns	No	Yes

1.8 Conclusions

In this chapter, we have presented ROS, discussed the key concepts of the framework, and how its APIs work in regards to the internal network (computational graph). We showed how ROS can be manipulated over the XML-RPC API and introduced the other possible threats that affect the framework. This also justifies why ROS has been considered as the base of our research, despite early assessment that has shown severe vulnerabilities and potential for manipulation [59].

[15]https://aws.amazon.com/it/robomaker/
[16]http://wiki.ros.org/IDEs

We discussed the clues behind the novel analysis tools ROSPenTo and Roschaos, highlighting how those may help in identifying possible threats and introduce some ideas about possible countermeasures, leaving aside best practices in ROS development that are out of scope at this stage and, if not respected, are not considered as vulnerabilities from this analysis but device-specific bugs.

In the last part of the chapter, we dived more into the evolution of the framework in its second iteration, namely ROS2. Still, despite it being under development and showed improvements over ROS1, as discussed in [91], from their experience, in the industrial scene robots that use ROS2 are years from being widely deployed. In fact, despite the announcement of ROS2 was in 2016 and a stable version has been released in 2017, the list of robots running it, at the time of writing, is very small and mainly composed of hardware experimental platform and developer kits [76]. Meanwhile, ROS1 still prevails, giving reasons for the agnostic works we present in this thesis and the push we want to provide into applying the core pillars of cybersecurity to the field.

The similarities with other CPS framework and the fact that has been used mainly in research and closed facilities, like home automation, service, and industrial robots (usually operating behind firewall or VPN), gave the users a sensation of security, demonstrated false by the fact that there are quite some vulnerabilities documented in the literature and are highlighted by the fact that there are already some instances of ROS running openly accessible via the internet [29].

Chapter 2

Access Control Policies

In this chapter, we discuss access control methods we develop focusing on techniques to enforce Authentication and Authorization properties in a distributed robotic network.
In particular, our goal is to develop solutions that could easily scale on large-scale deployments of connected smart devices, in which we observed a tremendous grown in the amount of personal data that are stored and processed every day. Considering the sensitive nature of this information there is a widespread suspicion concerning how these flows into the infrastructures. Especially with the increase of large interconnected networks connecting smart cyber-physical devices forming the so-called Internet of Things (IoT). Such solutions apply in different domains such as smart-grids, intelligent transport systems (ITS), smart homes, smart cities, public health, agriculture, etc. In further detail, we decided to develop our solutions in the **Application-level layer**. Indeed, our focus is on the logical approach of the problem, namely on the data-centric analysis of the infrastructure.

However, due to the intrinsic limitations of those devices, either from the power consumption point of view and the actual computational power at our disposal, engineering is not trivial. Traditional authentication schemes that are already in use in other applications, are infeasible and not applicable here due to the necessary software-components overhead and their resource consumption.

Additionally, since we want to design solutions that could be applied to a wider scope of devices it's safe to assume that we want to keep as real-time performance as possible while retaining the smallest footprint on the final device.
Therefore, with those premises, a suitable research direction is to develop static solutions - that could exploit a prior offline computational power - for the definition and distribution of access policies. By means of such techniques, the effort to secure the system is shifted to a third party, and avoid the consumption or waste of resources for real-time dynamic solutions that could burden the network, such as intrusion detection or prevention systems, and overall, degrade the device's performance.

Indeed, the direction we followed on the development of SROS [98] was to embed the access control policy as extensions of the x.509 certificate. The implementation was straightforward since we were already using the x.509 certificates to enforce confidentiality and authenticity respectively to encrypt the communication channels via TLS/SSL and to identify the agents in the network.

Still, the offered solution was far from being ideal, since piggybacking access policies as an extension of the identity certificate came with some drawbacks. To understand which are those limits, how we tackled them in more detail, and the reasoning behind the different choices, in this chapter we give an introduction to the field and discuss novel approaches that we present in the field.

The rest of this chapter is structured as follows:

- **Background:** in this section we introduce the reader to access control's issues and properties specific to the research field to provide the base concepts for the next discussions.
- **Distribution Architectures:** in this section we analyze in more detail authentication schemes. We present the properties and an overview of the different schemes available.
- **Access Control Policy:** in this section we present our novel static and dynamic solution for certificate and attribute distribution, namely user-pull and server-pull.
- **Procedurally Provision Access Policy:** in this section we propose ComArmor and Keymint; our novel toolchain specific for robotic middleware to mitigate the risks of improper provisioning.
- **Conclusions:** in this section we summarize what we have presented in the chapter and discuss some of the open issues in the field.

The work discussed in this chapter has been published in [22] and [97].

2.1 Background

In the following, we introduce which service and specific security issues arise in the Application-layer.
In detail, it contains the core logic of the system and the high-level protocols in charge of message passing as Message Queuing Telemetry Transport (MQTT) from the Organization for the Advancement of Structured Information Standards (OASIS), Data Distribution Service (DDS) from the Object Management Group (OMG), Constrained Application Protocol (CoAP), TCPROS and UDPROS from OSRF, etc.

In particular, we can identify some of the core security issues such as:

- **Data Privacy and Identity (Confidentiality):** the enforcement of a suitable encryption level in heterogeneous networks that supports legacy devices is problematic. Often, forcing the connection to abide by insecure legacy protocols is an easy attack vector (e.g. Bluetooth protocol attacks [56, 47]). Moreover, in cross-platform networks, managing identities and Quality Of Service (QoS), often leaves unintended backdoor.

- **Data Access and Authentication (Integrity):** managing access to data in a complex network of applications, that spans a great number of users can be challenging. In particular, malicious users could exploit lax permissions policies or perform undercover actions by exploiting authentication loopholes issues.
- **Availability:** those networks often have to maintain availability as a pivotal feature of the system in real-time components (i.e. 3D stabilizers in a humanoid robot, Anti-lock Braking System (ABS) in automotive, etc). A huge quantity of data inside this level often has a big impact on the availability of the service.

The requirements to alleviate those problems are, indeed, authentication and authorization mechanisms. The remaining issue, now, is to identify how those components should be applied.

Authentication Schemes

As said, applying solutions that are not specifically designed for robotic networks, underestimating the impact that those might have on a system, could be disastrous.
To get a better understanding, and identifying a common set of comparison mechanisms we need to distinguish some key components of authentication schemes. As discussed by the authors in [68], to evaluate them, we can identify six criteria that define the taxonomy of an authentication mechanism:

1. **Authentication Factor:** this criteria specify the way two entities identify in the network, which can be either **Identity-based** or **Context-based**. In the first case, we provide a hash or a symmetric/asymmetric key (i.e. x509 certificate). In the latter, we either provide a `physical` feature such as fingerprint, retina scan, etc; or `behavioral` as in keystroke dynamics, gait analysis, etc.
2. **Tokens:** in this criterion we identify if the user authentication mechanism works on a previously generated identification token or via credentials (user/pass).
3. **Authentication Procedure:** this criterion specify how two agents authenticate within a communication:
   - `One-way Authentication:` one agent identify itself to the other, while the latter remains unauthenticated
   - `Two-way Authentication:` mutual authentication, both agents authenticate each other
   - `Three-way Authentication:` a central authority authenticates the two parties as guarantee
4. **Authentication Architecture:** this criterion defines if the agents authenticate either in a **Distributed** or **Centralized** fashion. Where the difference is whether or not the agents need a centralized authority to authenticate the communication

5. **IoT Layer:** this value address which level of the Three-layer architecture we are analyzing, either: Application, Network, Perception.

6. **Hardware based:** here we address all the hardware components that could be employed to the authentication, such as True Random Number Generator (TRNG), Trusted Platform Module (TPM), or specific component for key-storing or key-generation.

2.2 Distribution Architectures

In this section, we analyze distribution architecture models by discussing a novel approach we propose [22]. In detail, we discuss about Identity-based application in which agents authenticate through x.509 certificates; authentication, depending on which setup the user deploys in the network, is managed via a one-way procedure and, either in distributed or centralized architecture.

With the specific goals of keeping the resource usage under control, 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 [74, 73], we propose two different architectures for IoT network: **User-pull** and **Server-pull**.

By applying our solution of smart attribute certificates we can easily enhance how users define how agents communicate in the network.

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 the application level, the disclosure of sensitive information and blocks unauthorized agents by applying a priori access control model.

To better understand how we develop the proposed solution, it's important to have a grasp of the framework that has been selected. Therefore, we need to briefly evaluate the general concept behind SROS.

SROS is a set of security enhancements for ROS that aims to secure ROS API and ecosystem using native TLS/SSL support for all IP/socket level communication. Also, with the usage of x.509 certificates, defined chains of trust employing 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, SROS defines 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 Public Key Infrastructure (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

By embedding policies as plaintext in the extension field of the x.509 certificate, we expose the device to privacy issues related to the easy reconstruction of the topology of the network via passive sniffing attack.
To alleviate those problems, we propose a family of static and dynamic architecture models for certificate distribution and attribute verification. The general idea behind the proposition is to exploit the x.509 PKI infrastructure and deploy specific certificate authorities (CAs) with different authoritative characteristics.

2.2.1 User-pull Architecture

As the name suggests, in this architecture, every new agent that enters the network it has to query the certification authority, extract its own certificate bundle and archive it locally. When it needs to authenticate to someone else, it will trivially share the public certificate with the receiver in the network.
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 (CAs) which are `trusted` entities in the system. In detail, in this category, we find the previously mentioned implementation in which we store the profiles as extensions of the x.509 certificate. However, to solve the privacy issues highlighted, we introduce an additional type of certificate defined in the x.509 standard and decouple the agent information into: **Identity Certificates** (ID) and **Attributes Certificates** (AC)[35].

On the one hand, identity certificate is the most common certificate that acts as an identity card and binds a set of attributes (e.g. name, organization, etc) and a public key; on the other hand, attributes certificate 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's holder.

The advantages of using AC for authentication in place of trivially embedding the policies as extensions are (i) identities often do not have the same lifetime as access policies. Therefore, if we have to grant or revoke access to a resource we need to revoke the existing certificate and issue a new one, actually shortening the whole certificate lifetime; (ii) to avoid threats and reduce the surface attack, the authoritativeness of the certificate authorities are separated between attribute CA and identity CA. The first can only issue attribute certificates (usually domain-specific), while the latter issues identity proofs with broader scopes that could potentially spans in cross-domain applications (e.g. passport).

There are several ways in which we can bound the authorization with the identity certificate. Below we present three different approaches: **Monolithic**, **Autonomic**, and **Chained signatures**.

Monolithic

Monolithic is the first hands-on approach and the easiest to deploy. By means of a single certificate authority, we create a certificate that holds the identity and attributes information. To do so, we piggyback the attributes in the extension fields of the agent's x.509 certificate.

The resulting certificate, depicted in Figure 2.1, tightly couples the identity and attribute information in a single signed artifact. Therefore, if we have to make any change either on the identity or, more likely, to the access policy rules, we must revoke the certificate and issue a new updated one.

On the plus side, the management of this particular solution simplifies the decisional and operational chain since we need to trust a single CA and check only one Certificate Revocation List (CRL). Still, this approach has several drawbacks: (i) multiple CAs are very difficult to orchestrate. We can't revoke a certificate if we aren't the issuer CA and there is the possibility of issuing multiple certificates with different attributes for the same agent exposing the device to exploitation (ii) due to the design of the solution we are not able to maintain different life-time for multiple different attributes; in fact, all the attributes share the same lifetime of the certificate.

All in all, the monolithic approach is the most favorable solution if we are limited in terms of resources or if we are looking for a statical solution, even though we sacrifice flexibility in maintainability and agent privacy.

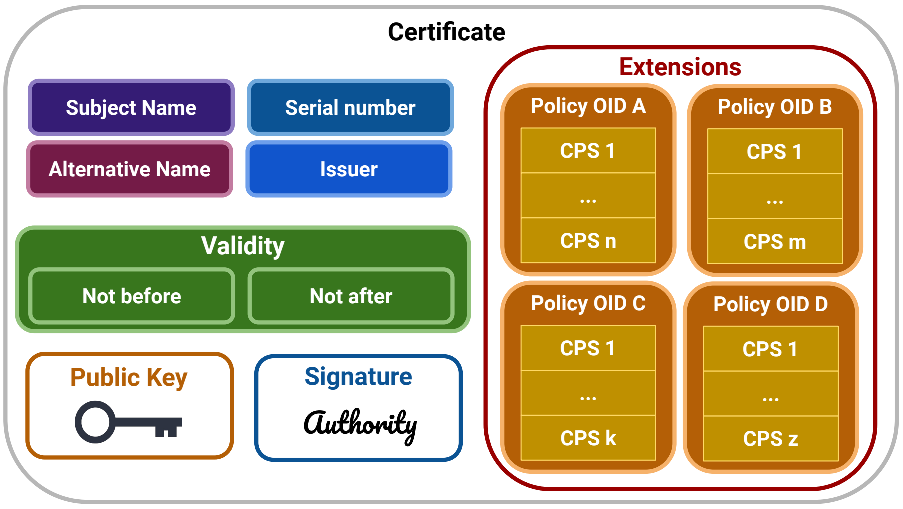


FIGURE 2.1: Monolithic certificate

Autonomic

In Autonomic we evolve the solution and support multiple CAs by differentiating between identity and attribute certificates. This means that the resulting cryptography artifacts of identity and attribute are going to be loosely coupled.

This solution allows the existence of multiple AC certificates per agent provided that there is an injective function from the certificates to the agent ID; which implies 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 subset of information from the identity as subject's name, public-key, certificate serial number, etc.

Depicted in Figure 2.2, Attribute certificate B is bonded to both ID certificates represented by the *alternative name* property, while certificate A is bonded to certificate X only by *name* and *issuer* properties.

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 such, if we choose to bind the attribute certificate to the unique value of an agent's identity certificate, 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 represented by a careless choice of the binding set. In fact, even though we have an injective function from the certificates to the agent there aren't constraints on the information that are stored in the certificates. Thus, if we accidentally choose a common set of information, the same attribute certificate can be used by unauthorized agents that share the same information with an authorized agent.

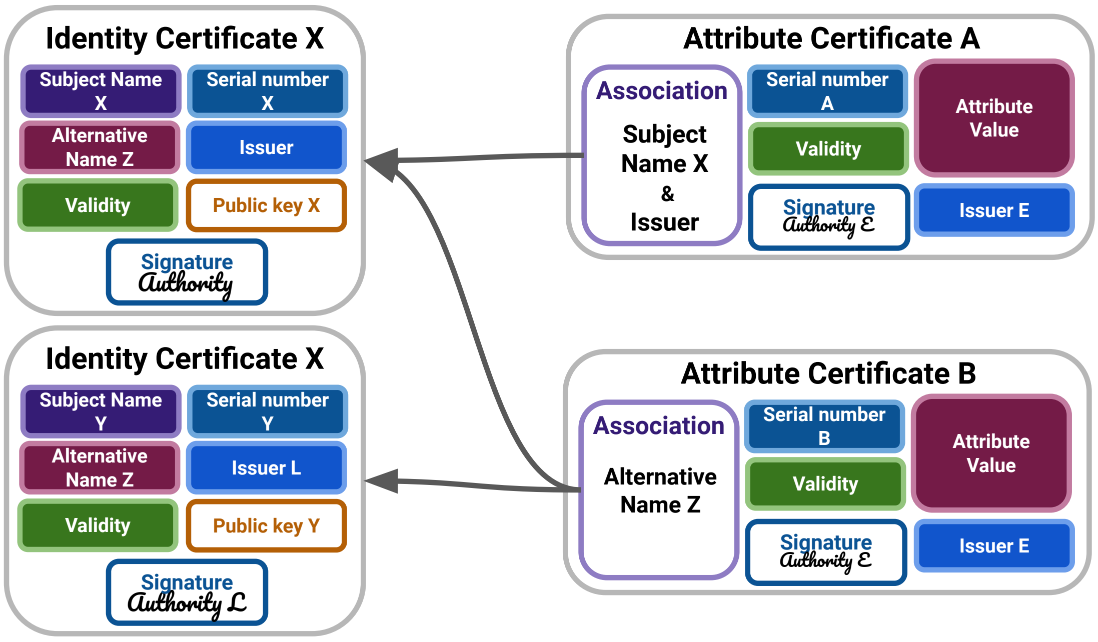


FIGURE 2.2: Autonomic certificate

Chained-Signature

Lastly, chained-signature aims to combine the bests from the previous two solutions by granting the security features of monolithic and the flexibility of the autonomic approach.

Like in the previous case, also in the chained signature approach an agent can have multiple AC 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 as depicted in Figure 2.3. In fact, if the information in the referenced ID certificate is changed (a new certificate is issued), the digital signature must change as well. Still, 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 advantages 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 to share with other agents only the necessary information. In fact, with a monolithic certificate, all the policies of the agent are available in a single certificate, instead in this way we can share only the necessarily attribute certificate enforcing a new privacy feature and dynamic management of profiles.

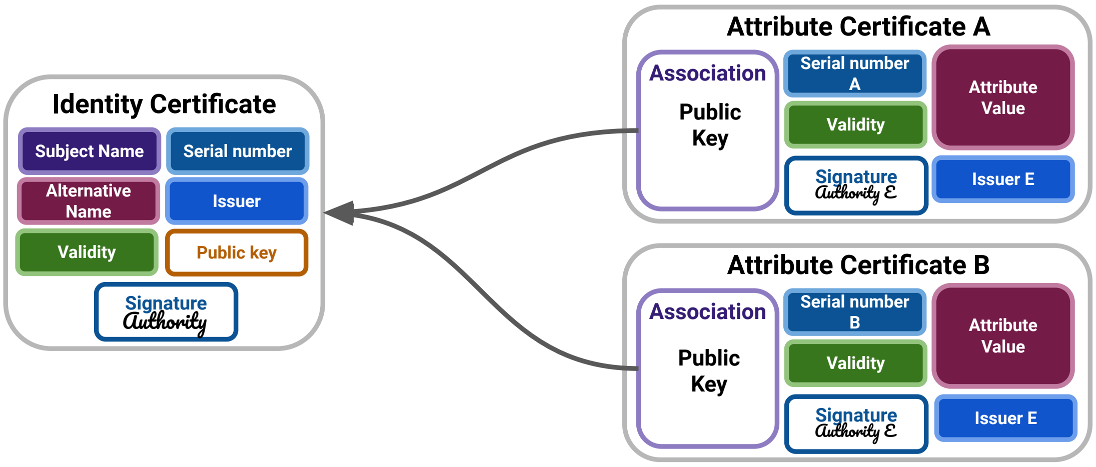


FIGURE 2.3: Chained certificate

2.2.2 Server-pull Architecture

Contrary to User-pull, in this architecture, we demand the authentication phase to the attribute authority (AA). The goal is to define a dynamic solution in which apart from the AA no one needs to know the attribute information maximizing privacy in the network. However, as in the 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 to 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 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). Considering that all the agents need to query the attribute authority to receive a response about the permissions if it fails the entire system will stop working. In fact, from the attacker's point of view, it will be enough to tamper with the AA to compromise the entire system. Besides, we introduce by design an overhead in the handshaking process; in fact, what was a straightforward local check of the profile in the user-pull model, became a remote request to the external attribute authority. However, we recognize that this problem could be slightly mitigated by resuming previous sessions via `Session Ticket` as introduced in TLS 1.3 [40].

2.3 Access Control Policy

In the previous section we discussed how we can securely share access control policies in the network. However, access control remains one of the most critical points in today's Information Technology [7]. Here, we provide a basic understanding of access control policies and their properties and characteristics. In particular, access control policies specify which subject (e.g. user, process, application) can access which resource or object (e.g. files) for performing which actions or operations. Additionally, the access control model defines the relationships between these entities [36]. There exist many access control mechanisms which are applied in many different systems, such as operating system, database management system, network, etc. Bertino *et al.* [14] gave an in-deep analysis of all the different kinds of the existing models and their pros and cons.

In particular, this thesis focuses primarily on the eXtensible Access Control Markup Language (XACML) [83] policy language which is the most used in robotic IoT applications. XACML is an XML-based language model from OASIS in which a policy is organized according to the previously introduced four elements: subject, resource, action, and environment. Where the subject specifies the entity requesting access with a particular action on a resource within an environment; the policy specifies if the request is allowed or denied.

As a high-level concept, we can consider an access control policy as directives stemming from the managing party to managed parties.

In particular, these directives can be differentiated based on their type:

- **Constraint-based:** these are the most common policy type we use, where the actions to which the managed party is subjected are explicitly listed. The actions are either **allowed**, **denied**, or **obligatory**.
- **Goal-based:** in this type of policy we find specific goals that the managed party must achieve. For example, maintain a minimum temperature or respects a deadline.
- **Utility-based:** those specify which value function needs to be respected to achieve the best outcome. For example, minimizing resource consumption.

In our scenario, all the policies are constraint-based since it gives a better representation in *Discretionary Access Control* (DAC) and *Mandatory Access Control* (MAC) schemas.

However, to specify an appropriate policy, a crucial part of the work is represented by proving the correctness or quality of the policy. Assessment of policy quality, as introduced by Bertino *et al.* [15, 16], can be stated as the problem of verifying that the set of policies are compliant with the following properties:

- **Consistency:** this property refers to verifying that the policy sets don't include policies that contradict each other. In this case the concept of access policies doubles as ensuring that the policy sets don't include a policy allowing and one denying the same request, as inconsistent policies lead to conflicts at policy enforcement.
- **Minimality:** this property holds when the sets of policies don't include redundant policies. As for redundancy increase the administration effort to manage and amend the whole policy.
- **Relevancy:** this property requires that the policy sets do not contain rules that do not apply in the application. In fact, irrelevant policies may lead to loopholes and security vulnerabilities.
- **Completeness:** this property refers to verifying that all the actions executed in the policy sets are within the network resources. Moreover, all the resources have to be covered by at least an action.
- **Correctness:** this property holds when all the policies are free of errors and compliant with their intended goal. Ensuring this property requires validation of the syntax and testing all the possible application's contexts and scenarios.

Still, the task of properly generate, maintain, and distribute the number of signed public certificates, ciphered private keys, and access control documents attributed to every identity within the distributed network can prove beyond tedious and be error-prone. In fact, the additional complexity and scalability of IoT networks make the secure orchestration of those systems a demanding process. In the next section, we tackle these issues introducing a novel toolchain to the provision of robotic middleware credentials.

2.4 Procedurally Provision Access Policy

To mitigate the risks of improper provisioning, in this section, we contribute a set of novel tools to provide users with an automated approach for the systematic generation and verification of necessary cryptographic artifacts in a familiar, yet extendable, meta-build system layout via workspaces and plugins [97].

We seek to mitigate the risks imposed from improper provisioning of robotic middleware credentials that could otherwise compromise system security. To achieve this, we provide the provisioning process of all transport artifacts via build automation. Such compilation is made possible by defining an intermediate representation to express the higher-level semantics of general permission policies, enabling the compiler to abstract away lower-level cryptographic operations. This approach also affords administrators to design policy profiles that are agnostic to the deployed transport, facilitating further consistency of security permissions across transport type, version, and vendor.

In fact, by leveraging on the network computational graph representation we can depict the network as an agnostic bipartite graph which allows transferring the aforementioned policy across different robotic network middleware.

Our approach contributes to two original complementary tools to be used to describe and automate the process for secure and access-controlled communication in data-driven middleware. The first tool, namely **ComArmor**, consists of a syntactic language to succinctly describe policy profiles for subjects including any rules for objects while establishing their respective first and second-order priorities. The second tool, namely **Keymint**, builds off the first and consists of a cryptographic toolchain for compiling a global symbolic policy representation resulting in individual transport artifacts as required to deploy each subject.

2.4.1 ComArmor

ComArmor[1] is a profile configuration language for defining Mandatory Access Control (MAC) policies for communication graphs. ComArmor is akin to other MAC systems, but rather than defining policy profiles for Linux security modules as with AppArmor [9], ComArmor defines policy profiles for armoring communications, as the project name's alliteration suggests.
ComArmor is built on the primitives of XACML. Like XACML provides a formalized XML-based markup for specifying governance, and accompanied XML Schema Definition (XSD) for validation. These policies, constructed from hierarchical nesting of compositional profiles that bind objects to subjects with prescribed permissions via attachment expressions, are later read by meta-build stages to procedurally generate end-use transport credentials. For example, a ComArmor profile excerpt from the simple talker-lister network we introduced in Section 1.3 is shown in Figure 2.4. In the policy, we specify two

[1]github.com/comarmor/comarmor

profiles: *'My Talker Profile'* and *'My Listener Profile'*. Both include the basic node's rules, which are omitted here for readability, and specify which topic the two peers can respectively publish and subscribe to, namely "chatter". As showed in the example, in detail, a profile encapsulates an unordered set of rules, child profiles and includes statements. In particular, imported elements via XInclude[2], are either a collection of rules or an entire profile. The defined XACML profiles are given a name, an attachment, and a scope of further permissions and/or sub profiles. The name is simply used to label the profile for the user when debugging, while the attachment is used to define types of subjects the profile is applicable for. An important note, for attachments of sub profiles, is that any expression used in the sub profile should only be expected to be quired if-and-only-if the attachment of the parent profile matches already. I.e. a child profile can only be applicable if the parent profile is applicable as well. The permissions can be arbitrary to the object they govern, but should specify a type and designate a qualifier. The qualifier can be either ALLOW or DENY to permit a MAC framework. Conflicting permission applicable to the same subject that govern the same object will alway conservatively resolving to deny. Given deny supersedes any allow, an applicable allow rule alone is insufficient as the absence of any precedent deny rule must also be satisfied. In this way, users can curtail policies with blanketed allow rules over subspaces of an object, but single out elements in those subspaces, thus revoking specific access to unique resources, e.g. such as a debugger tool publication access to all topics except those for safety critical E-stop signals.

Borrowing design patterns from the AppArmor community, ComArmor provides an equivalent profile concept, but as opposed to attaching to a process by its executable's path, ComArmor attaches profiles to subjects by their Uniform Resource Identifier (URI), e.g. a node namespace in ROS. Again, as for classical access control policies, defined rules are either allowing or denying a specified set of permissions for a given object by URI attachment, e.g. ROS' topic. ComArmor also maintans the same deny by default MAC assumptions of AppArmor.

However, compared to other more general formats, such as the aforementioned XACML, ComArmor takes an approach that is more straightforward in horizontally transferring permission policies onto the computation graph, where objects are essentially channels on a data bus. Additionally, ComArmor is meant to be compactly human-readable while also remaining easily machine generatable. Note that the parser checks for schema compliance after expansion, thus syntax error for a profile may originate from included elements rather than from the root profiles. However, we would still like to eventually support a standard XACML translation compiler to piggyback on the additional static analysis tools available for XACML, affording more formal verification methods, as discussed by the authors in [46], as well as we will discuss in the next chapter.

[2]https://www.w3.org/TR/xinclude/

```
<profiles xmlns:xi="http://www.w3.org/2001/XInclude">
    <xi:include href="tunables/global.xml" parse="xml"/>
    <profile name="My Talker Profile">
        <attachment>/talker</attachment>
        <xi:include href="tunables/node.xml" parse="xml"/>
        <topic qualifier="ALLOW">
            <attachment>/chatter</attachment>
            <permissions>
                <publish/>
            </permissions>
        </topic>
    </profile>
    <profile name="My Listener Profile">
        <attachment>/listener</attachment>
        <xi:include href="tunables/node.xml" parse="xml"/>
        <topic qualifier="ALLOW">
            <attachment>/chatter</attachment>
            <permissions>
                <subscribe/>
            </permissions>
        </topic>
    </profile>
</profiles>
```

FIGURE 2.4: A minimal access control policy for the talker listener example formulated in ComArmor's profile language.

2.4.2 Keymint

On the other hand, Keymint[3] is a framework for generating cryptographic artifacts used in securing middleware systems like ROS, Data Distribution Service (DDS), etc.

Keymint is akin to other meta-build systems, but rather than compiling source code and installing executables in workspaces as with other command-line interfaces (CLI) build sets as colcon [4], Keymint mints keys and notarizes documents in key stores, as the project name's alliteration again plays upon. Keymint provides users pluggable tools for automating the provisioning process for customizing PKI artifacts used with SROS, or Secure DDS plugins [41].

Keymint's approach in minting cryptographic artifacts resembles that of other common meta-build systems, used to compile binary artifacts from source code. Similarly, users create keymint_packages within a workspace initialized by a keymint_profile; a package being a structured source manifest describing how and what artifacts are to be generated for an identity, while the workspace provides a tunable profile to adjust the global build context for all packages. Also, Keymint shares a staggering development cycle, where a workspace has to be *initialized*, *built*, and *installed*.
While each stage in the cycle is subject to the behavior of the plugin invoked as determined by the package's declared format. An example of the workflow pipeline with

[3]github.com/keymint/keymint_tools
[4]https://github.com/colcon/

Keymint is shown in Figure 2.5 for the robot `/RedOne/BB8`. In grey we identify the Comarmor config for a generic robot *BB8*, the *Rebellion* keystore configuration, and the Keymint package configured to work with ROS2 DDS plugin. Those are the necessary files that we need to setup before initializing the workspace. As a consequence, the tool creates the permission and governance specifications (blue in the figure), and instantiates the two Certificate Authority (green in the figure). Once the infrastructure is in place, we can automatically generate the cryptography artifacts, identified in the purple area, to be provisioned to our robot.

While the Keymint library and CLI are intended to be both transport and policy format-agnostic, and instead simply operate upon source packages in a workspace containing public and private resources, plugins for ComArmor and ROS2/DDS are included by default. Future policy acquisition plugins for XACML and MySQL may also be added for more advanced policy management. The ComArmor profile and ROS2/DDS build plugins work together with Keymint's Keymake compiler to gather the applicable policy from the package's URI and compile it into an intermediate representation before installing the generated PKI and fixating the permission and governance files via SMIME.

Essentially, this automates many of the delicate steps in correctly formulating the policy as to be compliant with the transport-specific format. Given the mantra that security and usability must go hand in hand, Keymint provides a conservative default bootstrapped workspace suitable for basic users, in which the only configuration required on part of the user is to provide an initial ComArmor profile for the targeted deployment. This in itself is a task that can be automated via training as demonstrated and further exemplified in the following section 3.6.
In addition to compliance, the Keymint policy compilation process can also ensure the transport artifacts result in a faithful interpretation of the original symbolic policy. For example, ComArmor's unordered rules sets and deny overrides must be considered accordingly when translating to Secure DDS default plugin permission structure given that its Policy Decision Point (PDP) evaluates upon the first found matching rule in an ordered list. Thus ComArmor deny rules must be arranged in the list as to always be considered first for a given object. Additionally, only the minimally applicable subset of the global ComArmor policy is finally embedded into an individual subject's credentials.

These optimizations are perhaps two of many to consider, with additional ones including policy compression via folding of collapsible rules that share compatible criteria, thus saving payload overhead in secure transport handshaking; perhaps another ordered prioritization of rules, i.e. quickening average handshakes by placing more frequently requested rules further ahead in the list for faster lookup. Either of these could be beneficial for real-time communication that must be adapted to support security overhead.

At the time of development, we delayed implementing such further optimization until ROS2's DDS namespace mapping is declared stable. As of this writing, ROS2 is still in definition and the applicable rules for access control policies are still a work in progress

in conjunction with DDS partners.
Abstracting policy definitions away from such complex entanglements is perhaps yet another reason for relying on using intermediate representations and compilers to perform the task on behalf of the system administrator.

With ComArmor and Keymint enabling repeatable and reproducible cryptographic artifacts, revision controlling the source configurations now becomes both rational and elegant. With this, as with the AppArmor community and Debian packaging, we envision further adoptions to allow ROS package maintainers to provide default configurations, audited, and maintained by the community and domain experts.
Anticipating further development discussions of static analysis or manifests of the topology of a system employing orchestration tools and upstarts, using Keymint makes it possible to pre-provision all necessary artifacts for deployment. Alternatively, Keymint's API could be called dynamically to generate artifacts on the fly, as required when remapping subsystem namespaces using the ROS2 launch orchestration.

Public key infrastructure has remained relatively unchanged for many years, at least as it is used in industry. With rapid advances in cryptographic research, new and more powerful cryptographic mechanisms have emerged, such as Ciphertext-Policy Attribute-Based Key Encapsulation Mechanisms (CP-AB-KEM)[44] or other functional encryption schemes that could afford roboticists the definitions of more flexible and secure access control outlines.
With Keymint, we have abstracted the policy from the transport-specific details, so with the ratification and industrial adoption of new paradigms, it is possible to simply upgrade the Keymint Keymake compiler to support newer artifact types.

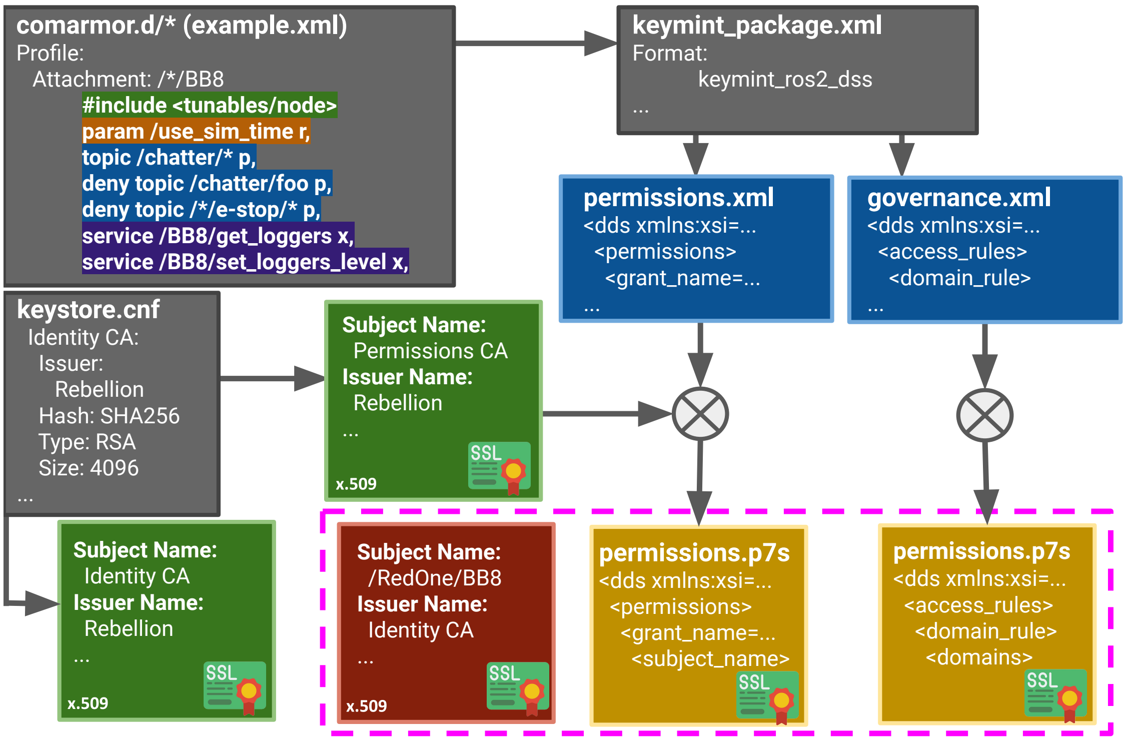


FIGURE 2.5: Flow chart visualization of keymint keystore pipeline

2.4.3 Results

To evaluate our framework, we demonstrate our toolchain on the classic ROS2 talker-listener demo in which the agents communicate via publish-subscribe on the topic chatter. The test procedure works on an operational yet insecure robotic application to verify the integrity of both the policy and the implementation at the transport level. To build our computational graph model G_s we collect the DDS' transport discovery data. The graph is made of a node for each element and oriented edges are created if a connection between two elements exists. In addition to the topic chatter, many more subsystem level topics are also utilized to extend ROS2 node functionality, the latter of which will be shown troubling to properly secure in ROS2 Ardent.

Therefore, starting from the graph G_s we can extract a minimal satisfactory ComArmor policy P_s which is composed of the existing edges. Then, once again, we use the graph G_s to extrapolate the fully connected bi-graph G_{fc} by connecting all the elements with one other. In detail, G_{fc} adds all the permissions to all the objects for all the subjects. We then evaluate graph model G_{fc} semantically using policy P_s to classify the edges and output labels L_{fc} for permitted actions.

Next, we generate the transport policies P_{tp} from P_s by procedurally compiling with Keymint, in this case manifesting as DDS Security artifacts. P_{tp} is then tested by attempting to deploy G_{fc} using a transport implementation. We then infer the permitted action labels L_{tp} for edges in G_{fc} via logged runtime events from the transport.

Thus, we are ready to assert that the set of allowed edges in both L_{fc} and L_{tp} each equate to the set of original edges in G_s. By comparing the differences between the labels against the original model we can extract the set of **false positive** (unintended allow) and **false negative** (unintended deny).
To better understand the functioning, in Figure 2.6 and 2.7 we can see the results from our talker-listener case example, where G_{tp} is the graph equivalent of acquired experimental labels L_{tp}.
The annotated graphs depicts transport test results where only the subject *talker*'s policy amended for the empty partition. The colored edges green/red correspond to allowed/-denied actions respectively. Additionally, *True* positive/negative labels are *dashed*, while **False** positives/negatives are **solid**.

In detail, Figure 2.6 depicts the intersection between labels from G_{tp} with graph G_s, while Figure 2.7 is the relative complement of G_s in G_{tp}. Given the amendment, *talker* is now properly capable of connecting to its own objects, as the case is opposed for *listener* shown via **solid** edges. However, *talker* is now also capable of accessing objects intended solely for *listener*. A nuance exists here in that *talker* and *listener* share a common service name, though no namespace, any participant with misconfigured QOS/policy exchanging with *listener* could leak messages to *talker*. Summarizing, any attempted fix that expands the minimal policy set only serves to open new attack surfaces.

Interestingly, our example results show several false positives and negative for the transport label set enabling unintended circumvention of the policy by way of cross-talk between ROS2 subsystems with namespace omitted. Utilizing our tools, we were able to identify an anomaly with the ROS2 Ardent rmw_connext_cpp implementation whereupon runtime start-up, certain core ROS2 node services are first initialized to the empty string partition. This has resulted in a temporary workaround within SROS2 that simply amends the empty string partition to the list of allowed partition criteria in the transport policy to afford node startup. As of writing, this issue has been reported and solved with changes in DDS namespace mapping in ROS2.

The source material[5] for repeating and reproducing our experimental results is available. This experimental material additionally exemplifies a typical workflow using ComArmor and Keymint.

While auditing experimental results, a set of gratuitous permissions within the SROS2 default template was also brought to our attention through a simple comparative analysis between our minimum spanning policy generated from runtime discovery data and that provided by the SROS2 template. This issue stems from a forgotten holdover workaround in whitelisting DCPS related topics previously necessary for an older DDS security implementation and has also been ticketed upstream.

[5]PPAC_ROS2 Experiments: github.com/ruffsl/PPAC_ROS2

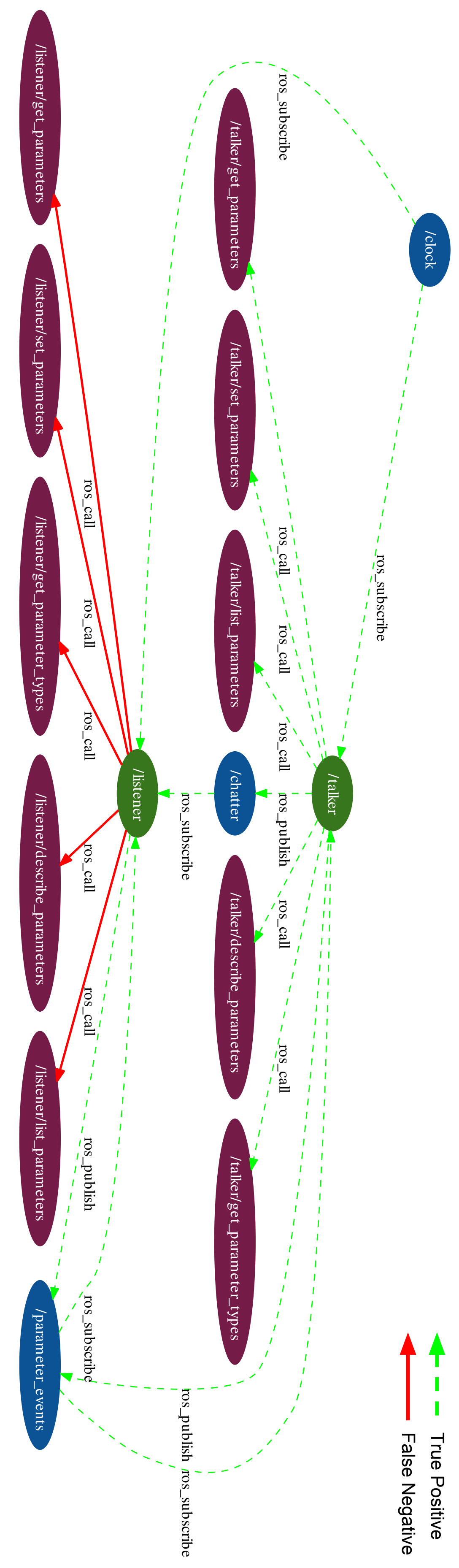


FIGURE 2.6: Publish-Subscribe annotate graph example: intersection between labels from G_{tp} with graph G_s

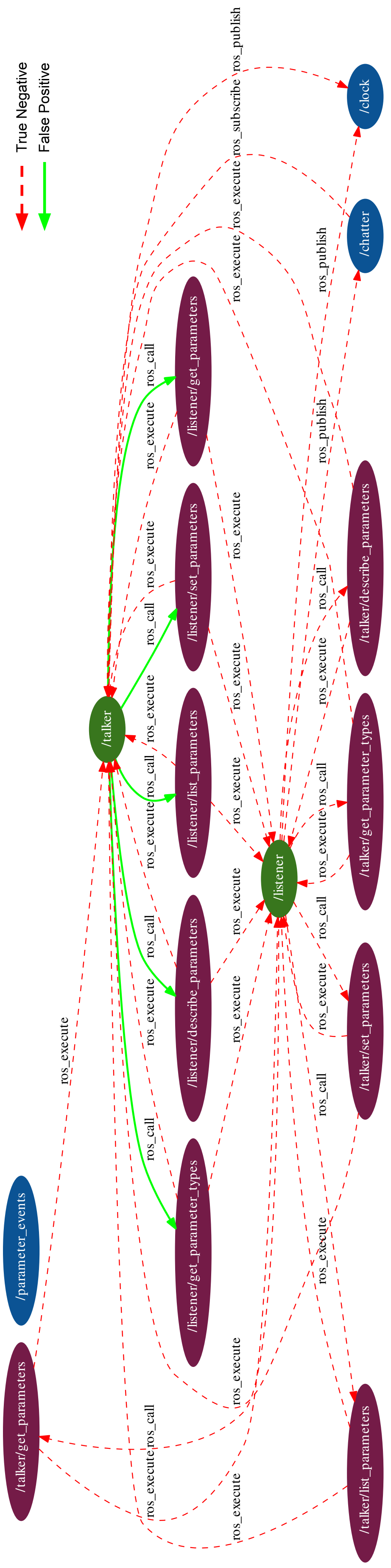


FIGURE 2.7: Publish-Subscribe annotate graph example: relative complement of G_s in G_{tp}

2.5 Conclusions

In this chapter we discussed the concept of access control, the issues and properties of this field with a particular emphasis on the concepts of Authentication and Authorization. We focused on authentication schemes and we introduce novel static and dynamic solution for certificate and attribute distribution in IoT robotic networks.

In particular, we proposed ComArmor as a syntactic access control markup language applicable for robotic systems and computational graphs that leverage channelized hierarchies for invoking actions on subsystem objects. This policy schema was then demonstrated as a proof of concept profile plugin in Keymint, our open source cryptographic meta-build system designed for automating the provisioning of transport specific artifacts enabling secure and controlled communications.

Additionally, we provided a degree of model verification for semantic permission profiles and a systematic test methodology for checking both application satisfiability and rigour of deployed implementation of a governing policy. The take-away here being the surmountable inherent value of higher level policies that service intermediate representations can provide when coupled with a compile framework to afford transport/vendor agnostic access control definitions.

Still, several issues are part of our discussion as mitigates the disclosure of sensitive information (i.e an agent profile), improves privacy in access to partially unauthorized resources (e.g. function output custom sanitization); as well as decoupling the cryptographic operations from the authenticity mechanisms through middleware implementations as oneM2M [72] or other DDS.

Lastly, we've covered potential vulnerabilities in SROS2 as encountered over the development and experimentation of our frameworks, emphasizing the importance of continuous security evaluation throughout design development cycle.

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.

Chapter 3

Vulnerability Excavation

In this chapter, we dive in detail into model verification and privacy issues. In particular, we continue along the line of weak security design choices and we take care of the source of concerns in terms of safety and security in product deployment. To address those problems we can distinguish between two lines of research, either studying the security and hardening solutions for the devices or focusing on the communication infrastructures.

As this thesis centers on the application-layer data-centric approach, our analysis focuses on the latter aspect of the issue and, in particular, we discuss Data Distribution Service (DDS)[2]. We first introduced the highlights of the framework in Section 1.7.2, since it is one of the most used[1] real-time middleware communication mechanism (based on a publish-subscribe model), it makes an interesting use case for the sake of our analysis. Moreover, it is employed in several critical industries including Automotive, Transportation, Healthcare, Energy systems, Aerospace, Defense, etc., being a pivotal component in large-scale distributed networks. However, such applications require a rigid security mechanism since any potential vulnerability can lead to millions in economic losses or damages.

By pursuing the same line of research we discussed in Section 2.4 for provisioning access policy, we identify an interesting opening in the threat model for IoT network framework. In detail, the security model adopted by DDS SPI infrastructure is meant to provide: confidentiality of the data samples, data and messages integrity, authentication and authorization of DDS agents, message-origin and data-origin authentication, and optionally non-repudiation. By enforcing those properties, threats such as unauthorized subscriber and publisher creation, tampering and replay messages, and unauthorized access to data, are blocked. Nevertheless, the proposed threat model doesn't cover permission confidentiality [2], actually exposing the system to similar threats like the one we introduced in section 2.2.
To understand those issues, in this chapter, we explain in further detail how DDS and its Secure Plugins [1] works and how we actually excavate those vulnerability [96].

[1]https://omgwiki.org/dds/who-is-using-dds-2
[2]https://issues.omg.org/issues/DDSSEC12-13

Lastly, besides the certificates' decoupling User-pull architecture discussed in Chapter 2, we focus on possible countermeasures. In particular, we discuss a novel approach to guarantee the confidentiality of the topology of the network via Attribute-Based Encryption (ABE). The general idea is that only the receiving entities - that own the inverse rules of the sender - can decrypt the received policy. Therefore, if the rule allows an entity to subscribe to a topic, only an entity that can publish that topic can decrypt the entry. Still, this is not straightforwardly achievable, since we should encrypt each entry in the policy with a different set of rules.

The rest of this chapter is structured as follows:

- **Background:** in this section we introduce the reader to the threats we are exposed to when policies are leaked or transmitted in plain text and to our analysis.
- **Data Distribution Service (DDS):** in this section we analyze in more detail DDS. We provide an in-deep presentation of the framework and the Authentication and Access Control Security Plugins.
- **Threats & Attack Model:** in this section we discuss our threat and attack model assumptions.
- **Approach:** this section details our approach in partially reconstructing data bus topology and inferring reachability throughout the network at scale.
- **Implementation:** in this section we document our experimental setup and testing infrastructure.
- **Results:** in this section we demonstrate how an attacker may isolate information flow from a single node by identifying critical targets or verify reachability from a selected source to a target destination.
- **Countermeasures:** in this section we discuss some countermeasures to the threats we discussed in the chapter and present a novel approach to secure policies through the usage of Attribute Based Encryption (ABE).
- **Conclusions:** in this section we summarize what we have presented in the chapter and discuss some closing remarks on the work.

The work discussed in this chapter has been published in [96].

3.1 Background

In this section, we further discuss which are the threats we face when policies are leaked or transmitted in plain text. Policies define an agent capability to read and write data in a certain domain in the application. By leaking such information, we can infer the topology

by comparing the capabilities of each agent in the domain and deducing possible connections without having to decrypt ciphered message data. For example, in case the topic names may remain sensitive, might include when a topic offers some pivotal clue as to the amount of confidential resource. In industrial applications, topics might be indexed sequentially, or ordered in a specific manner (e.g. Vehicle Identification Number (VIN)), therefore attackers could use a classical statistical theory of estimation, similar to that applied during WW2 to solve the 'German tank problem'[78] for estimating the number of surveillance sensors or alarms armed in a network or in a physical subsystem. Additionally, it might be possible to identify a system by fingerprinting the associations between topics and entities in the system. In case of high level framework relying on DDS, as ROS2, the usage of standardized naming conventions as including the software package name or data type in the topic namespaces advertised is a common norm. Moreover, via these clear text traits one can assist in recognizing un-patched or exploitable versions of software/firmware.
Unlike traditional network reconnaissance methods like using traceroute, in which an attacker needs to query the network repeatedly to obtain information about the topology[61] that may trigger alarms to network administrators, the methods we present allow an attacker to construct a richer topology of the underlying data bus merely by passively sniffing the packets inside the network. As countermeasure, administrators can employ techniques that obscure the network itself [62] to impede an attacker from reconstructing the true network topology, or that trigger intrusion detection countermeasures before an actual attack is executed. However, in a passive attack scenario, it becomes substantially harder to identify an attacker before any malicious operation is performed.
By analyzing the plugin, we observe how participant' handshakes are performed by exchanging a plain text permission file. Although digitally signed, to preserve integrity and block an unauthorized node from accessing resources via forged permissions, its transmission plain text voids its confidentiality. Thus the significance or internal function of networked devices as with automotive ECUs or robotic sensors may remain transparent to unauthorized users, regardless of whether or not DDS Security is enabled. Therefore, we investigate how revealing the data flow semantics for each node and its functional role in the network renders DDS networks more vulnerable when facing malicious adversaries.

3.2 Data Distribution Service DDS

The Data Distribution Service (DDS)[2][3] is a standardized network middleware protocol and API that aims to provide reliable and scalable services based on a publish-subscribe model, i.e. a data centric model based on a conceptual *Global Data Space*. It provides low-latency data connectivity, extreme reliability, and scalable architecture that mission-critical Internet of Things (IoT) applications need. It simplifies the development of distributed systems by letting software developers focus on the specific purpose of their

[3]https://www.dds-foundation.org/what-is-dds-3/

applications rather than the mechanics of passing information between applications and systems.
From our perspective, a pivotal feature of DDS we underline is its uniquely data-centric setting; in other terms, it works by sending information between applications and systems. Data centricity ensures that all messages include the contextual information an application needs to understand the data it receives. This decoupled nature of publisher-subscriber compared to an ordinary request-response model renders the protocol more suitable for real-time systems and IoT applications.
Applications can choose to have publishers and/or subscribers, where the data model underlying the *Global Data Space*, also known as DDS *Domain*, is a set of data objects. In particular, to the application, the global data space looks like local memory accessed via an API, similarly to for ROS' Parameter server (see Section 1.2).

Similarly to ROS' entities, in DDS a **Publisher** is an object responsible for data distribution and may publish data of different data types. Similarly, a **Subscriber** is an object responsible for receiving published data and making it available for the receiving application. These **DomainParticipant**s (or agents) can respectively write or read in a **Domain**, which denotes the set of all applications that can communicate with each other. Topic objects conceptually fit between publications and subscriptions, and uniquely identify the name, data type and corresponding Quality of Service (QoS) associated with the data on both the publisher and the subscriber sides. In Figure 3.1 we see an high level representation of a simple network in which we have talkers (data writer) and listeners (data reader) over three different topics in the same DDS Domain.

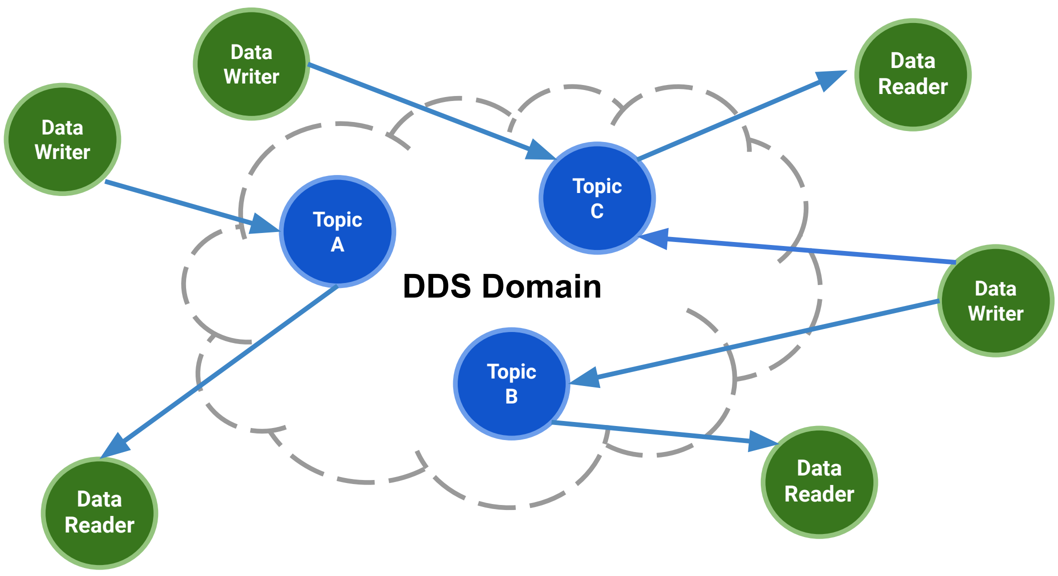


FIGURE 3.1: DDS Domain

3.2.1 Authentication

In this section, we provide the background of the **Authentication** Plugin of Secure DDS. In particular, prior to joining the DDS domain, each DomainParticipant must be authenticated. As we can see in Figure 3.2, on start, a DomainParticipant (agent) authenticates its local identity to others in the network using its own public certificate. In the Figure, participant A sends its identity to B and vice-versa. This Identity Certificate is signed by the Identity Certificate Authority (CA)[1]. Each DomainParticipant will then verify the authentication of a discovered remote peer through a mutual handshake request and reply messages. Among other tokens inside the handshake request, the Identity Certificate and the Domain Participant Permissions (detailed in the next section) of a remote peer will also be included (depicted in bold in the figure); this is precisely the information leakage we exploit in this work.

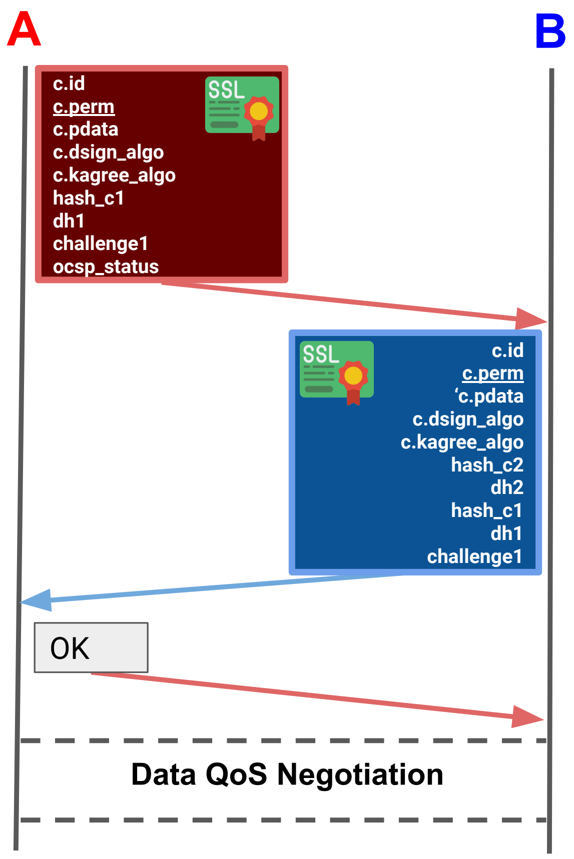


FIGURE 3.2: Simple representation of the Discovery Protocol mechanism

3.2.2 Access Control

In order to ensure authorization of DDS publishers and subscribers, DDS defines an **Access Control** Plugin. The DomainParticipant must be provisioned access to given domains, publish access to topics for data it produces, and subscribe access to topics for data it consumes. In addition, there are more configurable permissions that further segment data access, such as DDS *partitions*, *data tags*, and *domain tags*, that we omit from the discussion at this stage but are accounted for in our analysis approach.

Three configuration documents are associated with the Access Control Service:

1. Permissions CA Certificate
2. Domain Governance signed by Permission CA
3. Domain Participant Permission signed by the Permission CA

The Domain Governance is a XML document specifying the protection policy inside this domain, including whether or not to enforce encryption, whether to set specific limitations on certain topics, etc. The Domain Participant Permission is a XML document containing the permissions of a DomainParticipant. Essentially, it is a set of *grants* that denotes the rules to either reject or allow the DomainParticipant to write or access certain *topics*, inside certain *partitions* of a domain, with certain *data tags* associated with the DomainParticipant. It also includes the domain the DomainParticipant allowed is to communicate in, and the time period that such permissions may be valid [1].

3.3 Threat & Attack Model

In this section, we analyze the approach we pursuit in both the threat and attacker models, including assumptions made when applying our approach.

In addition to the information disclosure of permissions via clear text, the complete **Threat model** we consider is the following.

1. Network traffic may be sniffed, via live or recorded.
2. Network topology may be originally unknown.
3. Network semantics may be originally unknown.
4. Network topology may be non-static.
5. Certificate Authorities remain un-compromised.
6. Participant issued certificates remain un-compromised.
7. Attackers may selectively disrupt network connectivity by dropping packet traffic, route poisoning, or physically disrupting a participant device, for some non-free cost.

Attack Model

As indicated in the list above, the minimum requirements for the execution of the attack is that we necessitate access to network level DDS traffic. For applications such as distributed IoT systems, a stronger assumption would be that an attacker owns all of

the victim's networks simultaneously. However, neither complete nor simultaneous network access is among these minimum requirements for passive or active attacks, given that our later approach inherently reconciles with partial observability over connectivity and time. Thus multi-site measurements, such as recording IP traffic over different connections one at a time, is sufficient for reconnaissance purposes.
One may argue that access to DDS network traffic itself is a rather strong assumption for an attack scenario, given that enterprise networks often operate through VPNs. However, as DDS is a decentralized protocol supporting a range of Quality of Service (QoS) and security features, applications necessitating its adoption often demand peer-to-peer connectivity over lossy channels that are bandwidth and energy limited. Centralized protocols dependent on reliable transport that add additional crypto overhead and deadline latency are subsequently ill-suited for these scenarios. Thus for applications using DDS, the assumption that Secure DDS traffic is observable over the physical network layer is probable, if not most likely for internal system networks, such as inside autonomous vehicles.

A representative IoT example of a highly distributed, realtime, peer-to-peer network would be the Cooperative Intelligent Transport Systems (C-ITS)[4] under development of the European Commission, whose goal is to build a smart-city scaled network to exchange realtime data among vehicles and other road infrastructural facilities to optimize traffic management and take full advantage of highly automated vehicles (level 4/5)[42]. We decompose the attack into two phases distinguishing on whether the attacker has the additional ability to control the network. If the attacker can only observe the network then it can perform **Passive** attacks; on the other hand, if it has some degree of influence on the network, **Active** realtime attacks become feasible.

- **Passive Attack:** in this scenario, an attacker can capture permission tokens from sniffed DDS security handshake traffic which enables it to gradually reconstruct the underlying computation graph. In addition to mapping the structure of the computation graph to physical network topology, reconstruction of the data flow and semantic connectivity is also obtained; e.g. how devices/participants interact with each other over specific data objects.

- **Active Attack:** having reconstructed a rich model of system connectivity in passive attack, thanks to the additional level of control over the network, an attacker is thus situated to execute far more targeted and specialized attacks. For example, if a targeted participant is directly inaccessible due to hardware protections, an active attacker may still selectively isolate it from certain data objects in the rest of the DDS domain by dropping network traffic identified as pertaining to a given topic, e.g IP port/address routes inferred from the secure handshake. In this way, an attacker may effectively remove the target participant from parts of the DDS domain, cutting the information flow from the network, or vice versa, while without

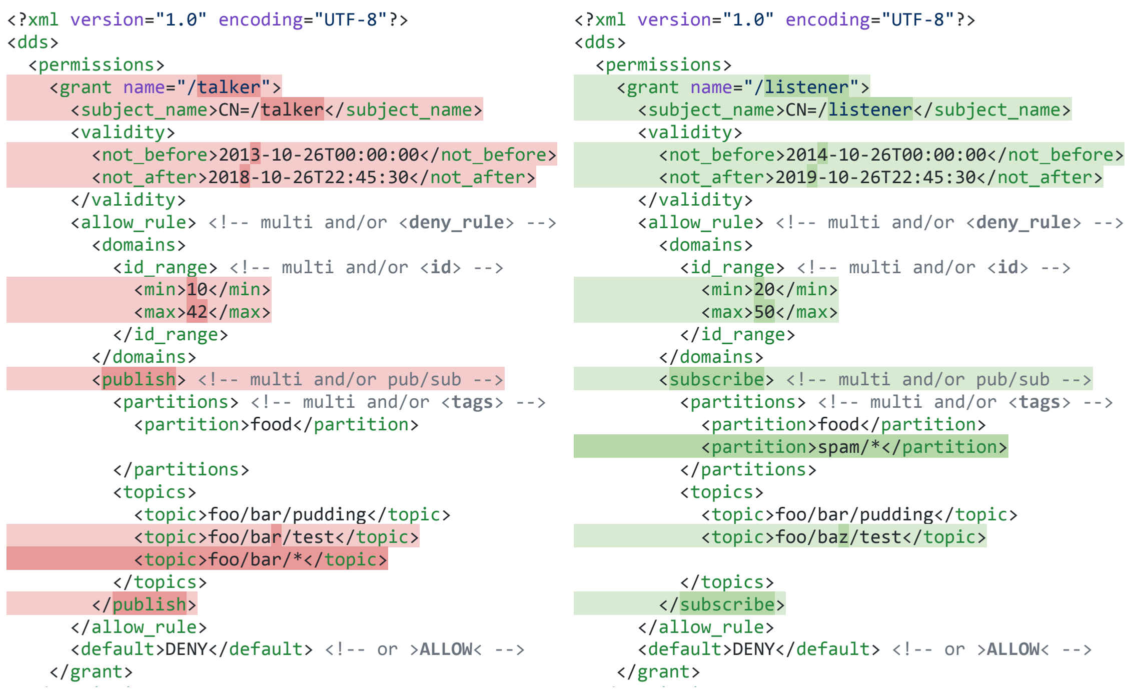
[4]https://ec.europa.eu/transport/themes/its_en

overly disrupting the connectivity to the rest of the physical network, or physically compromising the host hardware.

Interestingly, we notice that even if an attacker does not have the capability to control the traffic, as with secure wireless scenarios, an attacker could revert to less covert methods such as jamming the local spectrum or tampering the physical device. These methods come at a significantly greater cost for the attacker, and thus the expected Return on Investment (RoI) must justify the additional risk. Again, relying on the rich model of system connectivity, an attacker may better prioritize a partial attack surface, e.g. only damaging infrastructures know to host a targeted resource or data object type for the rest of the domain.

3.4 Approach

Under the assumptions discussed in the previous section, we know that once an attacker acquires the handshake packets it can construct the semantic network topology by interpreting the permission files. The sample snippets in Figure 3.3 depict the example permission files that we will use to illustrate this process. In the figure, we can see the permissions for the talker-listener example, in which the talker publishes in the domain *food* the topics in *foo/bar/*. Here we find two specific topics *pudding* and *test*, and a wildcard to publish in the whole domain whatever topic it wants. On the other hand, the listener can use the permission in the domains *food* and *spam* and subscribe to topic *foo/bar/pudding* and *foo/baz/test* in both domains.



```
<?xml version="1.0" encoding="UTF-8"?>
<dds>
  <permissions>
    <grant name="/talker">
      <subject_name>CN=/talker</subject_name>
      <validity>
        <not_before>2013-10-26T00:00:00</not_before>
        <not_after>2018-10-26T22:45:30</not_after>
      </validity>
      <allow_rule> <!-- multi and/or <deny_rule> -->
        <domains>
          <id_range> <!-- multi and/or <id> -->
            <min>10</min>
            <max>42</max>
          </id_range>
        </domains>
        <publish> <!-- multi and/or pub/sub -->
          <partitions> <!-- multi and/or <tags> -->
            <partition>food</partition>

          </partitions>
          <topics>
            <topic>foo/bar/pudding</topic>
            <topic>foo/bar/test</topic>
            <topic>foo/bar/*</topic>
          </topics>
        </publish>
      </allow_rule>
      <default>DENY</default> <!-- or >ALLOW< -->
    </grant>
```

(A) Talker Permissions

```
<?xml version="1.0" encoding="UTF-8"?>
<dds>
  <permissions>
    <grant name="/listener">
      <subject_name>CN=/listener</subject_name>
      <validity>
        <not_before>2014-10-26T00:00:00</not_before>
        <not_after>2019-10-26T22:45:30</not_after>
      </validity>
      <allow_rule> <!-- multi and/or <deny_rule> -->
        <domains>
          <id_range> <!-- multi and/or <id> -->
            <min>20</min>
            <max>50</max>
          </id_range>
        </domains>
        <subscribe> <!-- multi and/or pub/sub -->
          <partitions> <!-- multi and/or <tags> -->
            <partition>food</partition>
            <partition>spam/*</partition>
          </partitions>
          <topics>
            <topic>foo/bar/pudding</topic>
            <topic>foo/baz/test</topic>

          </topics>
        </subscribe>
      </allow_rule>
      <default>DENY</default> <!-- or >ALLOW< -->
    </grant>
```

(B) Listener Permissions

FIGURE 3.3: Highlighted diff between two Secure DDS permission.xml files depicting degrees of overlapping capabilities.

After obtaining the network topology, we also explore how an attacker may formulate queries regarding the network's connectivity. Example queries include:

1. Given the set of nodes (i.e. DDS participants) A and B in the cyclic graph G, what minimal set nodes in G, exclusive of A and B, would need to be disrupted to discontinue information flow from A to B.
2. Given a source A, what are the nodes that we need to offline in order to isolate all information flow from set A.
3. Given a destination set B, what set of nodes would need to be compromised to prevent B from only receiving information from the rest of G.

With this information, an attacker may then selectively partition any node from the rest of the network with minimal invested effort or detectable network disturbance.

3.4.1 Network Topology

To visualize the network topology, we depict it as a directed graph with vertices representing nodes in the network, and edges indicating that there exists at least one topic match between the two connected vertices. By representing the network via a directed graph we can distinguish publish and subscribe actions, which can be depicted using directional edges pointing from a publisher to a subscriber. Additionally, to account for a third 'relay' permission type, we decompose all relay actions to a combination of subscribe and publish capabilities on the topic. This reduction not only decreases the complexity of inferring information flow but also eases the graph visualization and introspection as depicted in Figure 3.4.

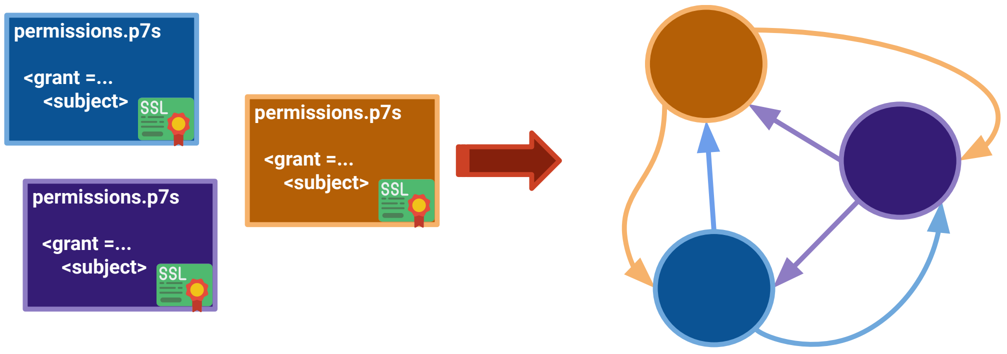


FIGURE 3.4: Starting from the permission file, we visualize the network as a directed graph

3.4.2 Heuristic Graph and Lazy Evaluation

Still, the direct graph representation suffers some drawbacks. In real world applications, a network may consist of hundreds or even thousands of nodes. Such tremendous scales inevitably make any graph construction an intensive task. A naive approach to constructing a network topology requires the consideration of all permission files when computing for the potential intersection in respective permission grants. However, this is impractical given the exhaustive complexity of $O(n(n-1)/2)$ it would result using the formal verification of grant intersections we discussed so far; this is among the most computationally intensive steps in our attacker pipeline. Instead, our approach reduces query time latency via admissible heuristics and lazy evaluation. By first generating a heuristic graph to approximate the information flow, we substantially curtail the number of expensive inferences on grant intersections. Thus, while the initial model may exaggerate apparent connectivity, we can remain assured that resulting reachability queries via formal verification remain complete.

Generating a heuristic graph mostly relies on the fast and admissible approximation as whether to connect or not two vertices. We decompose this approximation process into three phases:

1. A simple directed graph is created by indexing each grant in the permission files, adding respective vertices for both nodes as well as topics to the graph without duplicates, and then connecting nodes and topics according to the direction of information flow. This results in a directed bipartite graph such that vertex set U consists of all nodes in the network and vertex set V includes all topics involved in the network. Figure 3.5 shows the result of this evaluation on the simple network with a talker and a listener introduced above. In this simple network, vertex set U consists of nodes talker and listener, whereas vertex set V is comprised of four topics. This graph can be quickly generated as nodes as well as topics can be iteratively appended on the fly, rather than holistically batching the entire graph all at once and performing intersection checks between any two nodes' publish and subscribe topic expressions.

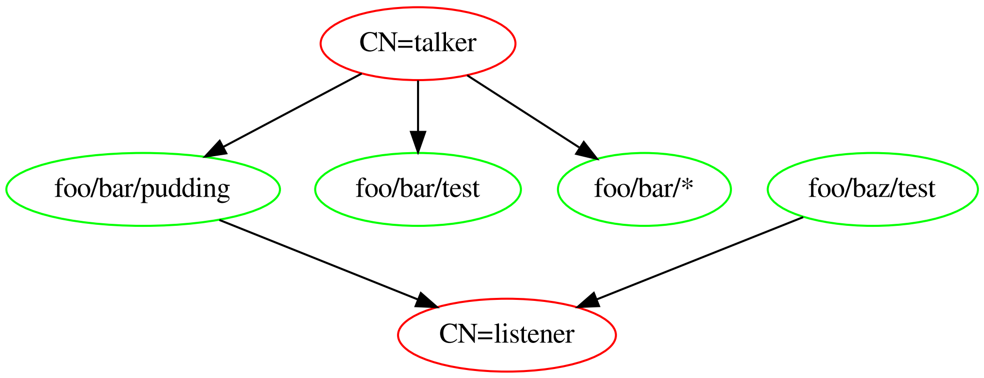


FIGURE 3.5: Raw Graph Obtained by Scanning Permission Files

2. The next phase focuses on combining related topics to form connected components of topics and then collapses the topics into a single vertex. By combining related topics, we mean drawing bidirectional edges between any two topics that match at least once using two way 'fnmatch': the POSIX string matching function chosen in the Secure DDS standard. An example of this is in Figure 3.6, where we have

one such connected component formed by three topics including *foo/bar/pudding*, *foo/bar/**, and *foo/bar/test*. The transition from Figure 3.6 to Figure 3.7a illustrates the process of collapsing the connected components into a single vertex. Although this process is simple, it may potentially increase the total number of paths between different nodes in the network. The extra paths we get do not exist in the real network topology, hence a heuristic graph instead of an exact model.

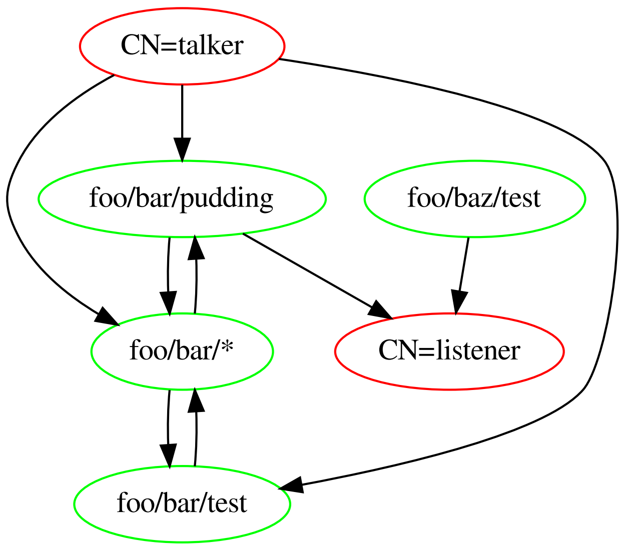
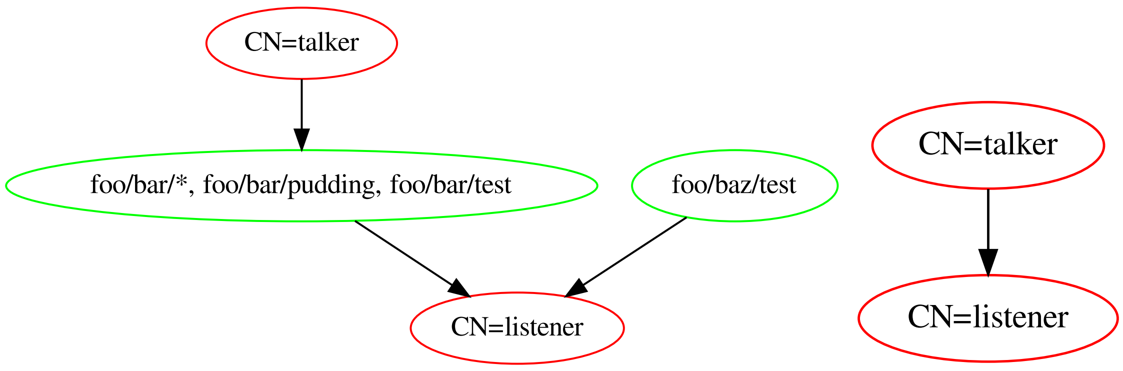


FIGURE 3.6: Connected Graph Obtained by Connecting Topics

3. During the last phase, we further reduce the bipartite graph to a regular network topology by eliminating topics vertex set and connect nodes that might have the capabilities to communicate on some topic. In our simple example, we get Figure 3.7b as a heuristic graph after completing this step, which serves as a foundation to answer the connectivity query. Given the retrieving of a heuristic graph, naive queries on reachability using the simple edge traversal would be inaccurate; our approach resolves this via lazy evaluation. First, using the naive path computed on the heuristic graph, i.e. using Dijkstra Algorithm or A*, the resulting edge sequence or node pairs are iteratively verified for directional connectivity using a satisfaction constraint solver. We describe the reachability verification process in detail in Section 3.4.3. By pruning paths and edges sequences at query time, we avoid unnecessarily checking unfeasible flows derived from topic permission mismatches.



(A) Contracted G (B) Heuristic G

FIGURE 3.7: Contracted Graph is obtained by collapsing related topics into single node, while the Heuristic Graph then is obtained by collapsing topic vertices.

3.4.3 Reachability Verification

During the handshake phase, two DDS DomainParticipants will each verify that the other has the permission to access the resource in question. For the subject node that is advertising its access, we will abstract this into a *subject* representation; containing the information about the name of the subject, the action it is requesting, the topics that it advertises to publish or subscribe, and other subject criteria regulated by access control. Algorithm 1 below details how each node will validate the provided subject representation with the subject's respective permission file, and return a qualifier: ALLOW or DENY of the request. In detail, it will start with the EVALUATE function that for each *grants* verifies with CHECKRULES function each CRITERIA and returns via MATCH function the resulting request response.

Algorithm 1 DDS Security v1.0 Default Access Control Logic

```
procedure EVALUATE($permissions, subject$)
    for $grant$ in $permissions$ do
        $match \leftarrow grant.subject\_name.match(subject)$
        $valid \leftarrow grant.validity(current\_date\_time)$
        if $match$ and $valid$ then
            $qualifier \leftarrow$ CHECKRULES($rules, subject$)
            if $qualifier$ is $None$ then
                return $grant.default$
            else
                return $qualifier$
    return $ERROR$
function CHECKRULES($rules, subject$)
    for $rule$ in $rules$ do
        $domain \leftarrow subject.domain$ in $rule.domainSet$
```

```
15:         $criteria \leftarrow rule.get(subject.action.type)$
16:                                   ▷ Action types: $publish, subscribe, relay$
17:         $match \leftarrow$ CHECKCRITERIA($criteria$, $subject$)
18:         if $domain$ and $match$ then
19:             return $rules.qualifier$
20:                                   ▷ Qualifier types: $ALLOW, DENY$
21:     return $None$
22: function CHECKCRITERIA($criteria$, $subject$)
23:     for $criterion, i$ in $criteria.criterions$ do
24:         $matches[i] \leftarrow$ any $(criterion.match(subject))$
25:                                   ▷ Criterion types: $topics, partitions, tags$
26:     return all $(matches)$
27: function MATCH($publisher$, $subscriber$)
28:     $isMatched \leftarrow publisher.action = PUBLISH$ and
29:     $subscriber.action = SUBSCRIBE$ and
30:     $publisher.topic = subscriber.topic$ and
31:     $publisher.partition = subscriber.partition$ and
32:     $publisher.data_tag = subscriber.data_tag$
33:     return $isMatched$
```

The access control algorithm checks the grant in the permissions file that matches the supplied subject and is valid at the time it is evaluated. For this grant, it sequentially enumerates through all the rules *in order*, and returns immediately if there is a match between the rule and the subject. The matching is conditioned upon many *criteria*s including topics, partitions and data tags. If no rule is matched, the returned qualifier falls through to the grant's default behavior.

To check for permissive exchanges between grants and determine whether data flow between given nodes is possible, we must formally verify the intersection of the two permissions files; i.e. either assert or refute the existence of a pair of matching subjects that satisfy all pairwise constraints. More precisely, given two nodes A and B, and their corresponding permission files PermA and PermB, find two subject actions ActA and ActB such that all the following hold:

$$Evaluate(PermA, ActA) = ALLOW \tag{3.1}$$

$$Evaluate(PermB, ActB) = ALLOW \tag{3.2}$$

$$Match(ActA, ActB)\ or\ Match(ActA, ActB) \tag{3.3}$$

The constraints above dictate that both subject instances must conform to the respective permissions, while the QoS attributes of both subjects such as topic, partition, and data tags must also correspond. The following section details the construction and consumption of such constraints.

3.5 Implementation

To validate our approach, we construct an experimental setup with a reproducible test harness as a pipeline for the entire attacker model[5]. Docker is used to containerize three main processes, as well as virtualize a targeted Secure DDS deployment. In detail, as shown in Figure 3.8 we have a SAT solver, the attacker process, and a simulation environment.

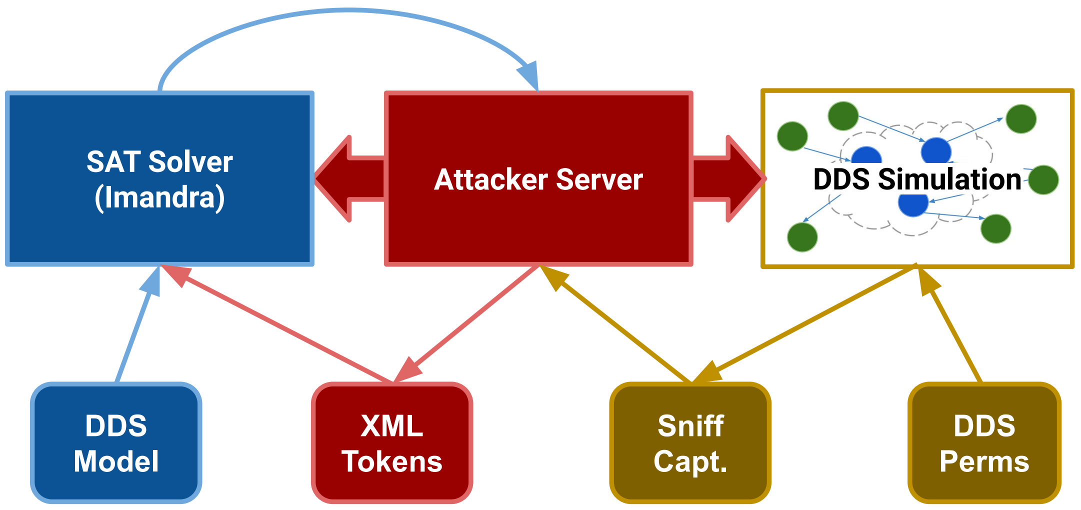


FIGURE 3.8: Deployed test setup for demonstration

In detail, we first programmatically synthesize a DDS application with minimal spanning permissions, valid PKI and CA trust anchors, where the digitally signed governance enforces authenticated encryption for all transport, depicted as *DDS Perms* in the picture. This experimental configuration is then provided to an isolated DDS simulation control that launches each participant in separate containers within a controlled software defined network. The first few seconds of network traffic is consecutively recorded to capture initial Real Time Publish Subscribe (RTPS) protocol discovery data, and then given to the attacker.

The attacker process strips all permission tokens for the raw packet capture and constructs a graph based database of permission tokens and respective origin/destination IP address. This database is then shared with the SAT solver to compute client queries.
For formal verification, we utilize Imandra[6] a formal verification tool originally purposed for model checking financial market software and exchange protocols [45]. It is highly adaptable and performs the nonlinear arithmetic and automated induction, that

[5]https://github.com/ruffsl/dds_security_sniffer
[6]https://www.imandra.ai

we need to infer the proofs or counterexamples to resolve out SAT formulation of permission intersections. By replicating the access control evaluation logic as defined by the DDS specification in OCaml, we can quickly prototype and experiment with alternate security plugin designs with minimal modification. When surveying DDS networks at scale, solving such SAT queries would remain a bottleneck, and thus should be optimized by reducing the number of queries required when inferring about the network. Using Imandra, we simplify the implementation of our approach by replicating the DDS Security SPI specification as functional programs in OCaml, to faithfully model default plugin logic. This also allows for generalizing our automated attack pipeline for non-default plugins; merely update the OCaml model to reflect a new SAT.

Therefore, the model of the access control logic and accompanying token database is used by the SAT service to solve for incremental reachability inquiries from the inquisitive attacker. The attacker uses the proved or refuted subject instances as feedback to prune the heuristic graph until the overall reachability inquiry is determined.
Armed with the associative model of DDS objects to physical network address, the attacker may finally sabotage the target application by selectively deteriorating DDS connections by commanding the simulation controller to drop specified containers from the software defined network.

3.6 Results

In this section, we showcase some example scenarios to elucidate our work and demonstrate the correctness of our implementation on a more complex network; a 2D grid of consisting of 36 nodes is used to maintain readability.

Our simulated DDS network is simply composed of broadcast nodes that periodically publish a KeepAlive diagnostic message to all topics they can publish. Each node also serves as repeater, relaying any subscribed KeepAlive messages to all topics it can publish after appending its own id to avoid cyclic packets. By sampling the packet lineage from different points in the distributed application, unit tests for bisecting information flow can be verified.

In the following we discuss three different scenarios in which we execute different queries.

3.6.1 Source and Target

If both source node and target node are specified, our model outputs a list of nodes as the path from source to target. For example, in Figure 3.9, if the source node is *(5, 0)*, and the target node is *(0, 3)*, then the model outputs a list of nodes containing all the green nodes.

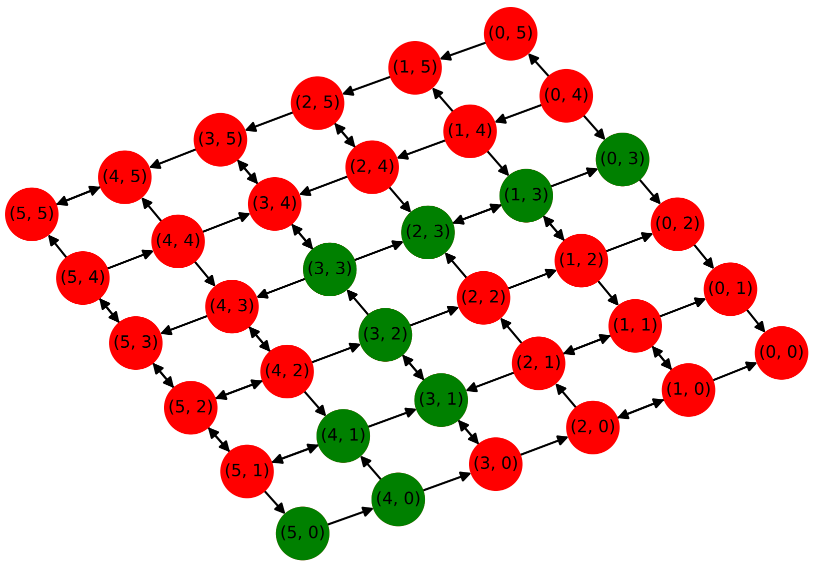


FIGURE 3.9: Query source and target result

3.6.2 Source Only

Given only a source node, our model displays a minimal set of nodes that an attacker needs to take down to prevent the source from passing data to all its subscribers. As shown in Figure 3.10, the source node is colored in blue, and the possible target nodes are colored in green. If the input is the blue node, then the model outputs a set including the three green nodes.

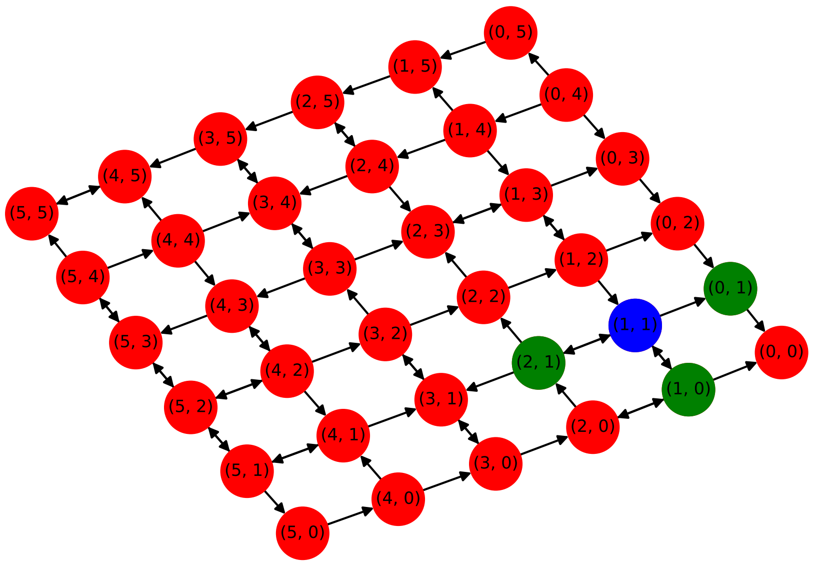


FIGURE 3.10: Query isolate publisher result

3.6.3 Target Only

Similarly, if only a target node is given, we will obtain a minimal set of nodes an attacker needs to attack to prevent the target from acquiring any new information from the network. This is illustrated in Figure 3.11, where the target node is colored in green and its source node is colored in blue.

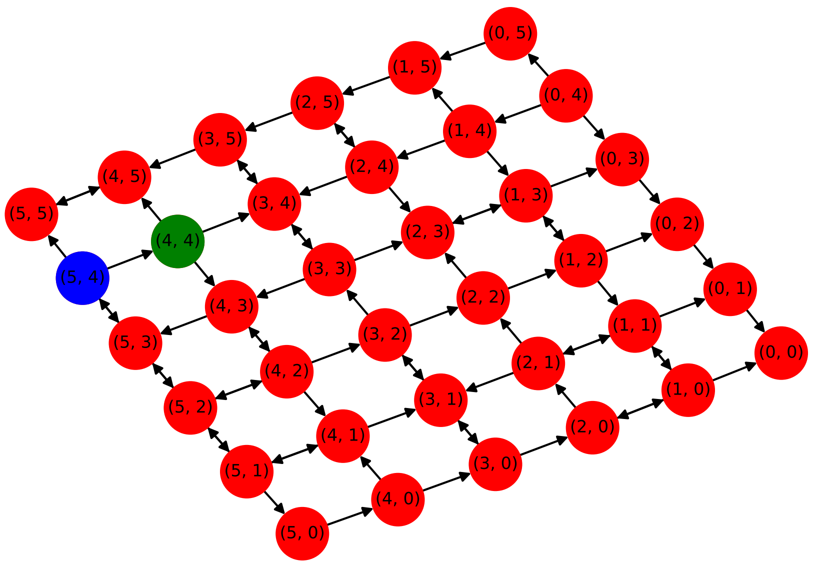


FIGURE 3.11: Query isolate subscriber result

3.7 Countermeasures

In this section, we offer an analysis of countermeasures for the issues highlighted so far. The goal is twofold, on the one hand, the confidentiality of the topology of the network, and on the other optimize clustering of the entities in the domain.
Overall, we want to optimize the network while retaining confidentiality, and - concerning the number of nodes - reduce the usage of the resources by grouping similar agents in a single entity in the domain. Here we focus more on the first aspect of the problem, by discussing solutions to enhance confidentiality.

Since the approach presented predominantly makes use of the current Secure DDS default plugin standard, thus resolving this issue would largely serve to mitigate the feasibility of the attacks demonstrated. The easiest and straightforward answer would be to revise the integration between the crypto and access control plugins to alternatively postpone permission token exchange through a secure channel after the crypto handshake has concluded. This may subsequently add another round trip delay to the overhead introduced in securing connections; however, granted the crypto handshake does not include the action request or response to begin with, it stands to reason that the permission token could be appended to the payload of the subsequent secured requests or responses.

Alternatively, one could seek to obscure the permissions embedded in the token by using an HMAC with a known key, either embedded in the token or distributing it out of band. Each topic/partition/data-tag element in the XML permission document could be replaced with say the base64 encoded digest of the expression string it replaces. Thus, upon receiving an action request from a remote participant, the local participant merely applies the same HMAC to the action and searches for the matching digests in the remote permission list. This has the benefit of obscuring permissions from those sniffing handshake network traffic while making minimal changes to existing vendor libraries. In Fast RTPS for example, the above obfuscation is implementable in less than 60 additional lines using OpenSSL. Although this works for basic string matching, support expression expansion remains an issue given the expressions in the permission list are just as obscured from the interned recipient.

However, both of these mitigations thus far, either postponing permission exchange or obfuscating the fields in the permission token have their potential drawbacks. Using HMAC is particularly vulnerable as message authentication codes only really afford integrity and not confidentiality, i.e. once an attacker knows what they are looking for, it can easily ascertain whether the permission it seeks is present in the token. Given that systems that build upon DDS, like ROS2, commonly use predictably or standardized topic names, it may be trivial to brute force obscured permissions from a limited corpus of topic names, or correlate matching digests across tokens to infer connectivity.

In postponing permission exchange, we merely delay the invocation of the Policy Decision Point, affording a secure channel to remote participants whose privileges we have not yet attested to. Only a single trusted identity need be compromised to begin scraping the permission tokens of others in the same secure distributed network. While DDS discovery information could also be decrypted with the same compromised participant identity, the permission tokens that divulge what data a participant can access versus what they currently advertise can still be advantageous to an attacker as described previously.

Additionally, we could investigate the application of non-interactive zero-knowledge proofs to provide a mechanism for remote attestation of privilege in an access-controlled protocol (without divulging anything more than necessary). However, aside from the provisioning of proving and verification key materials for PKI identities with periods of validity, particular challenges in using frameworks such as zk-SNARK [12] (zero-knowledge succinct non-interactive argument of knowledge) with applications using DDS networks, is maintaining real-time performance in terms of security overhead and scalability. In particular, by limiting the upper bound of computation time for verification, conserving bandwidth for sending larger proofs over the wire, and limited input sizes when transforming permission sets into a boolean circuit.

The priorities here are to maintain **privacy** and support federated network for **interoperability** in highly distributed systems, with multiple CA and domain-specific Attribute Authorities. Moreover, we want to **minimize resources usage** by limiting the overhead only to policy parsing in discovery. Although valid, the possibility of obfuscating the traffic with a Synthetic Packet Engine (SPE) - that generates and injects additional packets to the network whenever needed - to secure from traffic speculations, has been held back due to the trade-off of introducing additional traffic in IoT networks.

As discussed in [5], the majority of the IoT network platforms in the market such as AWS from Amazon, ARM Mbed from ARM, Azure IoT from Microsoft, Google Smart Things, Homekit from Apple, SmartThings from Samsung, Kura from Eclipse, etc, use x.509 certificates and OAuth/2 to manage Authentication. However, access control (Authorization) is handled differently. In particular AWS and Azure, the closest to DDS, are policy-based frameworks. All the others, instead, rely on internal 'oracles', similar to ROS' Master, to store and mediate access to resources/actions, which is useful in a small and well-defined domain (e.g. home automation) but does not scale well in our scenario. Interestingly, all those frameworks' logic works the opposite when compared to DDS. While DDS distributes to the agents the policy set containing their own permissions, the other frameworks, provision to the agents the list of the authorized entities that can query them. Ideally, our goal is to support interoperability, but the solutions offered by the other frameworks are closed with respect to a specif implementation. We want to engineer a solution that maintains confidentiality, therefore, we could exploit Attribute-Based Encryption (ABE) [38].

3.7.1 Attribute Base Encryption

Through ABE, we provide confidentiality in the policies, in such a way that only the receiving entities that own the inverse rules of the sender can decrypt the received policy set. For example, if the rule allows an entity to subscribe to a topic, only an entity that can publish that topic should be able to read the entry.

There are two types of ABE systems [38]:

- **Ciphertext-Policy (CP-ABE):** where ciphertexts are associated with access policies, and keys are associated with sets of attributes
- **Key-Policy (KP-ABE):** where keys are associated with access policies, and ciphertexts are associated with sets of attributes

Still, the base form of ABE is not flexible enough on the set of attributes to be acceptable in a heterogeneous network like the one we are targeting. As discussed in [79], the usage of KP-ABE and XACML lacks flexibility in terms of public attributes that can be used in the encryption. Instead of relying on those base form, the usage of a mixture of CP-ABE and KP-ABE with a suitable access tree with threshold gates could be the solution to our problems [39]. This means that we need to transform our policy sets in a logic circuit in *conjunctive* or *disjunctive* normal form for the ABE in functional encryption scheme (CNF or DNF).

In further detail:

- **Conjunctive Normal Form:** in this form, we need to express the policy as an **AND** of **ORs** and **NOT**
- **Disjunctive Normal Form:** in this form, the policy is an **OR** of **ANDs** and **NOT**

Where the NOT operator can only be used as part of a literal, which means that it can only precede a propositional variable or a predicate symbol.

3.7.2 Policy Representation

To ease the process of conversion to DNF/CNF, we translate the policy into a binary tree. The definition of this machine-readable intermediary representation, ease the creation of the normal forms' formula. In detail, while parsing the policy XML file, we can populate a binary tree by exploiting the URI structure of the resources. In Figure 3.12, we see an exemplifying tree generated starting from the provided policy below.
As an additional optimization, to reduce the analysis cost and facilitate parallelization, we want to decouple the policy into different trees, each rooted in a specific high-level capability (e.g. Publish, Subscribe, etc). This solution simplifies the complexity of the trees and reduces the size of the final ciphertext payload. By doing so, overall, we optimize the number of parameters (n) thus ciphertext and private key sizes - which are the principal agents in the definition of the encryption and decryption complexity - which, in turn, is directly influenced by the number of parameters and might blow up by $n^{3,42}$ factor limiting overall the usefulness of ABE in practice [93].

Figure 3.12 depicts the binary tree of the following simple Policy:

```
Publish
    /frob
    /foo/qux
    /foo/bar/baz
    /foo/bar/mung/bat/camera
```

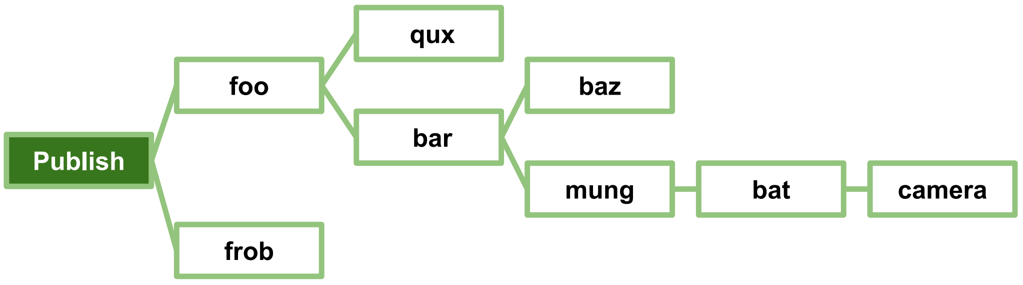


FIGURE 3.12: Binary-tree representation of the Policy

In detail, the depth of the tree is the result of the longest rule in the policy. Interestingly, in this representation, it is trivial to group rules that share a common prefix since they end up in the same branch. This becomes pivotal to perform an analysis of the policies in framework and optimization.

3.7.3 Policy Conversion

As the final step, we propose a rules conversion technique from the tree to DNF. The idea behind the realization is that for each level of the tree we construct, for each node on that level, three labels (parameters):

- An exact match
- An exact deny
- A matching globbing expression (wildcard character)

Below, we can see the results of this transformation to the previous policy:

```
A = Publish
A1 = /
A2 = deny /
A3 = /*

B1 = /foo/                              P1 = /frob/
B2 = deny /foo/                         P2 = deny /frob/
B3 = /foo/*                             P3 = /frob/*

C1 = /foo/bar/                          K1 = /foo/qux/
C2 = deny /foo/bar/                     K2 = deny /foo/qux/
C3 = /foo/bar/*                         K3 = /foo/qux/

D1 =  /foo/bar/baz/                     E1 =  /foo/bar/mung/
D2 = deny /foo/bar/baz/                 E2 =  deny /foo/bar/mung/
D3 =  /foo/bar/baz/*                    E3 =  /foo/bar/mung/*

F1 = /foo/bar/mung/bat/                 G1 = /foo/bar/mung/bat/camera/
F2 = deny /foo/bar/mung/bat/            G2 = deny /foo/bar/mung/bat/camera/
F3 = /foo/bar/mung/bat/*                G3 = /foo/bar/mung/bat/camera/*
```

To generate the access rules in DNF mode, we need to traverse the tree and for each node, we define a perfect match, the current globbing minus the deny rule, and all the previous globbing minus the deny rule. The only node that does not follow this rule is root that have no parent node.

Then, we express the access attributes for the entire tree with the following formulas that are used to encrypt the policy for each level:

- **/:** (A & A1) or (A & A3)
- **/frob/:** (A & P1) or (A & A3 & notP2)
- **/foo/:** (A & B1) or (A & A3 & notB2)
- **/foo/bar/:** (A & C1) or (A & B3 & notC2) or (A & A3 & notC2)
- **/foo/qux/:** (A & K1) or (A & B3 & notK2) or (A & A3 & notK2)
- **/foo/bar/baz/:** (A & D1) or (A & C3 & notD2) or (A & B3 & notD2) or (A3 & notD2)
- **/foo/bar/mung/:** (A & E1) or (A & C3 & notE2) or (A & B3 & notE2) or (A3 & notE2)
- **/foo/bar/mung/bat/:** (A & F1) or (A & E3 & notF2) or (A & C3 & notF2) or (A & B3 & notF2) or (A & A3 & notF2)
- **/foo/bar/mung/bat/camera:** (A & G1) or (A & F3 & notG2) or (A & E3 & notG2) or (A & C3 & notG2) or (A & B3 & notG2) or (A & A3 & notG2)

As result, to reach the policy a receiver needs to decrypt the tree starting from the root formula, towards the leaf, as we illustrate in the following example.

Example: Policy with wildcard character and deny rule

We want to test which entries a subscriber with the following policy can decrypt:

```
Policy: Sub /foo/bar/*
        Deny Sub /foo/bar/baz
```

Then we create a decryption key with the following attributes which is provisioned to the agent:

```
Key attributes {
    A = Publish
    A1 = /
    B1 = /foo/
    C1 = /foo/bar/
    C3= /foo/bar/*
    D2 = deny /foo/bar/baz
}
```

Then, the decrypt execution is the following:

1. OK (A & A1): Get node *foo* and *frob*
2. FAIL no attribute: Drop *frob*

3. OK (A & B1): Get into *foo*
4. OK (A & C1): Get *bar* and *qux*
5. FAIL no attribute: Drop *qux*
6. FAIL (A & D2) Drop *baz*
7. OK (A & C3 & notE2): Get *mung*
8. OK (A & C3 & notF2): Get *bat*
9. OK (A & C3 & notG2): Get *camera* and extract the **Policy!**

The system worked as expected, we are able to extract the policy */foo/bar/mung/bat/-camera*.

Example: Policy with wildcard character and specific topic

In this second example, we want to test the behavior of the solution when the subscriber possesses a different wildcard rule such as:

```
Policy: Sub /foo/bar/*/camera
```

Again, we provision the agent with the decryption key and proceed:

```
Key attributes {
    A = Publish
    A1 = /
    B1 = /foo/
    C1 = /foo/bar/
    C3 = /foo/bar/*
    L = /foo/bar/*/camera
}
```

The execution is the following:

1. OK (A & A1): Get node *foo* and *frob*
2. FAIL no attribute: Drop *frob*
3. OK (A & B1): Get into *foo*
4. OK (A & C1): Get *bar* and *qux*
5. FAIL no attribute: Drop *qux*
6. OK (A & C3 & not D2): **!!!ERROR *baz* extracted!!!**
7. OK (A & C3 & notE2): Get *mung*

8. OK (A & C3 & notF2): Get *bat*
9. OK (A & C3 & notG2): Get *camera* and extract the **Policy!**

Unfortunately, as we can see, the policy was able to extract **baz**. This is the limit of the current implementation which can't support wildcard expressions in between rules. We could solve the problem by adding an extra step to extract the policy or add additional mechanisms (i.e attribute(s)) but this is not trivial.

We could think of a way to split in *prefix* and *suffix* the wildcard string and use them to compute the decryption but, unfortunately, from the policy perspective, we have zero knowledge about the kind of string we should look for. Even the possibility of listing all the possible wildcard policies for the resource is unfeasible because we do not know the deep of the leaf node we are interested in; e.g. the node camera in */foo/bar/* is nested in */foo/bar/mung/bat/*.

One of the possible solutions is to get rid of the wildcard symbols and 'unroll' the policies for the specific network it will be deployed into, similar to the approach we have in ComArmor and Keymint. As discussed, having minimal policies is a paramount feature we should achieve when working with networks' policies.

Unfortunately, we have not identified a perfect solution yet. We still need to investigate which alternatives are most suitable to identify a lightweight approach as with FAME [3], via Fully Secure Ciphertext-Policy Hiding CP-ABE [53] or the application of decision gates (attribute-hiding) inner-product predicate encryption (PE) [55].

3.8 Conclusions

In this chapter we discussed an approach for conducting passive network reconnaissance on systems relying upon Secure DDS, ascertaining a partial topological model of the underlying data bus, and associative mapping between data objects to network addressable participants. Using formal verification and model checking, we can then inquire about directed reachability through the distributed computation graph to efficiently perform vulnerability excavation offline without ever actively engaging with the targeted system. We then demonstrate how such acquired system models may then be used by an active attacker to prioritize targeted participants based on the data objects they represent or the connectivity they facilitate in the larger picture of the system, either by selectively isolating data flow to or from a given data producer/consumer without directly disturbing other participants.

Additionally, over the course of development, two notable vulnerabilities in existing DDS software were discovered while validating our default security plugin models as compared to the OMG specification verses widely used vendor implementations. Firstly, the checking of partition permissions from remote participant connections was found to been omitted from the Policy Decision Point in the access control plugin [7]. This departure in compliance results in unintended declassification of topic data to remote participants who lack the proper authorization for participating within the same secure DDS partition.

Secondly, improper use of topic expression matching was also found in the same vendor implementation[8]. By naively swapping arguments for the query and pattern string in the fnmatch call-sites, two participants can establish a connection using topic names with embedded expressions that match onto topic expressions lists within the permission document. This discrepancy from the specification was first observed and subsequently verified during the aforementioned experiments.

In the last part of the chapter, we discussed some countermeasures, including a novel discussion on the usage of attribute-based encryption (ABE) in robotic networks. We argue about the drawbacks of existing solutions and how we could solve the threats represented by our work.

Although the reconnaissance methods and vulnerability excavation tooling developed over the course of our approach may inevitably prove to be of use to malicious actors, they are also immediately beneficial for general system validation and penetration testing, as when auditing mission-critical systems for flaws in access control design or implementation. For example, when certifying interface isolation between the multimedia and drive-by-wire subsystems in an autonomous automotive, manufacturers may be required to formally prove or refute the set of all satisfiable data channels between the two that would be admissible by the factory permission policy, and assure that no satisfiable channels (covert or otherwise) exist outside of the anticipated set.

[7]https://github.com/eProsima/Fast-RTPS/issues/443
[8]https://github.com/eProsima/Fast-RTPS/issues/441

Chapter 4

Accountability

In this chapter, we discuss accountability and non-repudiation focusing on techniques to monitor and record runtime event data for traceability in design and deployment of IoT network applications. Documenting the operation of a distributed network of multiple components while generating a comprehensive secure trace of the operation is essential for quality control, debugging, systems verification, etc. Indeed, for debugging information flow or in cases of unexpected robot behavior, event logging is fundamentally integral for accountability.

However, when the absolute security of a robotic CPS cannot be guaranteed, the correctness of such event logs is subsequently tenuous. Considering the widespread interest in exploiting self-driving cars and autonomous drones [60], in addition to the recent history in automotive exploitation [64], there can be no doubt that the lack of security represents a real threat [51, 24, 63, 66]. Since digital forensic investigations (DFIs) [77] use digital logs as evidence in post-event analysis or intrusion detection systems (IDS) in electronic devices, their integrity needs to be not underestimated.

To guarantee the device has not been tampered, without relying on specific hardware, a standard technique is to continuously broadcast abridged cryptographic commitments of the system state to a centralized server. As discussed by Veitas *et al.* [89], due to its straightforward server-centric data architecture, this technique is particularly suited for devices without power constraints such as in the automotive industry. However, despite the drawback associated with storage cost, storing personal data, is a daunting task that requires an adequate back-end infrastructure and cybersecurity team to be compliant with governmental regulatory agencies and privacy regulations. Still, IoT and mobile robotic platforms, badly pose with respect to those kinds of solutions. Due to their mobile nature, we can’t assume full-time network availability, in other terms, they are not guaranteed to be always online, which strongly affects the guarantees we need. Rather than wasting resources and bandwidth on a continuous ascertain, we should research other solutions.

Moreover, considering the mass-manufacturing restrictions for mobile robots including build-of-materials, serviceability, data rates, and the cost associated with the utilization of low volume high-cost tamper-proof storage devices (e.g. Write Once Read Many

(WORM) memory), it would be financially unprofitable to commit to specific hardware solutions.
By following the application-layer approach that characterizes this thesis, we want to address an interesting application scenario in which we verify the integrity, authenticity, and completeness of robotic event data while under the threat of malicious/erroneous insertion, omission, or replacement.

To this end, we explore the application of an Event Data Recorder (EDR) based upon cryptographic linked integrity proofs, disseminated via distributed ledgers. In this chapter, we present Black Block Recorder (BBR), an approach combining the use of Digital Signature Algorithms (DSA), keyed-hash Message Authentication Codes (HMAC), and Smart Contract (SC) via Distributed Ledger Technology (DLT) to enable tamper-evident logging, while considering the limited resources available for mobile robotic deployments.
More in detail, our contribution is (i) a framework to make robotic logs immutable by using distributed ledgers via blockchain; (ii) a smart-contract architecture to enforce authenticity and non-repudiation of log integrity proofs.

The rest of this chapter is structured as follows:

- **Background:** in this section we introduce the concepts of token-based ledgers, distributed ledgers technology (DLT), and immutable logs in Event Data Recording necessary for the next discussions.
- **System Architecture:** in this section we define which are the design and implementation properties we account for in the definition of our framework.
- **Approach:** in this section we present the core components of the framework and how we exploit HMAC and smart contract for creating immutable logs.
- **Implementation:** in this section we showcase the implementation and test the performance of the framework on ROS2 and Hyperledger Sawtooth.
- **Conclusions:** in this section we summarize what we have presented in the chapter and comment on the usage of EDRs in robotic systems.

The work discussed in this chapter has been published in [95].

4.1 Background

In this section, we introduce the concepts behind the proposed solution we discuss in the chapter. In particular, we debate the general topic of token-based ledgers, their main properties, and how we apply distributed ledgers technology (DLT) for immutable logs in Event Data Recording for autonomous systems via Secure Enclaves.

Before we dive into details on how we can securely store and manage logs, we need to identify a tool or mechanism that can guarantee the integrity of logs under our threat hypotheses. To guarantee the security of the logging pipeline, we propose to use a Hardware-assisted Trusted Execution Environments (HTEE or simply Trusted Execution Environments (TEE)). Thanks to the HTEE's off-the-shelf solutions that manufacturers are including (by design), on their mass-produced CPUs, we can guarantee high-level security while retaining low-cost.
Often addressed as Secure Enclaves for computation, HTEE provides varying degrees of Isolation, Attestation, and Sealing. In particular, Isolation defines the confinement and protection of runtime execution and memory within the Enclave from other processes sharing the same system. Attestation defines the processes for proving or verifying the integrity of the Enclave to initialize trust across HTEE modules. Sealing defines the secure storage and loading of persistent secrets in the Enclave.
Considering the integration rates from the OEMs of those solutions, the security benefits of applying an affordable hardware solution such as Intel's Software Guard Extensions (SGX) [26] or ARM's TrustZone, for robotic application has become an interesting and active area of study. Staffa *et al.* [82] identified a security bottlenecks in robotic software architectures, and present how HTEE can be integrated with specific ROS components to improve the overall robot security without disrupting the existing infrastructures.

In Figure 4.1, we showcase a logging Enclave application in a high-level view of the network. By using an Enclave, by definition, we provide a security layer and guarantees which are true regardless of the state of the other network's components. Still, despite Enclave covers Log generation, once logs are moved into the external storage "Log Storage" we lose the secure environment provided by the Enclave and any logs' integrity guarantees.

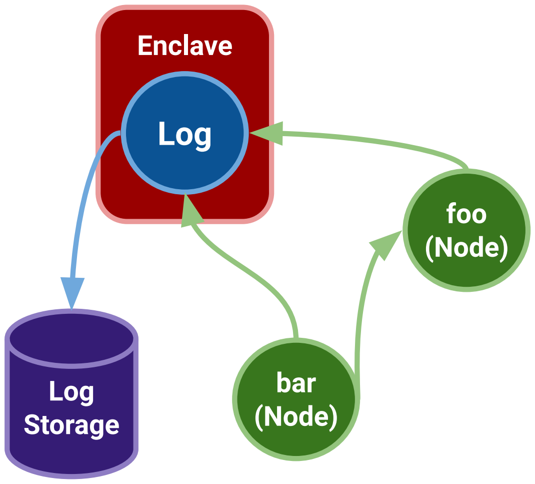


FIGURE 4.1: High-level representation of a network with an Enclave logging node

To secure logs in the storage phase, we could use temper-prof memory. However, the price associated with such devices is not justified for all kinds of applications. Our goal is to identify a less costly resilient storage component that could securely store our logs while remaining inspectable for DFIs. The solution we identified is a token-based blockchain.

In detail, a token-based blockchain is a peer-to-peer distributed ledger that derives its security from public-key cryptography. It might function both as a digital ledger and a mechanism enabling the secure transfer of assets without an intermediary. Anything from currencies to legal titles votes can be tokenized, stored, and exchanged on a blockchain network. The first manifestation of blockchain technology emerged in 2008 with the Bitcoin blockchain [67], a secure, peer to peer substitute to banks and the conventional monetary system. However, in addition to the secure transfer of value, blockchain technology provides a paramount feature, a **permanent forensic record of transactions**.

To record a transaction, each participant in the network has a public address within the Merkle Tree [10], e.g. derived by the hash of its public key, which identifies the user uniquely among all the other participants. Transactions between users are defined by providing as *input* the users' blockchain addresses, the balance transfer, and the hashes of the *outputs* of the last accepted block. Candidate transactions are signed then broadcasted in the p2p network and collected by *validators* that aggregate them in blocks. A candidate block is produced when validators "mine" it - by solving the challenge of the consensus algorithm - whereupon it will be proposed and added to the chain of previous transaction blocks. In the case of a fork, it is only adopted by a validator after it is determined to be the longest chain among the network where all transactions remain valid. The security of the approach is assured by the Byzantine Fault Tolerance (BFT) of the consensus algorithm used, and by relying on the difficulty or inherent cost in subverting the consensus algorithm as a deterrent against malicious actors (i.e. Proof of Work (PoW)).

There exist several kinds of blockchain solutions, ledger technologies, and event data recorder methods. To better understand the state of the art and exemplifying our reasoning, in the remaining part of this section we dive into each topic individually.

4.1.1 Distributed Ledgers Technology

Before DLTs, horizontally scalable Distributed Databases (DDB) were commonly used to replicate record states across trusted storage devices. However, when relying upon CPS infrastructures for data retention, auditing the integrity of classical DDB updates in face of transiently available or compromised devices can deteriorate into an underconstrained problem. Reconstructing postmortem consensus of chronological changes across remaining DDB replications with potentially revoked credentials are classes of issues that can be avoided when disseminating data integrity using DLTs instead.
As an example, Bitcoin [67] provides an alternative to the use of trusted third parties to process and mediate transactions; i.e. the main focal point being the introduction of

distributed trust even under mutually distrusting validators. The resulting distributed ledger contains a chronological evidentiary trail of consensus that every participant can easily audit.

As discussed by BitFury and Garzik white papers [18, 17], blockchain-based ledgers have gained popularity among banks and other financial institutions with the ongoing development of several applications that leverage upon Blockchain's immutability and consensus to validate transactions. Still, the original idea of resource consumption and the democratization of miners poorly adapts from public blockchain to finance blockchains which also suffers from constraints due to latent/limited transaction throughput and scalability due to energy and opportunity costs consumed by traditional Proof of Work (PoW) [69] consensus. To overcome these limitations, and enforce enterprise-level security mechanisms, alternate variants have emerged by defining *public* and *private* distributed ledgers.

On one hand, public ledgers such as Bitcoin have no restrictions on submitting transactions, leading to issues in terms of block's acceptability mechanisms complexity. On the other hand, private ledgers limit those actions to a predefined list of entities and optimize the management of the ledger. To further specialize ledger management and control, an additional classification has been defined, dividing them as *permissioned* and *permissionless*.
In the first case, the identity of peers that act as validators is restricted and predefined (e.g. whitelisted public keys). In the latter, ledgers do not predetermine their miners. As a result of those additional arrangements, we categorize blockchains as in the following:

- **Public Permissionless:** these blockchains are the ones that comply with the original Bitcoin blockchain definition.
- **Public Permissioned:** those ledgers are used to keep control of 'certified' validators and are usually utilized by entities whose operations have to be public.
- **Private Permissioned:** these ledgers work similarly to enterprise distributed databases, in which miners are predetermined and information is not accessible to all.
- **Private Permissionless:** those kinds of ledgers are not used since the application scenario in which the information needs to be private and publicly mined without inspection is not likely to happen.

Moreover, other novel approaches to ledgers have emerged in the Hyperledger Project [30] from Linux Foundation, which seeks to improve the performance of the distributed ledgers by creating open-source enterprise standard libraries for specific applications. However, those can be roughly traced back to the classifications we listed above.

4.1.2 Immutable Logs

As the name suggests, immutable logs require robust tamper-proof logging capabilities. Using cryptographic functions we can enforce integrity, authenticity, and non-repudiation of the logs' entry. Several proposals to achieve immutable logs already exist in the literature. The usual general idea is to use a combination of DSAs and Message Authentication Code (MAC) to unambiguously validate log entries.

It is possible to enforce accountability [20] in a heterogeneous distributed environment and reduce the number of trusted devices. However, the need for central authorities to store and verify the logs makes it necessary to build an additional chain of trust and deploy a distributed storage system for logs (e.g. distributed databases) which is not ideal for our target applications. The use of a distributed versioning implementation such as IPFS [13] can also be a valid option. Still, the use of Merkle DAG does not incorporate verification mechanisms such as smart contracts which are vital to apply validation logic to the system.

Following the discussion in 4.1.1, considering the similarity with Blockchain and its intrinsic security features, leveraging on Bitcoin presents an appealing solution [6]. For example, Snow *et al.* [80] present how Factom[1] distributes immutable logs on Bitcoin chain using an OP_RETURN transaction to store the entry of their client logs. Similarly, Cucurull *et al.* [28] discuss how at Scytl[2] they incrementally secure electronic voting machine results on Bitcoin blockchain. However, cryptocurrencies developers regard this as among the more dubious emerging trends in the wild and an abuse of the OP_RETURN to piggy-back arbitrary data for storage on the Bitcoin Blockchain [8]. As discussed by Matzutt *et al.* [58] the impact of this abuse to store non-financial content on original cryptocurrency blockchains is unsustainable.

On the other hand, Sutton *et al.* [84] follow the concept of checkpoints presented by Cucurull *et al.* to propose a model using Linked Data to optimize the use of Blockchain by constructing a hashing tree rather than continuously dumping logging hashes into the chain. This becomes necessary since the misuse of OP_RETURN has several disadvantages either from the protocol point of view discussed above or because of the transaction fees incurred. All the transactions that need to be published in cryptocurrency blockchains need to pay a fee that will be *'burned'*, deducting the limited balance from the account. Considering the volatile increase of Bitcoin's exchange rate over the years, it's clear that this costly operation is not viable for large scale deployments.

Another barrier to the use of blockchains for storing immutable logs is presented by the **freshness** property of the Blockchain [37]. By design, Blockchain preserves the order of events (i.e. weak freshness), however, the accurate time of events (i.e. strong freshness) is

[1]https://www.factom.com
[2]https://www.scytl.com/en

not guaranteed. The work of Szalachowski [85] offers a workaround using a centralized third party, however, this plays somewhat against our own objectives of distributed trust and scalability. Mobile robots may roam autonomously beyond the network range of centralized base stations or any one particular neighbor, so any agreed reference to time must arise from a distributed consensus.

One notable work preceding much of the others thus far using DLT is that of Crosby *et al.* [27] and presents efficient data structures for tamper-evident logging using history-trees. Although the validation using history-trees is efficient, $O(log_2 n)$, the runtime time for adding checkpoints is no longer constant, $O(log_2 n)$ rather than $O(1)$ for hash-lists. Thus given the lopsided computing resources between robots and off-line auditing infrastructure, our approach opts for hash-lists given the constant overhead in terms of log length, while introducing indexing to enable the parallelization of auditing.

As a high-level overview of our approach, in Figure 4.2, we depict how we can make immutable the information of "Log Storage" by streaming the log data out of the arbitrary storage, by submitting striding checkpoints to the external blockchain, comprised of linked integrity proofs that are indexed as checkpoint transactions.

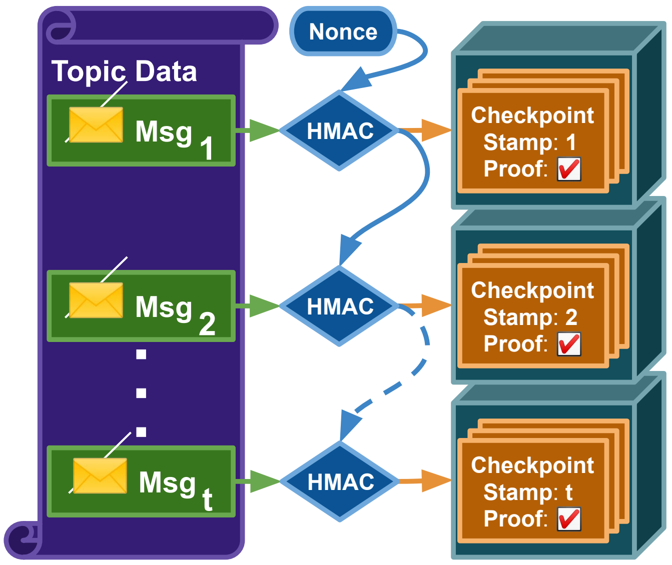

Topic Data
Msg 1
Msg 2
Msg t
Nonce
HMAC
HMAC
HMAC
Checkpoint Stamp: 1 Proof: ✓
Checkpoint Stamp: 2 Proof: ✓
Checkpoint Stamp: t Proof: ✓

FIGURE 4.2: High level overview of immutable logging.

4.1.3 Event Data Recorders

Lastly, we introduce the concept of Event Data Recorders (EDR). EDR has become prevalent within the automotive industry, due in part to regulatory compliance from governmental safety legislation, as well as OEM incentives concerning insurable liability and risk management. Reminiscent of Black Box Recorders in aviation, EDRs are used to log internal and external vehicular data during deployment, such as engine health and status, steering and brake operation, and accident reporting such as obstacle distances or inertial forces from impact. Among the list of transportation infrastructure primed to fully incorporate EDR deployments, autonomous driving vehicles are perhaps first among them. Questions now from both industry and regulatory agencies are being brought forth as per the privacy and security of such EDRs given the pervasive yet critical nature of the data they retain.

The works by Veitas *et al.* includes a two-part series about these particular issues; the first presents Policy Scan [90], a methodology for technology strategy design; i.e. developing concrete actions and products for guiding technology adoption. Policy Scan was developed to address specific types of 'ill-defined' problems in terms of observing, analyzing, and integrating technology developments with policy requirements, social governance, and societal expectations. The second paper [89] applies Policy Scan to the domain of autonomous driving and smart mobility, presenting a proposal for making future autonomous vehicles within collaborative intelligent transportation systems (C-ITS) using EDR as more socially acceptable and legally compliant.

Building upon the above works and also that from Taurer *et al.* [86], a bio-inspired approach to secure data recording for robots, we have designed BBR as an EDR implementation that conforms to the in-vehicle data recording, storage, and access management requirements as specified, while also remaining extendable to general autonomous AI applications using open source robotic middleware and distributed ledger software.

4.2 System Architecture

In this section, we formally define the architecture of the EDR systems in BBR. In detail, we divide the discussion into three sections where we enumerate our design/implementation conformity adapted from prior work in the literature from the Veitas and Taurer [89, 86] we introduced in the last section, and we demonstrate its compliance.

4.2.1 Obligated Roles and Observing Parties

We need to define which roles each entity in the network can play. Our goal is to define a structure in which each entry is guaranteed in terms of integrity and verifiability.

- **Auditors**: observing parties called upon to investigate and validate record archives. e.g. Regulatory Agencies or Governments
- **Custodian**: obligated subject of log content and tasked with log preservation. e.g. Robot or autonomous vehicle OEM.
- **Owner**: mediating party that has a stake in ensuring log integrity/authenticity/-confidentiality. e.g. End-User or Operator.
- **Reporter**: an independent party responsible for faithfully recording events. e.g. Trusted Logger or Recorder Enclave.

Depicted in Figure 4.3, we can see a network perspective of a swarm deployment (e.g. drone network). In particular, our goal is to guarantee Log's integrity of each robot in the swarm via the consensus of its peers.

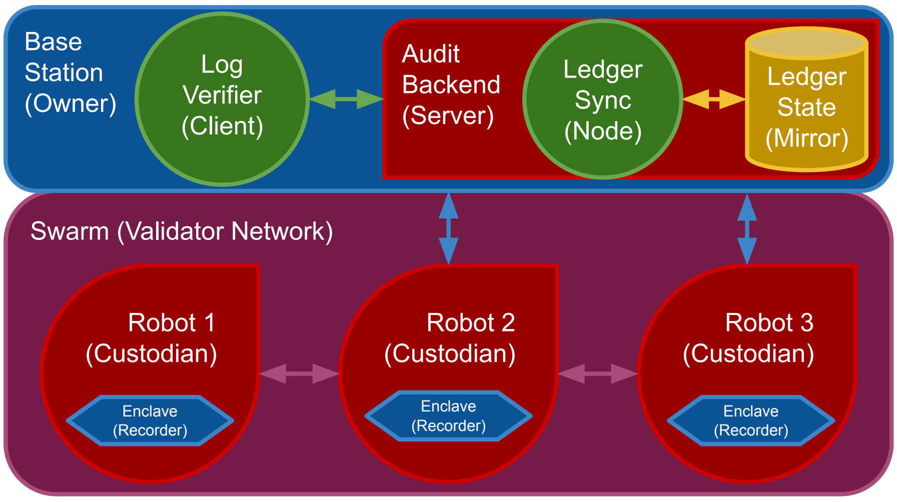


FIGURE 4.3: High-level representation of roles in BBR

4.2.2 Recording, Storage and Access Requirements

To perform operations in BBR, we define a set of requirements that needs to hold:

[R1] **Data provision conditions:** requires consent on behalf of the *Owner* who transitively controls the log assets tracked.

[R2] **Fair and undistorted competition:** trust should be distributed and shared across all validators (a.k.a *Custodians*). In other terms, no single organization may control/monopolize the state of the ledger more than they are trusted by the rest of the validator community/quorum.

[R3] **Data privacy and data protection:** the co-location of logs external to that of the *Custodian* must be prevented. To transmit logs off-platform, event records may be entirely quarantined to the robot's local encrypted storage, reducing the retention risks of co-located persistent data.

[R4] **Tamper-proof access and liability:** integrity and authenticity of logs must derive from an independent *Reporter*. We achieved the property via the distributed archive of checkpoint proofs via ledger, ensuring the integrity and authenticity of the historic clear text data even in case of key exposure (stolen/extracted).

[R5] **Data availability economy:** health and transparency of logs are contingent upon giving *Auditors* appropriate access. This property is facilitated via a shared ledger, a common resource whose health pivots upon the interoperability between many participants; greater participation from a wider community only bolsters the distributed trust.

4.2.3 Defined Primitives and System Properties

We define properties for the system and its primitives as well:

[P1] **Secure identification of physical data sources:** attestation between devices trusted by the *Custodian* and *Reporter* is conducted using standard PKI with chains of trusted Certificate Authorities, where remote attestation between devices inside the robot is achieved.

[P2] **Metadata enrichment:** log event context may be associated to respective *Owner*, *Custodian* and *Reporter* parties. The context is applied by embedding any meta-information into the linked integrity proofs, irrevocably intertwining it with the respective record data. Smart contracts are used to encode structured data that may reside in the public ledger directly.

[P3] **Data exchange and messaging:** authenticated encryption is used in establishing secure connectivity between parties using SROS2 via Secure DDS. Thanks to SROS2 we incorporate real-time compliance, peer to peer encrypted transport, auto-discovery, and quality of service networking.

[P4] **Data recording & storage:** reporting remains flexible in terms of QoS as well as reasonable in resource consumption. In particular, we use BBR storage and bridge plugins for ROSBag2, where the architecture of linked integrity proofs is designed for lossy network transmission and high bandwidth parallel disk IO.

[P5] **Access management:** rights, obligations, and authorization of parties must be explicitly defined and enforceable. In particular, access is managed via smart contracts, where ownership, custodianship, and recorder roles may be established, allowing stakeholders to mediate write access to data record checkpoints. Privacy-preserving querying is feasible given the non-disclosure of event data content via checkpoint proofs, however, anonymous identities remain an open challenge for our approach.

4.3 Approach

By leveraging on the defined system architecture, in this section, we discuss the approach we developed by presenting the two principal components of BBR: the integrity proof, and smart contract specification.
This section details the design and the justification for both to accommodate the constraints of mobile robots and open source frameworks.

4.3.1 Incremental Integrity Proof

To preserve the integrity of the logs without compromising system performance or publicly disclosing private log content, as in [28] we leverage the collision and pre-image resistance of HMAC [11] by chaining the log checkpoints together with key rotations, accommodating **R3**.

Borrowing terminology established in [28] we define a log checkpoint (Chk_i) to be linked with the previous one by using the prior digest (h_{i-1}) as the key bytes when computing the current HMAC digest (h_i) from the log message ($LogMsg_i$):

$$\begin{gathered} Chk_i = (i, h_i) \quad h_i = HMAC(h_{i-1}, LogMsg_i) \\ where \quad h_0 \leftarrow\!\!\$ \{0,1\}^m \end{gathered} \tag{4.1}$$

For privacy, a random nonce is included as the genesis digest (h_0) to inject initial entropy into the linked integrity proofs, ensuring that separate records with similar beginning contents do not repeat the same telltale signature of consecutive proofs.

This deviates from previous work that convolutes the log integrity proof with token-based blockchains and previous financial transaction outputs to achieve immutability. By keying the HMAC with the previous checkpoint digest instead, we reduce the validation of logs to the trivial task of checking a simple hash-chain: i.e. sequentially iterating through $LogMsg_i$ in the log file, ensuring the last linked digest corresponds to the final proof published into the ledger, satisfying **R4**.

By including the index (i) into checkpoints, partial validation or triage discrepancies in the face of missing or corrupted log events can be fine-grained. Provided indices are similarly embedded in log content, validation over large log files is easily parallelizable, accelerating the total verification process.

Previous works such as [28, 84] make the distinction between two different types of checkpoint entries: first, being an incremental link in a chained proof; second, being an anchor point that must always be published to commit to new secret keys while unveiling expired ones for later verification of integrity and authenticity. Our approach to checkpoints makes no such distinction, thus any checkpoint or sequence of checkpoints may be immediately published. This ensures that the latest checkpoints can always be submitted on short notice or without necessarily waiting for previous transactions to be finalized in the global blockchain.

For robotic applications in particular, where mobile computing may be subject to instantaneous brownouts due to self-reliant energy supplies, integrity proofs that require stateful cryptography [28] could leave a recorder without recourse for resumption, as the previously finalized transactions would have included a commitment to a future temporary key that must be revealed upon the next checkpoint. Our approach permits the recorder to quickly recover from the last known integrity proof and resume checkpointing the log wherever it left off (**P4**).

4.3.2 Smart Contract

Section 4.3.1 formalized the incremental integrity proof to ensure log file immutability; however, this efficient method of verification does not in and of itself offer the authenticity and non-repudiation properties still required. Smart Contracts (SC) encapsulate the access control logic for DLT validators to abide by when determining the validity of proposed checkpoint transactions, addressing **R1** and facilitating **R4**.

Instead of relying on colored coins or token metadata in financial blockchains to encode ownership, a dedicated transaction family is defined to regulate write access to the ledger state. A common criterion however is that the validity of candidate transactions must be deterministically computable; i.e. no context external to the current state of the ledger and transaction payload in question should be used in deliberation. This ensures that the validity of any block in the chain can be independently verified in the future.

In Figure 4.4, we depict how a robot (provisioned with its certificate) can submit checkpoint transactions to the blockchain. Upon receiving a submitted batch, the validator will check the corresponding batch's signature matches the custodian identity declared in each transaction. Each such signature is also checked to validate the public identity of the recorder. The recorder identity is used when determining the authorization in appending new checkpoints for a specified asset to the ledger. If the status of the asset has already been finalized, any following append actions are rejected. Otherwise valid checkpoints may be appended to a logged asset's record.

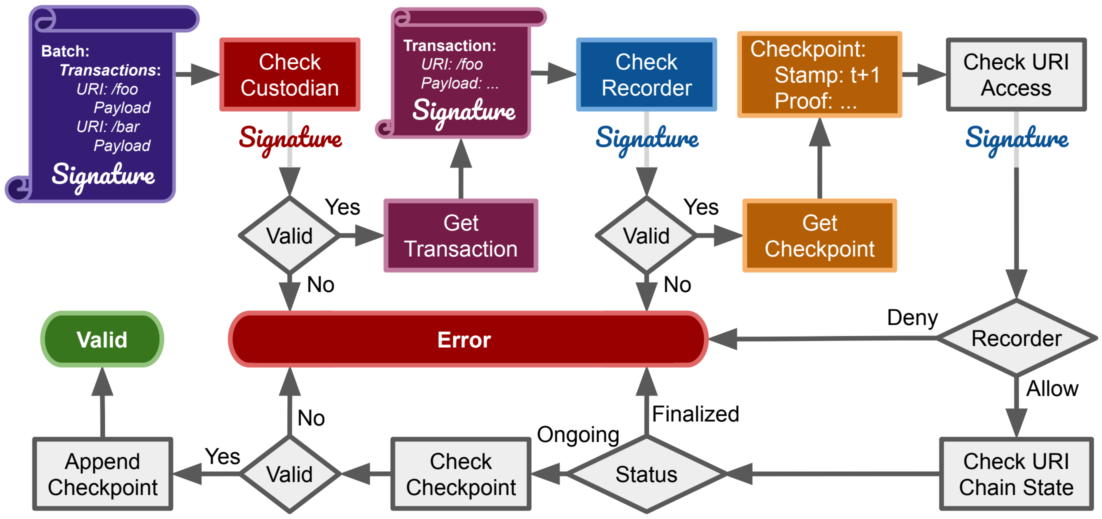


FIGURE 4.4: Flow chart visualization of validating smart contracts for checkpoint transactions.

To ensure the authenticity of a checkpoint committed into the blockchain, transactions are signed via an Elliptic Curve Digital Signature Algorithm (ECDSA), effectively notarizing the identity of the signer. For our purposes, we also register the identity into the blockchain by enrolling its public key into an access control policy stored in the distributed ledger to be used by SCs when verifying candidate checkpoint transactions. Thus, we limit recorders' permissions to append checkpoints only for log files priorly-authorized (**P2**).

To ensure non-repudiation of transactions, our SC mandates that checkpoint indices remain monotonically increasing. The striding of published checkpoint indexes is permitted to enable recorders to down-sample the rate at which integrity proofs are transmitted, versus rate locally generated, as a Quality of Service (QoS) to conserve energy or wireless network bandwidth and ensure the sustainable size of the distributed ledger's state. To curtail the memory growth of the ledger, each validator must maintain locally to participate, a paging ring buffer is adopted to keep rotating the n latest checkpoints for a given log file. The ring buffer size may also be allocated to comply with data retention window requirements per **R3**. However, the genesis digest is always preserved to ensure the indefinite immutability of the entire log into its first record. Another means of non-repudiation such as timestamping could be used to subsume the role of indexing in providing a monotonic counter underneath transaction signatures. However, any reliance on time may be misplaced given the issues of strong freshness discussed in section 4.1.2.

A particular problem presented in previous work includes the open-ended issue of finality of checkpoint termination, i.e. preventing further checkpoints for a given log from appending to the ledger after the log file is intentionally concluded. Such actions could be taken by recorders that are threatened or suspect intrusion, providing a self-destructive deterrent and reducing the utility of private keys recovered by an adversary.

Previous works using token-based blockchains could conclude a checkpoint record via output transactions that are addressed to random public identities, for which no private key is known. This extreme all-or-nothing ownership forfeit is probabilistically final but doesn't afford any other status for provenance, such as 'stalled', 'critical', or methods for reversal, useful for a conditional resumption of logs after a situation is resolved. SCs instead provide greater granularity in this regard for regulating the life cycle of checkpoint records.

4.4 Implementation

As a proof of concept, we implement Black Block Recorder using existing open-source robotic middleware and distributed ledger software. ROS2 was chosen, given its support for secure multicast networking (**P1**) and modular logging storage (ROSBag2[3]) plugin design, enabling secure and efficient tapping of internal/external robotic networks (**P3**). Hyperledger Sawtooth[4] was chosen as the ledger framework for its energy-efficient yet BFT consensus algorithm, multilingual SC processors, permissioned DLT support, and parallelizable transaction architecture.

As both custodian and reporter parties manifest as physical CPS devices, their identities are particularly susceptible to attack. Here, both are used to co-sign batched transactions for validator submission; thus appended forgery checkpoints necessitate the corruption of both the custodian and reporter.

In Figure 4.5, we depict how a custodian robot pipeline for batching transactions works. While every robot platform is held suspect, a Secure Enclave (e.g. TTE) is reserved for the recorder process. As we can see, logged input is securely received within the Enclave and used to cryptographically derive a linked integrity proof specific for each input asset being tracked. As the log data may be streamed to external storage, respective checkpoint transactions are bound to the robot's public identity for batching and then signed by the recorder's private key sealed within the Enclave. Thus only the robot's private key may be used to sign and relay batched transactions for validation. This is a necessary precaution since to append a forgery would necessitate the collusion of both the custodian and its assigned recorder.

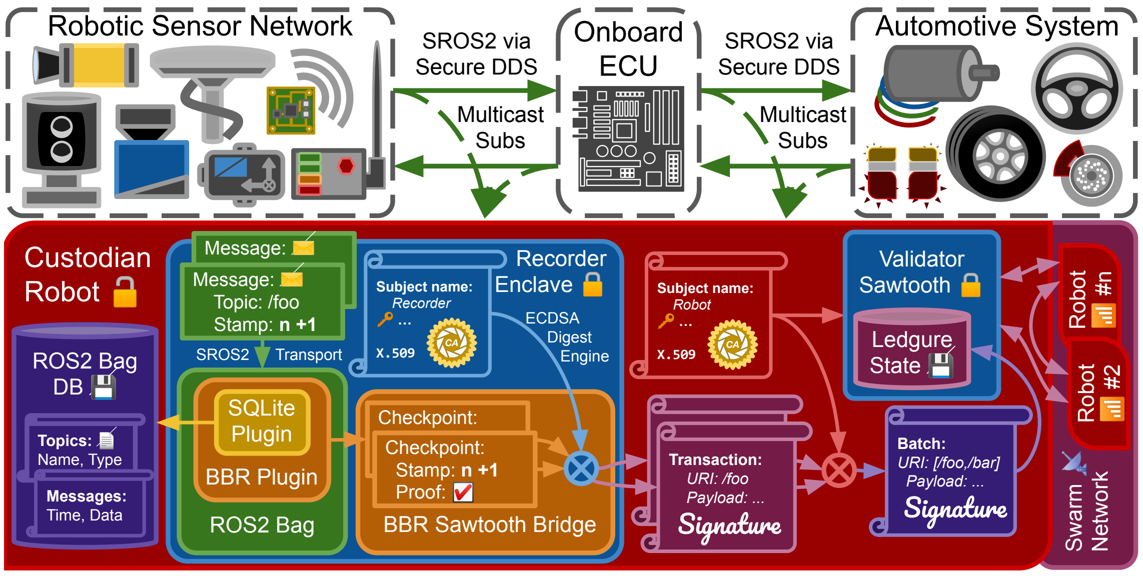


FIGURE 4.5: Flow chart visualization of the immutable logging pipeline

[3]https://github.com/ros2/rosbag2

[4]Hyperledger Sawtooth: hyperledger.org/projects/sawtooth

4.4.1 Checkpoint Integration

To integrate our linked checkpoint approach into ROSBag2, we extend the existing SQLite default storage plugin to additionally compute and broadcast the checkpoints. By constructing a pipeline for all the topics in the network, we commit to the blockchain the log of the robot as depicted in Figure 4.6. Commitments to data and its insertion into the database are achieved via 2D-array hash-chains to render bagfile databases into append-only data structures. The primary axis checkpoints each topic's genesis-block (*foo* and *bar*) and meta-info, while the secondary axis checkpoints the insertion of respective message data. This coupling affords a holistic integrity proof of the entire database while preserving topics as a time series atomic.

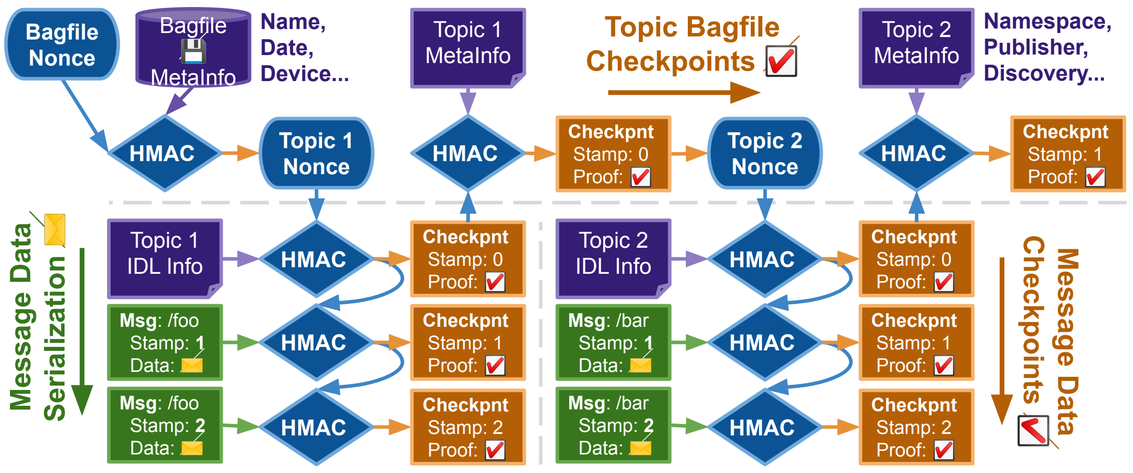


FIGURE 4.6: Hash-chain pipeline checkpoints

The following equations in conjunction with the table and color-coded flow diagram in Figure 4.6, depict the process for checkpointing topic insertions in the following:

$$bits_{bag} \xleftarrow{\$} \{0,1\}^m \tag{4.2}$$
$$bits_A \leftarrow HMAC(bits_{bag}, Proto(name_{bag})) \tag{4.3}$$
$$bits_{A_0} \leftarrow HMAC(bits_A\ , Proto(type_A, format_A)) \tag{4.4}$$
$$bits_B \leftarrow HMAC(bits_{A_0}, Proto(name_A)) \tag{4.5}$$
$$bits_{B_0} \leftarrow HMAC(bits_B\ , Proto(type_B, format_B)) \tag{4.6}$$
$$bits_C \leftarrow HMAC(bits_{B_0}, Proto(name_B)) \tag{4.7}$$
$$bits_{C_0} \leftarrow HMAC(bits_C\ , Proto(type_C, format_C)) \tag{4.8}$$

The nonce $bits_{bag}$ for bagfile (in blue) is combined with bagfile metadata (in purple), deterministically serialized via protobuf to avoid the ambiguity in hashing a list of items, to generate the nonce $bits_A$ for the first inserted topic. This is then combined with IDL information of the topic to generate the genesis digest $bits_{A_0}$. The previous topic's genesis digest and metadata are then combined to seed the nonce for the next topic $bits_B$, which is also reported as the checkpoint (in orange) for the bagfile itself (**P2**). Thereafter, the

cycle repeats for each additional topic.
For messages (in green), the previous digest for the respective topic is combined with the message and its time of arrival to compute the current digest:

$$bits_{A_1} \leftarrow HMAC(bits_{A_0}, Proto(time_{A_1}, data_{A_1})) \tag{4.9}$$

$$bits_{B_1} \leftarrow HMAC(bits_{B_0}, Proto(time_{B_1}, data_{B_1})) \tag{4.10}$$

$$bits_{A_2} \leftarrow HMAC(bits_{A_1}, Proto(time_{A_2}, data_{A_2})) \tag{4.11}$$

In this way, bagfile and message checkpoints are loosely coupled enough for auditing data provenance while remaining independent for concurrent computation and atomic record keeping, even across separate topic streams. It is important to notice in Figure 4.5, how different stages for recording vs signing checkpoints within an Enclave (in blue) have been defined. This allows for modular integration for swapping database storage drivers or alternate ledger infrastructures.

4.4.2 Transaction Family for EDR Smart Contracts

To develop BBR's SCs, we extend from Sawtooth's reference supply chain Transaction Family (TF), used for tracing the provenance and other contextual time-series information of assets. This is formalized and commented (in green) in Figure 4.7 and 4.8 by specifying our SC using the Digital Asset Modelling Language (DAML)[5], an open source domain-specific language for expressing contracts, parties, rights, obligations, and authorization directly (**P5**). The main SC for an EDR agreement is captured in lines 1-14, where the primary parties involved enter as signatories, while a set of external parties are provisioned observational access to the SC's state. Control for creating associative records is solely delegated to the recorder.

```
1  template Edr -- Smart Contract for EDRs
2   with
3     auditors: [Party] -- Regulatory Agencies
4     custodian: Party  -- Robot/Vendor Identity
5     owner: Party      -- User/Operator Identity
6     reporter: Party   -- Logger/TEE Identity
7   where
8     signatory custodian, owner, reporter -- obligated
9     observer auditors -- non-obligated parties
10    ensure unique (custodian :: owner :: reporter)
11    controller reporter can -- create many Records
12      nonconsuming Edr_Record : ContractId EdrRecord
13        with record: Record
14        do create EdrRecord with edr = this; record
15
16 data Record = Record with  -- data type struct
17    r_name: Text   -- "/sensors/exteroceptive/gps"
18    r_type: Text   -- "gps"
19    r_format: Text -- "rtps"
20    r_nonce: Text  -- "<bits_A>"
21    r_digest: Text -- "<bits_A_0>"
22    r_checkpoints: [Checkpoint] -- monotonic list
```

FIGURE 4.7: Smart Contract DAML for EDR agreement and data record structure

[5]DAML Specification: daml.com

Lines 23-40 capture the secondary SC specific to a particular record; i.e. log checkpoints for a single topic. Again the recorder is given the choice to append or finalize the record, while under the assertion that submitted checkpoints remain monotonic. The owner may also choose to finalize the record, whereupon the SC is archived and left entirely immutable in the DLT.

```
23 template EdrRecord -- Smart Contract for Records in EDR
24  with
25    edr: Edr       -- Reference EDR of origin
26    record: Record -- Initial Record state
27  where
28    signatory edr.owner, edr.reporter
29    observer edr.auditors -- custodian can be excluded
30    choice EdrRecord_Append : ContractId EdrRecord
31        with checkpoints: [Checkpoint] -- [] for batching
32        controller edr.reporter -- Only Reporter appends
33        do let -- Update Record with added checkpoints
34            is_valid = checkMonotonic record checkpoints
35            _record = appendCheckpoints record checkpoints
36          assert (is_valid == True) -- Error on invalid
37          create EdrRecord with edr; record = _record
38    choice EdrRecord_Finalize : () -- Archives Contract
39      controller edr.owner, edr.reporter
40        do return () -- Finalized Record is un-appendable
41
42 data Checkpoint = Checkpoint with -- data type struct
43    c_proof: Text -- "<bits_A_1>"
44    c_stamp: Int  -- 1
```

FIGURE 4.8: Smart Contract DAML for EDR Record

Lastly, lines 16-22 and 42-44 specify the structural data a recorder must submit upon choosing actions for the aforementioned SCs. The complete DAML model, including the pending SC for establishing the multiple-party agreement, as well as the test scenario is open sourced and publicly available[6]. As integration between DAML and Sawtooth is still in early development, the TF for BBR remains implemented in the Rust programming language. The DAML model is a faithful representation of the SC logic.

4.4.3 Performance Profiling and QoS Tuning

As a preliminary validation for the tractability of using BBR in robotic systems, we provide a quantitative benchmark comparison in the overhead introduced by utilizing the BBR storage plugin (ROSBag2's default SQLite) and bridging interface by evaluating drop rate performance and CPU load over a range of common sensor message sizes and frequencies. The tests have been executed on ROS2 (Crystal) with a 2.6GHz Intel i7-6700HQ, with RTI Connext RMW on the loopback interface.

Test results in Figure 4.9 show BBR's current performance falls closely in line with the default driver plugin whilst single thread workload remains unsaturated. Marginal performance gains during midrange workloads are likely artifacts attributed to fewer CPU cache misses, due to reduced process idle time given continuous overhead.

[6]EDR DAML Model: github.com/dledr/edr_daml

In regards to the depicted drop-off in throughput, though BBR seeks to checkpoint events at the write-rate to a database, in practice the signing and transmission of those checkpoints over the bridge interface should be rate-limited for purposes stated before. Given that ECDSA transaction signatures remain the predominant cryptographic bottleneck in the pipeline, QoS restrictions or moderating workloads for external validators are necessary (**R2**). Recall that as long as each event is incorporated into the hash-chain, down-sampling checkpoint publication would not inhibit the tamper-evident properties of the log segment with unpublished checkpoints, merely the resolution at which alterations may be pinpointed in the log.

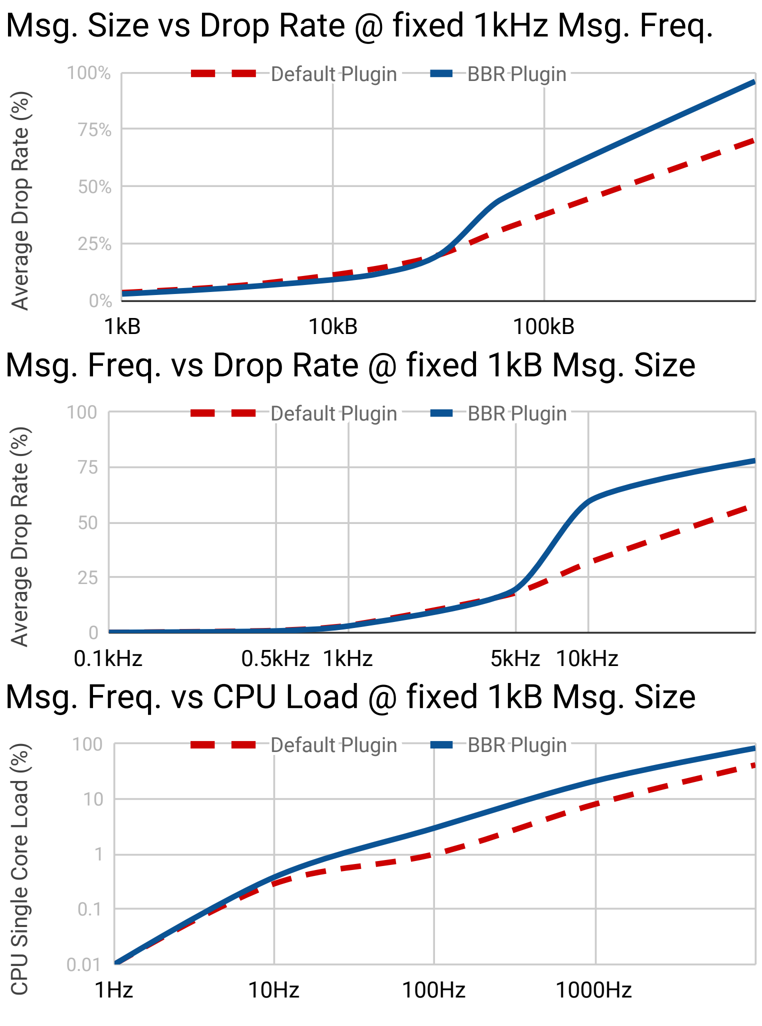


FIGURE 4.9: Performance comparison: BBR vs Default Plugin.

In regards to uplink network usage, validator traffic is conditional upon consensus algorithm, gossip protocol, number of participants, and frequency/size of submitted transactions specific to DLT implementation/framework used. However, to profile the network bandwidth usage specific to our BBR bridge, Table 4.1 includes the minimal payload requirements as calculated using the current serialization schema implemented.

Payload	Size (bytes)	Requirements
Chk_i	36	Ledger Disk Storage
Signed Transaction	$\geq$629	Network Bandwidth
Signed Batch	$\geq$965	Network Bandwidth

TABLE 4.1: BBR Payload Allocations

This depicts the lower bounds given a signed batch transmission with a list of one signed transaction containing an array of one checkpoint. While transactions, in general, may contain arbitrary topic metadata, a checkpoint is simply a tuple of a 256bit hash and 32bit unsigned integer. As a reference, rosbag recording all 20 unique topics on a typical TurtleBot3[7] running ROS1 navigation stack writes to disk at about 1.4MB/s at 1k messages/sec., publishing every checkpoint batched at 1Hz results in approximately 400Kbps in BBR uplink overhead. In practice, a more sensible striding of one checkpoint per topic per sec reduces this to around 110Kbps instead, with a sustainable <1KB/s of ledger state growth (**P4**).

4.5 Conclusions

In this chapter, we discussed the concepts of accountability and non-repudiation, by addressing at once integrity, authenticity, and completeness of robotic event data while under the threat of attacks and/or malfunctioning.

In particular, we focused on the development of a secure logging framework for robots using distributed ledgers and linked integrity proofs to ensure the immutability of continuous event data records.

We discussed how pivotal logging is in terms of autonomous robotic devices via event data recorders (EDR) and how we could overcome the cost-limits that mobile robotic platforms are subject to via blockchain and secure Enclaves.

We achieved authenticity and non-repudiation via dissemination of checkpoint proofs and smart contracts that respect the nature of mutually distrusting parties involved while enforcing a contractual symbiosis between regulators, robots, and users.

Lastly, we have implemented BBR on ROS2 and tested the performance. We discussed the resulting overhead incurred in recording logs via our solutions compared to the conventional robotic logging system (rosbag). We conclude that benefits in ensuring event records remain tamper resistant is pivotal for IoT network which needs an appropriate QoS for reporting topics of significance.

[7]https://www.turtlebot.com/

All in all, we believe that the practicality and utility of applying BBR in security-sensitive robotic domains remain advantageous. We expect this application domain for robotic EDRs for accountability will be one among many exciting frontiers to be explored along the intersection of cybersecurity of robotic systems and distributed ledgers, as the methods presented generalize across future DLT architectures.

Conclusions

In this thesis we have we tackled application security vulnerabilities for robotic systems, focusing our attention on ROS1 and its evolution ROS2 providing significant contributions on three relevant topics related to the cybersecurity of IoT mobile networks in the area of Confidentiality, Authenticity, and Accountability. In particular, we focused on the application-layer security of those networks discussing the threats and proposing a series of tools to aid developers and auditors in secure development and verification of applications via static and formal tools. Our approach is particularly interesting since it offers an independent agnostic toolchain by offering usable tools while retaining powerful mathematical guarantees of correctness.

From the thesis, the immaturity of IoT networks in this critical layer of applications, whose complexity spans across IT cybersecurity and OT safety in CPS is highlighted. We showcased those criticalities working on the open-source framework Robot Operating System (ROS), its evolution ROS2, and Data Distribution Service (DDS).

One of the main elements of complexity at this level is represented by authentication and authorization mechanisms to enforce confidentiality and integrity, as well as accountability in highly distributed networks. Those represent the core components necessary to enforce the CIA pillars.

Regarding authentication and authorization, we tackled the issue from two different points of view. On the one hand, we investigated how policies are distributed in IoT networks, contributing two main approaches to the automatic definition of embedded policy profiles in a trusted network. Actively prevent, at the application level, the disclosure of sensitive information and blocks unauthorized agents by applying a priori access control model. In detail, we discussed the privacy and security issues related to improper policy distribution and introduce two approaches that leverage on x.509 attributes and identity certificate its security guarantees. On the other hand, our work encompasses the contribution of novel meta-build system to the provision of robotic middleware credentials and formal verification tools. This allows to properly generate, maintain, and distribute the number of signed public certificates, ciphered private keys, and access control documents attributed to every identity within the distributed network. This is pivotal considering the complexity and scalability of IoT networks that make the secure orchestration of those systems a demanding process. We showed how an administrator can use our tool to simplify the maintenance and creation of those networks by exploiting the solution on the scale.

In this context, by exploiting the synthesized best practice on access mechanisms and policies defined, we develop formal analysis techniques to excavate discovery access policies payload. We defined a generalized threat model for the IoT network and demonstrate how we could extract network topologies using formal methods and lazy graph evaluations to identify vulnerabilities. Then, we discussed counter measurements to those threats via the application of attribute-based encryption (ABE) in access policy generation and distribution.

The last contribution provided in this thesis consists of a framework for Event Data Recorder (EDR) based on distributed ledgers, namely Black Block Recorder (BBR). Using such a framework, we demonstrated the benefits of ensuring tamper-resistant event record and how Quality of Service (QoS) can be tuned to guarantee a suitable level of accountability without relying on tamper-proof hardware. Finally, we plan to further evolve the work presented in this thesis by targeting multi-tier heterogeneous networks that span several different technologies. In fact, as robotics and networked infrastructures become further complex and integrated (e.g. cloud robotics), additional connectivity bears additional risks, broadening the attack surface, and threatening data privacy. So far, we mitigated the acquisition of sensitive information from networks by means of authenticated encryption and access control. However, by means of techniques such as static Information Flow Control (IFC), and formal analysis, we can complement access control policy to prevents any disclosure of data to subjects with insufficient security levels.

In fact, to overcome the limited onboard capabilities, off-board robotic cloud services are growing making domains such as domestic service robots more viable and capable. However, privacy principles can be difficult to retain given the declassification of data derived from secret or untrusted sources, as often necessary for practical applications. For example, by consuming stereo images endorsed by camera sensors, one cloud service could offer 2D objects recognition, and be transformed into declassified depth data for localization and planning algorithms. Thus the access control policy that is eventually deployed and enforced on the robot should subsequently provision the localization contexts only access data topics of a low-security level, preventing highly sensitive topics such as raw sensor data from being leaked off-board the robot.

This level of awareness is complex to retain, verify, and apply on distributed multi-tier applications. Following our meta-build system and formal approach to vulnerability excavation works, as well as the IFC line of research we plan to further develop white-box formal analysis tools for assurance and trustability of robotics systems.

Bibliography

[1] *About the Data Distribution Service Security Specification Version 1.1*. formal/18-04-01. Object Management Group. July 2018.

[2] *About the Data Distribution Service Specification Version 1.4*. formal/15-04-10. Object Management Group. Apr. 2015.

[3] Shashank Agrawal and Melissa Chase. "FAME: fast attribute-based message encryption". In: *Proceedings of the 2017 ACM SIGSAC Conference on Computer and Communications Security*. 2017, pp. 665–682.

[4] Khalil M Ahmad Yousef et al. "Analyzing cyber-physical threats on robotic platforms". In: *Sensors* 18.5 (2018), p. 1643.

[5] Mahmoud Ammar, Giovanni Russello, and Bruno Crispo. "Internet of Things: A survey on the security of IoT frameworks". In: *Journal of Information Security and Applications* 38 (2018), pp. 8–27.

[6] Nicolai Anderson. "Blockchain Technology: A Game-Changer in Accounting?" In: (2016), pp. 1–5. URL: `https://www2.deloitte.com/content/dam/Deloitte/de/Documents/Innovation/Blockchain_A\%20game-changer\%20in\%20accounting.pdf`.

[7] V. Atluri and David F. Ferraiolo. "Role-Based Access Control". In: *Encyclopedia of Cryptography and Security*. 2011.

[8] Massimo Bartoletti and Livio Pompianu. "An analysis of Bitcoin OP_RETURN metadata". In: *International Conference on Financial Cryptography and Data Security*. Springer. 2017, pp. 218–230.

[9] Mick Bauer. "Paranoid penguin: an introduction to Novell AppArmor". In: *Linux Journal* 2006.148 (2006), p. 13.

[10] Georg Becker. "Merkle signature schemes, merkle trees and their cryptanalysis". In: *Ruhr-University Bochum, Tech. Rep* (2008).

[11] Mihir Bellare. "New Proofs for NMAC and HMAC: Security without Collision Resistance". In: *Journal of Cryptology* 28.4 (2015), pp. 844–878. ISSN: 1432-1378. DOI: `10.1007/s00145-014-9185-x`. URL: `https://doi.org/10.1007/s00145-014-9185-x`.

[12] Eli Ben-Sasson et al. "SNARKs for C: Verifying Program Executions Succinctly and in Zero Knowledge". In: *Advances in Cryptology – CRYPTO 2013*. Ed. by Ran Canetti and Juan A. Garay. Berlin, Heidelberg: Springer Berlin Heidelberg, 2013, pp. 90–108. ISBN: 978-3-642-40084-1.

[13] Juan Benet. "IPFS-content addressed, versioned, P2P file system". In: *arXiv preprint arXiv:1407.3561* (2014). URL: `http://arxiv.org/abs/1407.3561`.

[14] Elisa Bertino, Gabriel Ghinita, and Ashish Kamra. *Access control for databases: Concepts and systems*. Now Publishers Inc, 2011.

[15] Elisa Bertino et al. "Provenance-based analytics services for access control policies". In: *2017 IEEE World Congress on Services (SERVICES)*. IEEE. 2017, pp. 94–101.

[16] Elisa Bertino et al. "The challenge of access control policies quality". In: *Journal of Data and Information Quality (JDIQ)* 10.2 (2018), pp. 1–6.

[17] BitFury Group and Jeff Garzik. "Public versus Private Blockchains. Part 1: Permissioned Blockchains". In: (2015), pp. 1–23. URL: `http://bitfury.com/content/5-white-papers-research/public-vs-private-pt1-1.pdf`.

[18] BitFury Group and Jeff Garzik. "Public versus Private Blockchains. Part 2: Permissionless Blockchains". In: (2015), pp. 1–23. URL: `http://bitfury.com/content/5-white-papers-research/public-vs-private-pt2-1.pdf`.

[19] Benjamin Breiling, Bernhard Dieber, and Peter Schartner. "Secure communication for the Robot Operating System". In: *Proceedings of the 11th IEEE International Systems Conference*. 2017, pp. 360–365.

[20] D. Butin, M. Chicote, and D. Le Métayer. *Log Design for Accountability*. 2013. DOI: `10.1109/SPW.2013.26`.

[21] Gianluca Caiazza. "Security Enhancements of Robot Operating Systems". MA thesis. Ca' Foscari University, 2016.

[22] Gianluca Caiazza, Ruffin White, and Agostino Cortesi. "Enhancing Security in ROS". In: *Advanced Computing and Systems for Security - Volume Eight, Fifth International Doctoral Symposium on Applied Computation and Security Systems, ACSS 2018, Kolkata, India, February 9-11, 2018*. Ed. by Rituparna Chaki et al. Vol. 883. Advances in Intelligent Systems and Computing. Springer, 2018, pp. 3–15. DOI: `10.1007/978-981-13-3702-4\_1`. URL: `https://doi.org/10.1007/978-981-13-3702-4\_1`.

[23] Alvaro A Cárdenas et al. "Challenges for Securing Cyber Physical Systems". In: ().

[24] Stephen Checkoway et al. "Comprehensive Experimental Analyses of Automotive Attack Surfaces". In: *System* (2011), pp. 6–6. ISSN: 15249050. DOI: `10.1109/TITS.2014.2342271`. URL: `http://www.usenix.org/events/security/tech/full{\_}papers/Checkoway.pdf`.

[25] George W Clark, Michael V Doran, and Todd R Andel. "Cybersecurity issues in robotics". In: *2017 IEEE conference on cognitive and computational aspects of situation management (CogSIMA)*. IEEE. 2017, pp. 1–5.

[26] Victor Costan and Srinivas Devadas. "Intel SGX Explained." In: *IACR Cryptology ePrint Archive* 2016 (2016), p. 86.

[27] Scott A. Crosby and Dan S. Wallach. "Efficient Data Structures for Tamper-evident Logging". In: *Proceedings of the 18th Conference on USENIX Security Symposium*. SSYM'09. Montreal, Canada: USENIX Association, 2009, pp. 317–334. URL: `http://dl.acm.org/citation.cfm?id=1855768.1855788`.

[28] Jordi Cucurull and Jordi Puiggalí. "Distributed Immutabilization of Secure Logs". In: ed. by Gilles Barthe, Evangelos Markatos, and Pierangela Samarati. Vol. 9871. Lecture Notes in Computer Science 2. Springer International Publishing, 2016, pp. 122–137. ISBN: 978-3-319-46597-5. DOI: `10.1007/978-3-319-46598-2_9`.

[29] Nicholas DeMarinis et al. "Scanning the Internet for ROS: A View of Security in Robotics Research". In: *arXiv preprint arXiv:1808.03322* (2018).

[30] Vikram Dhillon, David Metcalf, and Max Hooper. "The Hyperledger Project". In: *Blockchain Enabled Applications: Understand the Blockchain Ecosystem and How to Make it Work for You*. Berkeley, CA: Apress, 2017, pp. 139–149. ISBN: 978-1-4842-3081-7. DOI: `10.1007/978-1-4842-3081-7_10`. URL: `https://doi.org/10.1007/978-1-4842-3081-7_10`.

[31] Bernhard Dieber et al. "Application-level security for ROS-based Applications". In: *Proceedings of the 2016 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS 2016)*. Daejeon, Korea, 2016.

[32] Bernhard Dieber et al. "Penetration Testing ROS". In: *Robot Operating System (ROS): The Complete Reference (Volume 4)*. Ed. by Anis Koubaa. Cham: Springer International Publishing, 2020, pp. 183–225. ISBN: 978-3-030-20190-6. DOI: `10.1007/978-3-030-20190-6_8`. URL: `https://doi.org/10.1007/978-3-030-20190-6_8`.

[33] Bernhard Dieber et al. "Security for the Robot Operating System". In: *Robotics and Autonomous Systems* 98 (2017), pp. 192–203.

[34] Patrick Eugster et al. "The Many Faces of Publish/Subscribe". In: *ACM Comput. Surv.* 35 (June 2003), pp. 114–131. DOI: `10.1145/857076.857078`.

[35] Stephen Farrell, Russell Housley, and Sean Turner. *An internet attribute certificate profile for authorization*. Tech. rep. RFC 3281, April, 2002.

[36] David F. Ferraiolo et al. "Proposed NIST standard for role-based access control". In: *ACM Transactions on Information and System Security (TISSEC)* 4 (2001), pp. 224 –274.

[37] Arthur Gervais et al. "Tampering with the Delivery of Blocks and Transactions in Bitcoin". In: *Proceedings of the 22nd ACM SIGSAC Conference on Computer and Communications Security - CCS '15*. 2015, pp. 692–705. ISBN: 9781450338325. DOI: `10.1145/2810103.2813655`. URL: `http://dl.acm.org/citation.cfm?doid=2810103.2813655`.

[38] Vipul Goyal et al. "Attribute-based encryption for fine-grained access control of encrypted data". In: *Proceedings of the 13th ACM conference on Computer and communications security*. 2006, pp. 89–98.

[39] Vipul Goyal et al. "Bounded ciphertext policy attribute based encryption". In: *International Colloquium on Automata, Languages, and Programming*. Springer. 2008, pp. 579–591.

[40] Network Working Group. *The Transport Layer Security (TLS) Protocol Version 1.3*. URL: `https://tools.ietf.org/html/draft-ietf-tls-tls13-18`.

[41] OMG Group. *DDS Security version 1.1*. URL: `https://www.omg.org/spec/DDS-SECURITY/1.1/PDF`.

[42] DG GROW. *GEAR 2030 - High Level Group - Final Report*. European Commission, 2017.

[43] Alex Hern. *Hacking risk leads to recall of 500,000 pacemakers due to patient death fears*. 2017. URL: `https://www.theguardian.com/technology/2017/aug/31/hacking-risk-recall-pacemakers-patient-death-fears-fda-firmware-update`.

[44] Susan Hohenberger and Brent Waters. "Online/offline attribute-based encryption". In: *International Workshop on Public Key Cryptography*. Springer. 2014, pp. 293–310.

[45] Denis A. Ignatovich and Grant O. Passmore. *Creating Safe and Fair Markets*. AESTHETIC INTEGRATION, LTD. Feb. 2015.

[46] Amani Abu Jabal et al. "Methods and tools for policy analysis". In: *ACM Computing Surveys (CSUR)* 51.6 (2019), pp. 1–35.

[47] Markus Jakobsson and Susanne Wetzel. "Security weaknesses in Bluetooth". In: *Cryptographers' Track at the RSA Conference*. Springer. 2001, pp. 176–191.

[48] Rafiullah Khan et al. "Future internet: the internet of things architecture, possible applications and key challenges". In: *2012 10th international conference on frontiers of information technology*. IEEE. 2012, pp. 257–260.

[49] Jongkil Kim et al. "Security and Performance Considerations in ROS 2: A Balancing Act". In: *arXiv preprint arXiv:1809.09566v1* (2018). arXiv: `http://arxiv.org/abs/1809.09566v1 [cs.CR]`.

[50] Constantinos Kolias et al. "DDoS in the IoT: Mirai and other botnets". In: *Computer* 50.7 (2017), pp. 80–84.

[51] Karl Koscher et al. "Experimental security analysis of a modern automobile". In: *Proceedings - IEEE Symposium on Security and Privacy*. 2010, pp. 447–462. ISBN: 9780769540351. DOI: `10.1109/SP.2010.34`.

[52] G Lacava et al. "Current research issues on cyber security in robotics". In: (2020).

[53] Junzuo Lai, Robert H Deng, and Yingjiu Li. "Fully secure cipertext-policy hiding CP-ABE". In: *International conference on information security practice and experience*. Springer. 2011, pp. 24–39.

[54] Francisco Javier Rodrıguez Lera et al. "Cybersecurity in Autonomous Systems: Evaluating the performance of hardening ROS". In: *Málaga, Spain-June 2016* (2016), p. 47.

[55] Allison Lewko et al. “Fully secure functional encryption: Attribute-based encryption and (hierarchical) inner product encryption”. In: *Annual International Conference on the Theory and Applications of Cryptographic Techniques*. Springer. 2010, pp. 62–91.

[56] Andrew Y Lindell. “Attacks on the pairing protocol of bluetooth v2. 1”. In: *Black Hat USA, Las Vegas, Nevada* (2008).

[57] Ibrahim Mashal et al. “Choices for interaction with things on Internet and underlying issues”. In: *Ad Hoc Networks* 28 (2015), pp. 68–90.

[58] Roman Matzutt et al. “A Quantitative Analysis of the Impact of Arbitrary Blockchain Content on Bitcoin”. In: *Proceedings of the 22nd International Conference on Financial Cryptography and Data Security (FC). Springer*. 2018.

[59] J. McClean et al. “A preliminary cyber-physical security assessment of the Robot Operating System (ROS)”. In: *Proc. SPIE*. Vol. 8741. 2013, pp. 874110–874110–8. DOI: 10.1117/12.2016189. URL: http://dx.doi.org/10.1117/12.2016189.

[60] Jarrod McClean et al. “A preliminary cyber-physical security assessment of the Robot Operating System (ROS)”. In: 8741 (2013), p. 874110. ISSN: 0277786X. DOI: 10.1117/12.2016189. URL: http://proceedings.spiedigitallibrary.org/proceeding.aspx?doi=10.1117/12.2016189.

[61] Stuart McClure et al. “Hacking exposed: network security secrets and solutions”. In: (2009).

[62] Roland Meier et al. “NetHide: secure and practical network topology obfuscation”. In: *27th {USENIX} Security Symposium ({USENIX} Security 18)*. 2018, pp. 693–709.

[63] Charlie Miller and Chris Valasek. “A Survey of Remote Automotive Attack Surfaces”. In: *Defcon 22* (2014), pp. 1–90. URL: http://illmatics.com/remoteattacksurfaces.pdf.

[64] Charlie Miller and Chris Valasek. “Remote Exploitation of an Unaltered Passenger Vehicle”. In: *Defcon 23* 2015 (2015), pp. 1–91. URL: http://illmatics.com/RemoteCarHacking.pdf.

[65] Julien Mineraud et al. “A gap analysis of Internet-of-Things platforms”. In: *Computer Communications* 89 (2016), pp. 5–16.

[66] Santiago Morante, Juan G. Victores, and Carlos Balaguer. “Cryptobotics: Why Robots Need Cyber Safety”. In: *Frontiers in Robotics and AI* 2.September (2015), pp. 23–26. ISSN: 2296-9144. DOI: 10.3389/frobt.2015.00023. URL: http://journal.frontiersin.org/Article/10.3389/frobt.2015.00023/abstract.

[67] Satoshi Nakamoto. “Bitcoin: A peer-to-peer electronic cash system”. In: (2008). URL: www.bitcoin.org.

[68] L. Nastase. “Security in the Internet of Things: A Survey on Application Layer Protocols”. In: *2017 21st International Conference on Control Systems and Computer Science (CSCS)*. 2017, pp. 659–666. DOI: 10.1109/CSCS.2017.101.

[69] Karl J O'Dwyer and David Malone. "Bitcoin Mining and its Energy Footprint". In: *IET Conference Proceedings* (2014), 280–285(5). URL: `http://digital-library.theiet.org/content/conferences/10.1049/cp.2014.0699`.

[70] OMG. *Data Distribution Service (DDS) Security Specification, Version 1.1*. Object Management Group, 2018. URL: `https://www.omg.org/spec/DDS-SECURITY/1.1`.

[71] OMG. *Data Distribution Service (DDS), Version 1.4*. Object Management Group, 2015. URL: `https://www.omg.org/spec/DDS/1.4`.

[72] OneM2M. *The interoperability enabler for the entire m2m and IoT ecosystem*. URL: `https://www.onem2m.org/images/files/oneM2M-whitepaper-January-2015.pdf`.

[73] Joon S Park and Ravi Sandhu. "Binding identities and attributes using digitally signed certificates". In: *Proceedings 16th Annual Computer Security Applications Conference (ACSAC'00)*. IEEE. 2000, pp. 120–127.

[74] Joon S Park, Ravi Sandhu, et al. "Smart certificates: Extending x. 509 for secure attribute services on the web". In: Citeseer.

[75] Morgan Quigley et al. "ROS: an open-source Robot Operating System". In: *ICRA workshop on open source software*. Vol. 3. 3.2. Kobe, Japan. 2009, p. 5.

[76] Ricardo Tellez. *Robot Runnin ROS2*. `https://www.theconstructsim.com/a-list-of-robots-running-on-ros2/`. Last accessed 2020-11-09. 2020.

[77] Robert Rowlingson and Qinetiq Ltd. "A Ten Step Process for Forensic Readiness". In: *International Journal of Digital Evidence Winter* 2.3 (2004).

[78] Richard Ruggles and Henry Brodie. "An empirical approach to economic intelligence in World War II". In: *Journal of the American Statistical Association* 42.237 (1947), pp. 72–91.

[79] Xiaolin Si, Pengpian Wang, and Liwu Zhang. "KP-ABE based verifiable cloud access control scheme". In: *2013 12th IEEE International Conference on Trust, Security and Privacy in Computing and Communications*. IEEE. 2013, pp. 34–41.

[80] Paul Snow et al. "Factom Business Processes Secured by Immutable Audit Trails on the Blockchain". In: *Whitepaper, Factom* (2014). URL: `https://github.com/FactomProject/FactomDocs/raw/master/Factom{\_}Whitepaper.pdf`.

[81] Patrik Spiess et al. "SOA-based integration of the internet of things in enterprise services". In: *2009 IEEE international conference on web services*. IEEE. 2009, pp. 968–975.

[82] Mariacarl Staffa, Giovanni Mazzeo, and Luigi Sgaglione. "Hardening ROS via Hardware-assisted Trusted Execution Environment". In: *Robot and Human Interactive Communication (RO-MAN), 2018 IEEE/RSJ International Conference on*. IEEE. 2018.

[83] OASIS Standard. *eXtensible Access Control Markup Language (XACML) Version 3.0*. URL: `https://docs.oasis-open.org/xacml/3.0/xacml-3.0-core-spec-os-en.html`.

[84] Andrew Sutton and Reza Samavi. "Blockchain Enabled Privacy Audit Logs". In: *International Semantic Web Conference*. Springer. 2017, pp. 645–660. DOI: 10.1007/978-3-319-68288-4.

[85] P. Szalachowski. "(Short Paper) Towards More Reliable Bitcoin Timestamps". In: *2018 Crypto Valley Conference on Blockchain Technology (CVCBT)*. 2018, pp. 101–104. DOI: 10.1109/CVCBT.2018.00018.

[86] S. Taurer, B. Dieber, and P. Schartner. "Secure Data Recording and Bio-Inspired Functional Integrity for Intelligent Robots". In: *2018 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. 2018, pp. 8723–8728. DOI: 10.1109/IROS.2018.8593994.

[87] Tully Foote, Katherine Scott. *Community Metrics Report*. http://download.ros.org/downloads/metrics/metrics-report-2020-07.pdf. Last accessed 2020-10-20. 2020.

[88] Nilufer Tuptuk and Stephen Hailes. "Security of smart manufacturing systems". In: *Journal of manufacturing systems* 47 (2018), pp. 93–106.

[89] Viktoras Kabir Veitas and Simon Delaere. "In-vehicle data recording, storage and access management in autonomous vehicles". In: *arXiv preprint arXiv:1806.03243* (2018).

[90] Viktoras Kabir Veitas and Simon Delaere. "Policy Scan and Technology Strategy Design methodology". In: *arXiv preprint arXiv:1806.03235* (2018).

[91] Víctor Mayoral Vilches et al. *White paper: Red Teaming ROS-Industrial*. Tech. rep. Alias Robotics, 2020. URL: https://aliasrobotics.com/files/red_teaming_rosindustrial.pdf.

[92] Feng Wang et al. "A data processing middleware based on SOA for the internet of things". In: *Journal of Sensors* 2015 (2015).

[93] Brent Waters. "Ciphertext-policy attribute-based encryption: An expressive, efficient, and provably secure realization". In: *International Workshop on Public Key Cryptography*. Springer. 2011, pp. 53–70.

[94] Ruffin White, Henrik Christensen, and Morgan Quigley. "SROS: Securing ROS over the wire, in the graph, and through the kernel". In: *Proceedings of the IEEE-RAS International Conference on Humanoid Robots (HUMANOIDS)*. 2016.

[95] Ruffin White et al. "Black block recorder: Immutable black box logging for robots via blockchain". In: *IEEE Robotics and Automation Letters* 4.4 (2019), pp. 3812–3819.

[96] Ruffin White et al. "Network Reconnaissance and Vulnerability Excavation of Secure DDS Systems". In: *2019 IEEE European Symposium on Security and Privacy Workshops (EuroS&PW)*. IEEE. 2019, pp. 57–66.

[97] Ruffin White et al. "Procedurally provisioned access control for robotic systems". In: *2018 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. IEEE. 2018, pp. 1–9.

[98] Ruffin White et al. "SROS1: Using and Developing Secure ROS1 Systems". In: *Robot Operating System (ROS): The Complete Reference (Volume 3)*. Ed. by Anis Koubaa. Cham: Springer International Publishing, 2019, pp. 373–405. ISBN: 978-3-319-91590-6. DOI: `10.1007/978-3-319-91590-6_11`. URL: `https://doi.org/10.1007/978-3-319-91590-6_11`.