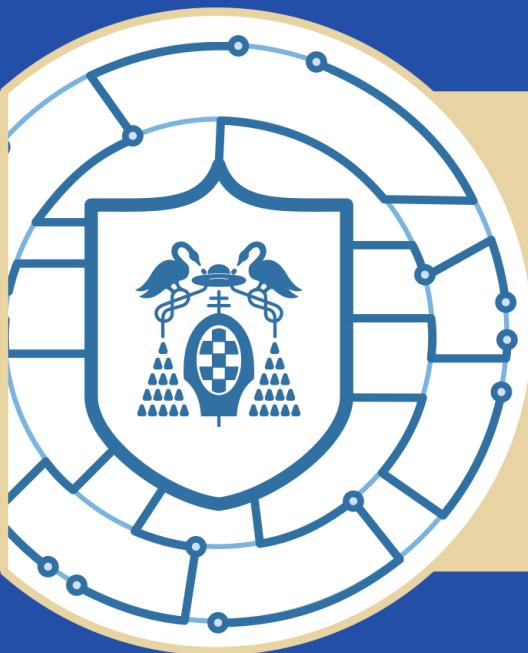


MÁSTER UNIVERSITARIO EN CIBERSEGURIDAD



PROCEDIMIENTO DE
BASTIONADO EN SISTEMAS
ROBÓTICOS

TRABAJO FIN DE MÁSTER

Autor: Lorena Bajo Rebollo

Director: Javier Junquera Sánchez

Codirector: José Javier Martínez Herraiz

UNIVERSIDAD DE ALCALÁ

Escuela Politécnica Superior

Máster en Ciberseguridad

Trabajo Fin de Máster

PROCEDIMIENTO DE BASTIONADO EN SISTEMAS ROBÓTICOS

Autor : Lorena Bajo Rebollo

Tutor : Javier Junquera Sánchez

Cotutor : José Javier Martínez Herraiz

TRIBUNAL:

Presidente:

Vocal 1º:

Vocal 2º:

Alcalá de Henares, a de de 2020

*“Ahora todos estamos conectados por Internet,
como las neuronas en un cerebro gigante”.*

Stephen Hawking

Resumen

Hoy día, con el constante crecimiento de la industria tecnológica y su automatización, se está dando lugar no sólo a la construcción de nuevas herramientas y sistemas de trabajo, si no también a la creación de nuevos puestos de trabajo más especializados que por desgracia irán de la mano de nuevas amenazas que afectarán a estos nuevos sistemas y herramientas informáticas.

El objetivo de este proyecto de investigación será realizar una evaluación de los posibles peligros que pueden producirse en un entorno que a día de hoy todavía está en fase emergente, los robots. Un robot no sólo será afectado por nuevas amenazas derivadas del hardware que utilizan, si no que también estarán en constante amenaza por ciberataques ya bastante conocidos a día de hoy.

Actualmente, además de estos ciberataques llevados a cabo con bastante frecuencia, ROS2, el software más utilizado en el mundo de la robótica será también puesto en el punto de mira. Para realizar la evaluación tanto del sistema como de ROS2 se hará uso de la herramienta *OpenSCAP* que permite tanto la utilización de políticas de seguridad ya definidas como la definición de nuevas y será de gran utilidad para realizar un análisis en profundidad de un sistema robótico durante este proyecto.

Para ello se realizará un experimento en el que se pondrá en evidencia la ventaja de activar la seguridad en ROS2, ya que claramente proporciona mayor fiabilidad al proceso de intercambio de datos, y de manera indirecta deja también en evidencia los beneficios de automatizar el chequeo de si esta activación de la seguridad se ha realizado correctamente.

Summary

Today, with the constant growth of the technology industry and its automation, it is leading not only to the construction of new tools and work systems, but also to the creation of new, more specialized jobs that will unfortunately go away. the hand of new threats that will affect these new computer systems and tools.

The objective of this research project will be to carry out an evaluation of the possible dangers that can occur in an environment that is still in an emerging phase today, robots. A robot will not only be affected by new threats derived from the hardware they use, but will also be constantly threatened by cyberattacks that are already widely known today.

Currently, in addition to these cyberattacks carried out quite frequently, ROS2, the most used software in the world of robotics, will also be put in the spotlight. To perform the evaluation of both the system and ROS2, the OpenSCAP tool will be used, which allows both the use of security policies already defined and the definition of new ones and will be very useful to carry out an in-depth analysis of a robotic system during this project.

To do this, an experiment will be carried out in which the advantage of activating security in ROS2 will be shown, since it clearly provides greater reliability to the data exchange process, and indirectly also reveals the benefits of automating the checking of if this security activation was successful.

Palabras clave

Ciberseguridad, robótica, bastionado, ROS2, OpenSCAP.

Índice general

1. Introducción	3
2. Objetivos, requisitos y metodología	7
2.1. Requisitos	7
2.2. Objetivos	8
2.3. Metodología	8
3. Marco teórico	13
3.1. Linux	13
3.2. ROS y ROS2	14
3.3. DDS y Fast-RTPS	17
3.4. Seguridad en ROS2	18
3.5. OpenSCAP	20
4. Desarrollo del trabajo	23
4.1. Puesta a punto	23
4.1.1. Estructura y detalles del entorno de trabajo	24
4.1.2. Comandos utilizados para los experimentos con SROS2	24
4.2. Trabajando con OpenSCAP	25
4.2.1. Repositorio para el desarrollo del trabajo	27
4.2.2. Uniendo OpenSCAP y ROS2	28
4.2.3. Creando un nuevo perfil de seguridad	29
5. Experimentación y casos de uso	41
5.1. Demo. Poniendo a prueba SROS2	41

ÍNDICE GENERAL

5.2. Demo real. Navegación	45
6. Resultados	53
6.1. Análisis de resultados.	53
7. Conclusiones	59
7.1. Consecución de objetivos	59
7.2. Competencias utilizadas y adquiridas	60
7.3. Trabajos futuros	61
Bibliografía	63

Índice de figuras

1.1.	Robots de uso hospitalario. [1]	4
1.2.	Malware creado por Alias Robotics en el robot UR3. [2]	5
3.1.	Robot TIAGo con sistema operativo Ubuntu 16.04. [3]	14
3.2.	Logotipos ROS Kinetic, Lunar y Melodic. [4]	14
3.3.	Logotipo ROS2 Eloquent.	15
3.4.	Interacción entre nodos ROS.	16
3.5.	Logotipos de ROS y ROS2. [5, 6]	16
3.6.	Arquitectura de ROS2. [4]	17
3.7.	Arquitectura software <i>OpenSCAP</i> . [7]	20
4.1.	Ficheros generados para el nodo simple_node_pub y sub.	25
4.2.	Instalación de <i>SCAP Workbench</i> . [8]	26
4.3.	Estructura de ubuntu1804ros2.	29
4.4.	Ficheros generados por el producto ubuntu1804ros2.	31
4.5.	Ejemplo de fichero generado en build para ubuntu1804ros2.	35
4.6.	Perfil cargado en <i>SCAP Workbench</i> de ubuntu1804ros2.	36
4.7.	Salida de <i>SCAP Workbench</i> después de realizar el escaneo.	37
4.8.	Salida de <i>SCAP Workbench</i> después de realizar el escaneo.	38
4.9.	Informe generado por <i>OpenSCAP</i> .	39
5.1.	Salida por terminal de la ejecución de los nodos simple_node_pub y sub.	44
5.2.	Navegación en ROS2. En el punto inicial	47
5.3.	Navegación en ROS2. Navegando hasta el destino final.	48
5.4.	Navegación en ROS2. Destino alcanzado.	49

ÍNDICE DE FIGURAS

5.5.	Acceso físico a un robot.	50
6.1.	Captura de Wireshark durante la ejecución de los nodos simple_node_pub y sub.	54
6.2.	Captura de Wireshark durante la ejecución de los nodos simple_node_pub y sub con SROS2.	55
6.3.	Medida de la latencia en la comunicación entre nodos sin SROS2.	56
6.4.	Medida de la latencia en la comunicación entre nodos con SROS2.	57

Índice de código

4.1.	Comandos a ejecutar para la utilización de SROS2 en la demo.	24
4.2.	Contenido del fichero group.yml.	30
4.3.	Compilación de ubuntu1804ros2.	30
4.4.	Contenido del fichero rule.yaml.	33
4.5.	Contenido del fichero installed.passh.sh.	33
4.6.	Contenido del fichero noinstalled.fail.sh.	33
4.7.	Contenido del fichero standard.profile.	34
4.8.	Comando de instalación de ntp.	37
5.1.	Nodo simple_node_pub.	42
5.2.	Nodo simple_node_sub.	43
5.3.	Comando para la ejecución de simple_node_pub.	43
5.4.	Comando para la ejecución de simple_node_sub.	44
5.5.	Comandos para la activación de SROS2 en dos terminales.	45

Capítulo 1

Introducción

En la actualidad, el constante crecimiento de la industria unido al constante desarrollo del mundo de la informática crearán un entorno propicio para el nacimiento y desarrollo de nuevas tecnologías que hasta ahora solo habían sido posibles en la gran pantalla. La robótica, ya se conoce a día de hoy como un campo de investigación y desarrollo bastante complejo ya que requiere de otras muchas disciplinas que ayudarán en la creación de robots más avanzados con una mayor facilidad de adaptación al entorno y rapidez en el aprendizaje.

La creación de estos nuevos robots o sistemas robóticos tendrán en muchos casos una finalidad concreta, como la realización de tareas muy específicas o críticas tanto en nuestros hogares como en entornos industriales. El alto nivel de cercanía con los humanos unido a las tareas tan delicadas que se puedan estar llevando a cabo en muchos casos hacen que estos sistemas se conviertan en objetivos de ataques intencionados.

Durante muchos años la protección de los sistemas informáticos ha sido una de las grandes preocupaciones de los especialistas del sector, pero también del personal no especializado, ya que un dispositivo tan pequeño como un reloj podría estar manejando una gran cantidad de datos personales que en ningún caso habrían de ser compartidos con el resto del mundo sin el consentimiento de su propietario. Con la aparición de estos sistemas robóticos el problema del manejo de esta información sensible sigue estando a la orden del día, pero además también se le añade el problema del alto número de sensores y actuadores que estos sistemas ofrecen, ya que ahora el problema no será sólo la información que un robot puede ofrecer si se viese comprometido, si no las tareas que podría llevar a cabo, ya que incluso podría llegar a ocasionar daños físicos en su entorno y poner en riesgo la integridad física de las personas que le rodean,

aumentando esta peligrosidad si el entorno en el que se encuentran trabajando estos robots son entornos críticos, como por ejemplo, robots en plantas nucleares o en centros hospitalarios como los de la Figura 1.1.



Figura 1.1: Robots de uso hospitalario. [1]

El desarrollo de todos estos campos anteriormente mencionados ha producido un crecimiento parejo del campo de la ciberseguridad ya que mientras se estaba avanzando en estas nuevas tecnologías también han ido apareciendo nuevas amenazas o ciberamenazas que al principio podrían tratarse de casos aislados o de bajo impacto pero poco a poco se han convertido en uno de los principales retos a los que se enfrenta el mundo tecnológico.

Afortunadamente en la actualidad no hay muchos ataques conocidos a robots, ya que la mayoría de los que se han producido se han llevado a cabo por investigadores o con la finalidad de concienciar a la población de las vulnerabilidades de los robots más famosos que se pueden alcanzar en el mercado actual. Existen varios ejemplos de estos "hackeos éticos", por ejemplo el llevado a cabo por ingenieros de la consultora IOActive que intentaron hackear a través de su software a uno de los robots más simpáticos que se conocen en el mercado actual, el robot Pepper. Estos ingenieros demostraron que en la versión 2.5.5 de este robot, existía la posibilidad de controlarlo de manera remota e incluso acceder a sus cámaras para espiar a los usuarios. [9] También encontramos el caso de la empresa Alias Robotics, cuyos ingenieros han llevado a cabo una investigación en la cual se ha desarrollado un malware que encripta y bloquea el sistema de uno de los robots más vendidos del mercado (UR3 de Universal Robots) y que termina este secuestro bajo un hipotético pago, ya que este malware en ningún momento ha sido liberado

puesto que sólo se ha realizado con la finalidad de evidenciar la necesidad de proteger estos sistemas de posibles ataques malintencionados. [10, 2]

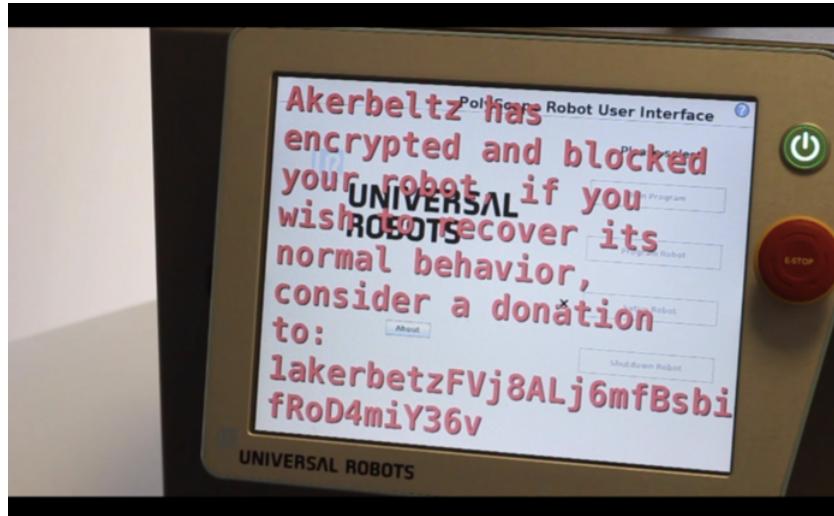


Figura 1.2: Malware creado por Alias Robotics en el robot UR3. [2]

De este problema nace la idea para el desarrollo de este proyecto de investigación, en el que el objetivo será realizar un análisis de las posibles amenazas y vulnerabilidades de un sistema robótico común y cómo podríamos afrontarlas con las herramientas software más comunes que podamos encontrar tanto en el campo de la ciberseguridad como de la robótica.

Al proceso de blindar o proteger la seguridad de los sistemas se le conoce como bastionado o *hardening*. [11] Concretamente este proceso trata de reducir al mínimo el número de vulnerabilidades que tiene un sistema eliminando software, servicios o procesos del sistema que pueden considerarse innecesarios. Estas operaciones de bastionado se pueden realizar de manera manual por un usuario, pero en algunos sistemas dichas operaciones se pueden realizar de manera automática ya que de un sistema a otro las operaciones pueden ser iguales o muy similares. La automatización de este proceso suele ser bastante frecuente en la fase de chequeo o búsqueda de estas vulnerabilidades, es por ello que existen aplicaciones de realizar este proceso de forma automática. Algunos ejemplos de aplicaciones son *Hardentools* (herramienta para realizar un bastionado en Windows de características bastante específicas) o *Microsoft Baseline Security Analyzer* (herramienta que realiza escáneres tanto locales como remotos de actualizaciones de seguridad), [12] además, también existen guías oficiales para realizar un bastionado de calidad para otros sistemas operativos que no sean Windows, por ejemplo para Linux, como se recoge en la página *CIS Security Benchmarks for Linux*. [13]

A día de hoy, la mayoría de desarrollos en el mundo de la robótica se llevan a cabo con ROS, que es un sistema meta-operativo de código abierto y el estándar utilizado en robots. [14] Pero actualmente, debido a diferentes demandas para su utilización y necesidad de mejoras se ha desarrollado una nueva versión conocida como ROS2, que entre las características más resaltadas para el interés de este proyecto cabe mencionar que provee al usuario de un nuevo y mejorado módulo de seguridad más robusto.

Conociendo las herramientas y procesos más habituales para el bastionado de sistemas más conocidos trataremos de realizar ahora nuestro propio bastionado para un sistema más concreto desarrollado en ROS2 con la ayuda de otras herramientas como *OpenSCAP*.

Capítulo 2

Objetivos, requisitos y metodología

En la siguiente sección se llevará a cabo una exposición detallada de los objetivos a alcanzar con la elaboración de este proyecto. Además, también se explicará con detalle las herramientas y requisitos necesarios para la realización del mismo así como la metodología empleada para su desarrollo.

2.1. Requisitos

Como la realización de este Trabajo Fin de Máster se mueve entre los campos de la ciberseguridad y la robótica, para la realización del mismo se requieren los siguientes medios:

- Ordenador con sistema operativo Linux u OS X (preferiblemente Linux). La elección de este sistema operativo viene determinada por las herramientas a utilizar, ya que aunque ROS2 a día de hoy también ofrece soporte para Windows y OS X, Linux ofrece un entorno más seguro y sencillo para la realización y desarrollo de los experimentos.
- ROS2 instalado en el sistema. Ya que uno de los objetivos de este proyecto es securizar el sistema por medio de ROS2 será obligatorio tenerlo instalado.
- *OpenSCAP*. Otro de los objetivos de este proyecto es la creación de normas SCAP, para ello también será un requisito indispensable tener instalada dicha herramienta en el sistema.

2.2. Objetivos

Como se ha mencionado anteriormente, el constante crecimiento de las nuevas tecnologías irán a la par con la aparición de nuevas ciberamenazas y ciberdelitos que con el paso del tiempo irán modernizando sus métodos de ataque y haciendo uso de tecnologías más sofisticadas que podrían causar un daño mayor a la víctima. En estas nuevas y sofisticadas tecnologías nos centraremos en este proyecto, concretamente en el campo de la robótica software, que a día de hoy está viviendo su auge y realizando una gran entrada en el mercado. Por estos motivos, el objetivo principal de este proyecto será realizar una guía práctica de securización de un sistema robótico y crear nuevas normas SCAP que ayuden a automatizar este proceso.

Para abordar este objetivo principal se ha dividido este proyecto en los siguientes subobjetivos más concretos:

- Establecer la importancia de proteger un sistema robótico.
- Utilizar las herramientas software más sofisticadas y estandarizadas actualmente en el campo de la robótica.
- Investigar qué opciones ofrecen dichas herramientas software para ayudar en la securización de estos sistemas.
- Ofrecer una guía práctica lo más generalizada posible que ayude a aplacar la intrusión de posibles amenazas en el sistema.
- Desarrollar nuevas normas SCAP que ayuden en la automatización del proceso de securización.

2.3. Metodología

La metodología empleada para el desarrollo de este proyecto se encuentra definida en los siguientes puntos:

- Familiarización y adaptación al entorno y herramientas software utilizadas. Recopilación y clasificación de toda la información relacionada con el proyecto a realizar para posteriormente proceder con la realización del mismo con pleno conocimiento del marco teórico en el que se situaría este proyecto así como las herramientas a utilizar.

- La descomposición del problema principal en subproblemas más simples ayudarán a que el desarrollo del trabajo sea más ágil y ayudará a una mejor comprensión del mismo.
- Desarrollo secuencial. El desarrollo de este trabajo se llevará a cabo de manera secuencial, ya que hasta que no se haya completado una tarea o subtarea por completo no se podrá continuar con la realización de la siguiente.
- Cuando se hayan completado los hitos o tareas de mayor peso tanto de carácter teórico como práctico se llevará a cabo una fase de experimentación o testeo. En esta fase del desarrollo del proyecto se llevarán a cabo un conjunto de pruebas o experimentos que comprueben que el desarrollo llevado a cabo no de lugar a errores en el sistema y se haya realizado de forma correcta.
- Conclusiones y representación de los resultados obtenidos. Realización de un breve análisis final con el objetivo de reflexionar sobre lo que supone esta aportación al campo de la ciberseguridad y la robótica y los posibles trabajos futuros que podrían realizarse en relación con este proyecto.
- Conclusiones y trabajos futuros. Una vez finalizado el proyecto se llevará a cabo una recapitulación del tema tratado durante el desarrollo del mismo y se realizará una reflexión y análisis final sobre las aportaciones de este trabajo respecto a otros proyectos similares del mismo campo y los posibles trabajos futuros que podrían surgir a partir de este.

Para llevar a cabo el desarrollo del trabajo y la descomposición del problema principal en subproblemas más simples es importante realizar antes un modelado de las amenazas que pueden afectar al sistema de ROS2, ya que será un punto central de este proyecto. Para ello, será interesante separar las amenazas que pueden venir de la parte software asociada a ROS2 y las que podría tener el propio sistema robótico. El modelado realizado será el descrito a continuación.

- Acceso al robot por medio físico.

Un usuario puede acceder físicamente al robot, de manera que podría acceder a este por muchas herramientas directas, por ejemplo introduciendo un dispositivo USB, un cable ethernet, HDMI...etc (dependiendo del tipo de robot con el que se esté trabajando tendrá más o menos puntos de entrada físicos).

- Un robot está compuesto de sensores y actuadores.

Estos sensores se encargan de recolectar la información del exterior que recibe el sistema y que más tarde se procesará para determinar el comportamiento que se llevará a cabo con los actuadores en consecuencia a estos datos .

- Acceso remoto al robot por medio de aplicaciones como SSH y aplicaciones de comunicación en general.
- Utilización de otras aplicaciones remotas con las que poder manejar los datos que recibimos de los sensores del robot.

Por ejemplo el almacenamiento en la nube de dichos datos o el envío de órdenes al robot por medio aplicaciones móviles como por ejemplo el envío de velocidades al robot por medio de una aplicación Android.

- Permisos y configuración del sistema.

Muchas veces el usuario abusa de mantener ciertas configuraciones que vienen predefinidas de fábrica. Esas configuraciones pueden ser contraseñas muy simples que dan acceso a funcionalidades muy importantes del sistema como la contraseña para iniciar sesión en el sistema, para conectarse al robot, permisos de acceso y lectura/escritura de ciertos ficheros y directorios críticos del sistema...etc.

- Configuración incorrecta.

Una mala configuración del sistema puede dar lugar a un elevado número de errores que pueden ser en algunos casos críticos, errores como: mala sincronización de los relojes del sistema, generación de logs erróneos, generación incorrecta de certificados, binarios corruptos.

Si no se tienen en cuenta todas estas debilidades o brechas de seguridad, podría dar lugar a todo tipo de ataques malintencionados, algunos no muy diferentes a los que se podrían producir en un ordenador común, pero otros propios del entorno en el que nos estamos moviendo:

- Publicación de datos incorrectos recogidos por los sensores.
- Creación de un nodo que desarrolle actividades maliciosas en el sistema.
- Publicación de comandos alterados para que el robot lleve a cabo comportamientos erróneos.

- Eliminación y alteración de los nodos que se ejecutan en el robot así como los logs que estos generan.
- Alteración del sistema de ficheros del robot y saturación del sistema de almacenamiento.
- Escucha, interceptación y alteración de los mensajes en los canales de comunicación.
- Eliminación o modificación de los datos guardados en la nube.
- Alteración de los certificados y claves generados para el robot.

Finalmente, para comprobar que realmente todos estos puntos propuestos son correctos se llevará a cabo una experimentación en la cual se realizará una comparación de los escenarios (comunicación entre nodos ROS2) con la seguridad de ROS2 activada y desactivada.

Capítulo 3

Marco teórico

Para comprender bien los objetivos y desarrollo de este proyecto será muy importante conocer el marco teórico en el que se desarrolla. Para ello, a continuación se definirán el conjunto de las herramientas, plataformas y recursos necesarios para el despliegue y desarrollo de dicho proyecto.

3.1. Linux

Linux es uno de los sistemas operativos más utilizados a día de hoy tanto para un usuario con unos conocimientos básicos de informática como para los mejores desarrolladores de aplicaciones. Además, tanto Linux como la mayoría de herramientas a las que se puede acceder desde él son *open source*, lo que provee a este sistema operativo de gran atractivo para todos los usuarios que sepan apreciar tanto su fácil uso como su gran potencia. [15]

Pero este sistema operativo y sus distribuciones ya no se encontrarán instalados en ordenadores "tradicionales" si no que también se encontrarán a bordo de otros sistemas más complejos, como por ejemplo a bordo de robots móviles.



Figura 3.1: Robot TIAgO con sistema operativo Ubuntu 16.04. [3]

3.2. ROS y ROS2

Como ya se ha comentado anteriormente, ROS (*Robot Operating System*) es la plataforma estándar para programación de robots. En sus inicios en 2007, comenzó a desarrollarse por el STAIR (*Stanford Artificial Intelligence*) y a partir del 2008 continuó con su desarrollo en el instituto de investigación de *Willow Garage*. Actualmente, ROS y ROS2 continúa con su desarrollo en la OSRF (*Open Source Robotics Fundation*), aunque siendo código *open source* y aceptando contribuciones de terceros. [16]

Normalmente, se desarrolla una nueva versión de ROS cada año, que está asociada a una versión de Ubuntu. ROS Kinetic, Lunar y Melodic (Figura 3.2) son algunas de sus últimas versiones.



Figura 3.2: Logotipos ROS Kinetic, Lunar y Melodic. [4]

Al mismo tiempo que se desarrollan nuevas versiones de ROS cada año, sucede lo mismo con ROS2, concretamente para este trabajo utilizaremos la versión ROS2 Eloquent que está

asociada al sistema operativo Ubuntu 18.04.

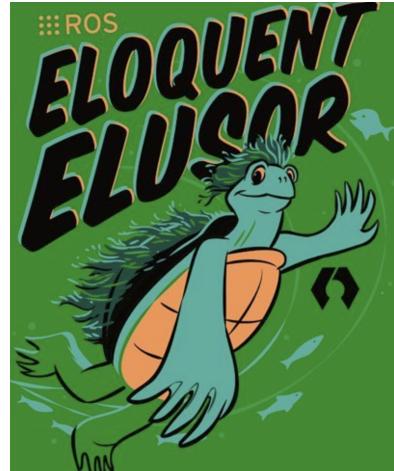


Figura 3.3: Logotipo ROS2 Eloquent.

ROS está compuesto por nodos que se comunican entre sí mediante topics. [17] A continuación se llevará a cabo una pequeña descripción de los elementos más importantes que participan en las comunicaciones ROS:

- **Nodos.** Ficheros ejecutables de ROS que se comunican entre sí con una funcionalidad específica. Estos nodos pueden publicar (publisher) o suscribirse (subscriber) a un topic y además debido a las múltiples librerías cliente que provee ROS estos nodos pueden estar escritos en diferentes lenguajes de programación. [18]
- **Topics.** Canales utilizados por los nodos para realizar un intercambio de mensajes. Este intercambio está configurado por defecto con el protocolo TCP/IP, pero también se puede configurar para realizar las comunicaciones mediante UDP. Cada topic tiene una estructura determinada y puede haber más de un publisher y subscriber por topic, permitiendo una comunicación asíncrona y unidireccional entre nodos. [19]
- **Mensajes.** Estructura de datos utilizada por los nodos para realizar sus comunicaciones a través de los topics.

Un ejemplo de la interacción entre los nodos ROS2 se encuentra representado en la siguiente Figura 3.4.

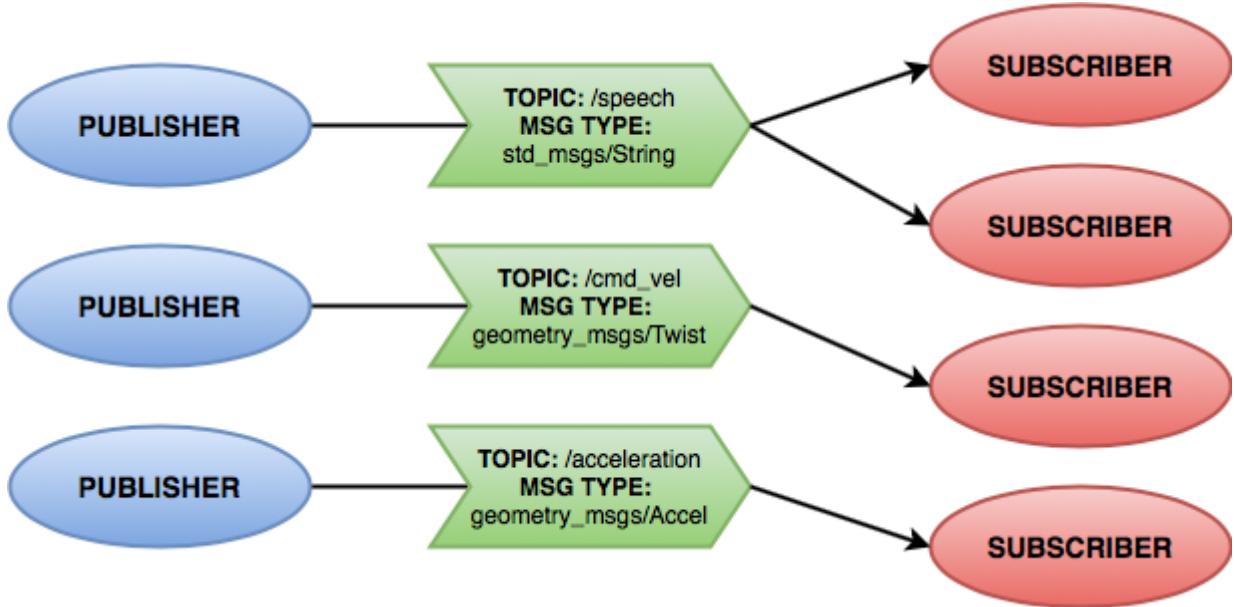


Figura 3.4: Interacción entre nodos ROS.

Una de las principales características que diferencian ROS de ROS2 es la existencia de un master node, conocido como un nodo principal que da servicio de registro para los demás nodos del sistema. En ROS2 este master node desaparece y da paso a DDS que utilizará su propia fase de descubrimiento y al no depender de un único master node la tolerancia a fallos será mayor ya que este sistema de descubrimiento se vuelve distribuido. Mediante la API de DDS, ROS2 obtendrá toda la información de todos los topics, nodos, conexiones etc. [14] La incorporación de DDS provee a ROS2 un mayor nivel de abstracción ya que todas las definiciones de mensajes y otros campos quedan ocultos, permitiendo que el usuario final no tenga que hacer uso del DDS de manera directa aunque es posible que el usuario pueda definir nuevos metadatos para la fase de descubrimiento.



Figura 3.5: Logotipos de ROS y ROS2. [5, 6]

Entre todas estas nuevas funcionalidades y ventajas de ROS2 hay que destacar el nuevo módulo de seguridad de ROS2 y la configuración de QoS¹ que permite configurar el comporta-

¹Quality of Service o calidad de servicio.

miento de las comunicaciones entre nodos y personalizar este sistema lo máximo posible para ajustarse a las necesidades de cada usuario. [20]

3.3. DDS y Fast-RTPS

Para comprender qué es un DDS hay que definir antes otro concepto: *Middleware*. El *Middleware* se define como un software situado entre las capas inferiores y las capas de aplicaciones para conectar el sistema operativo y las aplicaciones que se ejecutan en él. [21] En concreto, para ROS2 se incluye una interfaz middleware que se encarga de las comunicaciones entre la biblioteca de ROS2 y las implementaciones de DDS. [22]

El DDS o *Data Distribution Service* provee al sistema de una fase de descubrimiento, definición y serialización de mensajes y transporte de tipo publicación-suscripción. En resumen, es una capa de middleware de tipo publicación-suscripción que ha sido elegido por ROS2 como capa de su sistema de comunicaciones. [23] Una visión más global de la arquitectura de ROS2 se encuentra representada en la Figura 3.6.

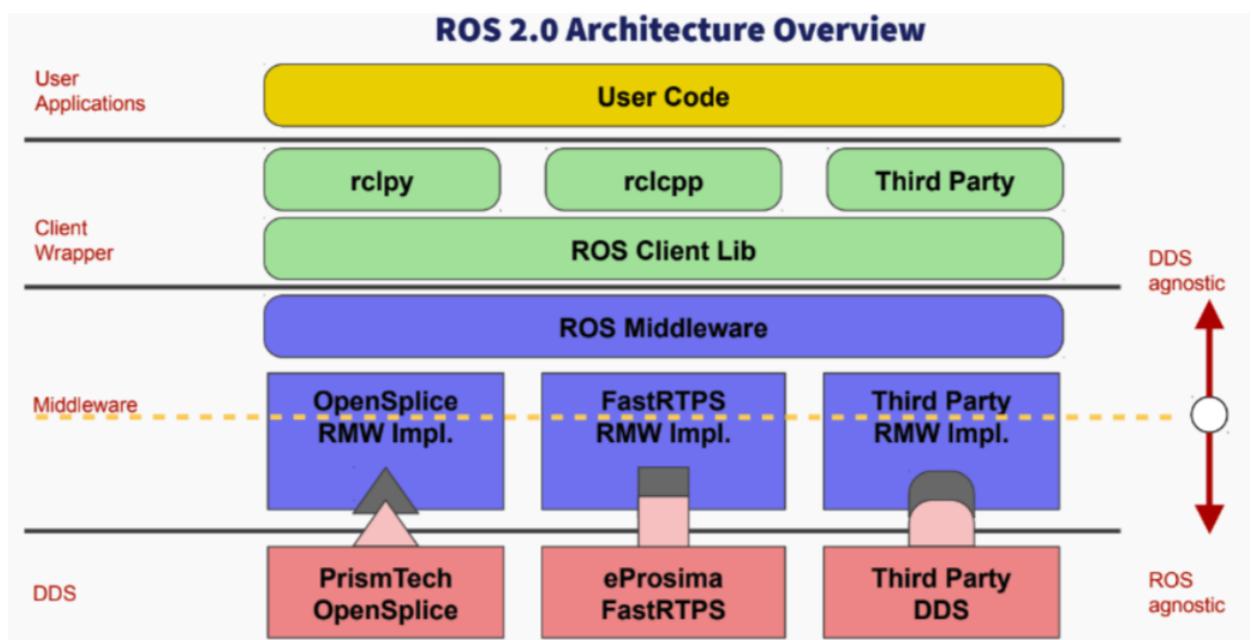


Figura 3.6: Arquitectura de ROS2. [4]

Las implementaciones de DDS más utilizadas en ROS2 son Connext-RTI, OpenSplice, Cyclone y Fast-RTPS que será el adoptado por defecto. [24, 25]

Ya que Fast-RTPS será el DDS a utilizar en este proyecto, hay que resaltar algunos aspectos relevantes de este DDS en cuestión. Fast-RTPS está programado en C++ y es el DDS estándar del OMG (*Object Management Group*), en el que se implementa el protocolo RTPS (*Real Time Publish Subscribe*). Este protocolo provee al sistema de comunicaciones construidas sobre protocolos como UDP y ofrecen al usuario total acceso a la configuración y partes internas de este protocolo. [26]

Para un usuario acostumbrado a trabajar con ROS2 es bastante sencillo cambiar de un DDS a otro, ya que simplemente hay que tener dicho DDS instalado en el sistema y cambiar una variable de entorno. [27] Sin embargo, cada usuario utilizará el DDS que más se adecúe a la aplicación que se esté desarrollando con ROS2. Es decir, la mayoría de estos desarrolladores saben que beneficios puede tener el utilizar un DDS frente a otro y eligen el que más les beneficie, pero de cara al desarrollo en ROS2 y en lo que aportar seguridad a la aplicación se refiere, no es habitual que un usuario intente parchear las posibles fugas que tenga un DDS si no que trabaje en aportar seguridad al sistema desde las capas superiores de ROS2.

Como algunos DDS son de pago y queremos hacer este proyecto de la manera más generalizada posible, el trabajo a desarrollar, como las configuraciones y creación de normas SCAP que se realizarán en adelante serán realizadas con Fast-RTPS como DDS seleccionado.

3.4. Seguridad en ROS2

Generalmente, cuando se habla de ciberseguridad se busca garantizar tres grandes aspectos: confidencialidad, integridad y disponibilidad. Como ya se ha comentado anteriormente, ROS2 utiliza DDS, para el que existe más de un proveedor con varias implementaciones. Para que DDS admita seguridad, se define a través de su especificación DDS-Security (aunque no todas las implementaciones lo admiten). Para ROS2, esta seguridad determinada por el DDS define una arquitectura de *Service Plugin Interface* (SPI). Hay cinco SPIs definidos: autenticación, control de acceso, criptografía, registro y etiquetado de datos, de los cuales actualmente sólo se utilizan las 3 primeras. [28]

- **Autenticación.** Para determinar de manera segura que un nodo es “el que dice ser” y así poder acceder a los topics específicos que le correspondan.

Para realizar esta autenticación ROS2 utiliza el plugin de autenticación llamado *DDS:Auth:PKI*-

DH que utiliza PKI (*Public Key Infrastructure*)². Consiste en la generación de tanto una clave pública como privada de cada participante del dominio además de un certificado x.509³ firmado por una Autoridad Certificadora (AC)⁴ y finalmente se vincula la clave pública a un nombre específico. [28]

- **Control de acceso.** Se ocupa de las restricciones asociadas al DDS de cada participante de un dominio. [28]

ROS2 utiliza el plugin de control de acceso llamado *DDS:Access:Permission* que también utiliza PKI. Cada participante en un dominio necesita dos ficheros para que se realice el control de acceso de una manera correcta:

- "Ficheros de gobierno". Fichero de formato XML firmado en el que se especifica cómo proteger un dominio.
- "Ficheros de permisos". Fichero de formato XML firmado en el que se encuentran los permisos de un participante en un dominio.

Ambos archivos han de estar firmados por una Autoridad Certificadora (AC).

- **Criptografía.** Se maneja todo lo relacionado con la criptografía empleada en las comunicaciones.

ROS2 utiliza el plugin criptográfico *DDS: Crypto:AES-GMC-GMAC* que proporciona al sistema cifrado autenticado utilizando AES⁵-GCM⁶. [28]

En los tres casos se utilizan estos plugins ya que están descritos en la especificación y así la compatibilidad entre DDSs es mayor haciendo uso del mínimo esfuerzo. Por defecto, la seguridad

²PKI se conoce como el conjunto de herramientas tanto software como hardware, así como políticas y protocolos de seguridad que ayudan a garantizar una comunicación segura. [29]

³Un certificado x.509 se emite de manera habitual por una Autoridad Certificadora y se encuentra asociado a una clave pública. [30]

⁴Una Autoridad de Certificadora es una entidad considerada de confianza y responsable que se encarga de la emisión o cancelación de certificados. [31]

⁵AES o *Advanced Encryption Standard* es un sistema de cifrado por bloques que como características a destacar cabe decir que tienen un tamaño fijo de bloque de 128 bits mientras que sus tamaños de clave pueden variar y ser de 128, 192 o 256 bits.

⁶GCM o Contador de Galois es una operación de cifrado de clave simétrica que consiste en cifrar por bloque (AES) con 128 bits en modo contador. Comúnmente también es muy conocido GMAC, que es una variante de GCM utilizada para autenticación

en ROS2 está deshabilitada, para habilitarla habrá que hacer uso de un conjunto de herramientas y características conocidas como *SROS2 (Secure ROS2)* y que permiten la integración con *DDS-Security*. La mayoría del código que se encarga de orquestar toda la habilitación y acople de cada implementación DDS con ROS2 se encuentra en la biblioteca *ROS Client*. [28]

3.5. OpenSCAP

OpenSCAP es una herramienta de código abierto muy utilizada para el análisis de vulnerabilidades en sistemas, concretamente se encarga de realizar un procesamiento automático de las configuraciones de seguridad de un sistema. [32]

SCAP (*Security Content Automation Protocol*) se encarga de recoger y estandarizar las formas de definir y manipular los datos de seguridad de un sistema. Además, como es abierta, la modificación de dichas definiciones podría llevarse a cabo por cualquier usuario. [33]

SCAP se utiliza junto con la herramienta *OpenSCAP* para automatizar y gestionar la seguridad de un sistema, realizando verificaciones, chequeos de vulnerabilidades y cumplimiento de normativas de seguridad. [32]



Figura 3.7: Arquitectura software *OpenSCAP*. [7]

La colección de herramientas que ilustra la anterior imagen (Figura 3.7) y que podemos encontrar al utilizar *OpenSCAP* son las siguientes: [34]

- *OpenSCAP*. Utilización de la línea de comandos para realizar escaneos en un sistema y realizar informes sobre ellos.

- *SCAP Workbench*. Herramienta gráfica utilizada para realizar escaneos de configuración y vulnerabilidades en un sistema e incluso realizar informes partiendo de dichos escaneos.
- *Script Check Engine* (SCE). Permite la creación de contenido de seguridad en lenguajes de script (Bash, Python y Ruby).
- *SCAP Security Guide* (SSG). Conjunto de políticas de seguridad para sistemas Linux. Es una guía práctica de consejos y recomendaciones de buenas prácticas de seguridad ligados al gobierno IT correspondiente.

Cabe resaltar que SCAP incluye también un conjunto de lenguajes específicos utilizados para interpretar políticas de seguridad, los más importantes son:

- XCCDF (*Extensible Configuration Checklist Description Format*). Lenguaje utilizado específicamente para fines descriptivos de la política de seguridad, sin entrar en detalles técnicos.
- OVAL (*Open Vulnerability and Assessment Language*). Utilizado para la realización de chequeos automáticos, recogida de información, análisis de esta y su posterior representación mediante informes detallados.

Para concluir, con *OpenSCAP* se pueden llevar a cabo verificaciones de *checklists* (*Security configuration checklists*) o configuraciones de seguridad que han sido elaboradas por organismos fiables en función del nivel de seguridad que se quiera tener en un sistema y además cada organización puede definir, añadir o modificar dichas configuraciones de seguridad para llevar a cabo una auditoría de sus sistemas más específica.

Capítulo 4

Desarrollo del trabajo

En este capítulo se procederá a dar una explicación detallada del proceso seguido para el desarrollo de este proyecto de investigación.

4.1. Puesta a punto

Como ya se ha mencionado en capítulos anteriores, la perfecta securización de sistemas informáticos a día de hoy es casi imposible ya que intervienen un elevado número de factores que dificultan este proceso. En el caso de los robots el problema de la securización es igual de complejo.

Para las personas familiarizadas con el mundo de la robótica y que en su día a día usan ROS, la herramienta SROS2 será un gran aliado en este proceso de securización del robot.

Afortunadamente ROS cuenta con muchas páginas de documentación, foros de ayuda y consultas que ayudan al usuario no sólo en el desarrollo de aplicaciones si no también a ser más ordenados y poder seguir una guía detallada de cómo llevar a cabo el trabajo, en este caso la utilización de SROS2 (Linux).

Para llevar a cabo una futura fase de experimentación y testeo primero habrá que crear un entorno de pruebas controlado.

4.1.1. Estructura y detalles del entorno de trabajo

El robot en el que desarrollaremos este trabajo estará dotado con Ubuntu 18.04, por lo tanto la versión de ROS2 a utilizar será Eloquent y tendremos 2 directorios o espacios de trabajo. En el primer espacio de trabajo o *workspace* tendremos el código de los nodos que vamos a probar en nuestros experimentos, en el segundo *workspace* tendremos SROS2. Existirá un tercer directorio que no consideraremos espacio de trabajo pero que será de gran importancia ya que en él se guardarán todos los certificados y claves generados por SROS2.

4.1.2. Comandos utilizados para los experimentos con SROS2

En la siguiente página [35] se puede encontrar un tutorial fácil de seguir para probar SROS2. Para nuestro proyecto en concreto es importante que seleccionemos la versión eloquent de ROS2 para entornos Linux.

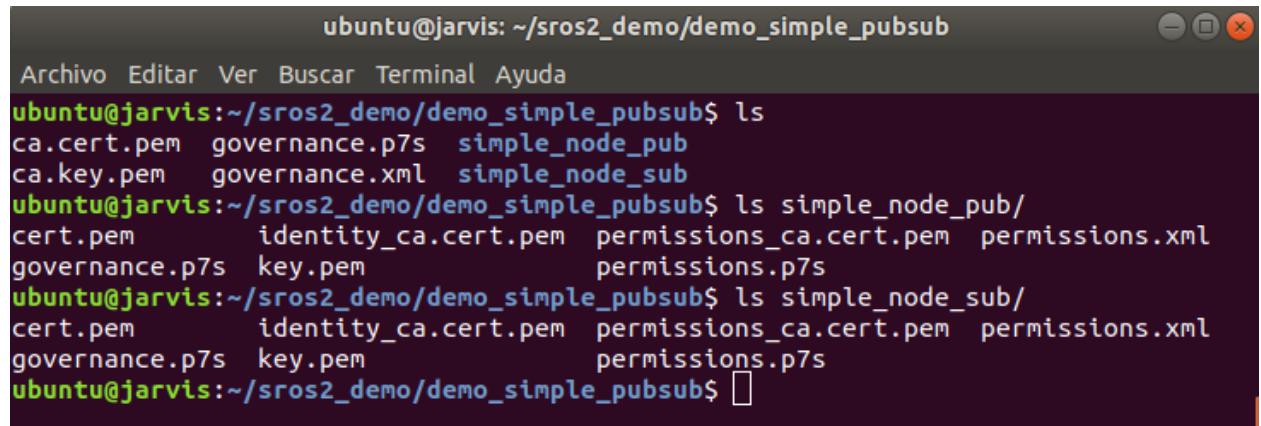
La versión más simple de probar es generar las claves y certificados para dos nodos, un nodo que publicará y otro que se suscribirá a esa información. En otras palabras, el ejemplo que se realizará a continuación proporcionará autenticación y encriptación a las comunicaciones de los dos nodos.

Para esta demo tendremos que ejecutar la siguiente secuencia de comandos en una terminal tal y como indica la guía pero modificando ciertos detalles para este ejemplo en concreto.

```
mkdir ~/sros2_demo
cd ~/sros2_demo
ros2 security create_keystore demo_simple_pubsub
ros2 security create_key demo_simple_pubsub /simple_node_pub
ros2 security create_key demo_simple_pubsub /simple_node_sub
```

Código 4.1: Comandos a ejecutar para la utilización de SROS2 en la demo.

Observamos lo que ha sucedido con la ejecución de estos comandos en la siguiente Figura 4.1:



```
ubuntu@jarvis: ~/sros2_demo/demo_simple_pubsub
Archivo Editar Ver Buscar Terminal Ayuda
ubuntu@jarvis:~/sros2_demo/demo_simple_pubsub$ ls
ca.cert.pem governance.p7s simple_node_pub
ca.key.pem governance.xml simple_node_sub
ubuntu@jarvis:~/sros2_demo/demo_simple_pubsub$ ls simple_node_pub/
cert.pem identity_ca.cert.pem permissions_ca.cert.pem permissions.xml
governance.p7s key.pem permissions.p7s
ubuntu@jarvis:~/sros2_demo/demo_simple_pubsub$ ls simple_node_sub/
cert.pem identity_ca.cert.pem permissions_ca.cert.pem permissions.xml
governance.p7s key.pem permissions.p7s
ubuntu@jarvis:~/sros2_demo/demo_simple_pubsub$ 
```

Figura 4.1: Ficheros generados para el nodo simple_node_pub y sub.

La experimentación que se ha llevado a cabo partiendo de los comandos ejecutados y los ficheros generados se encuentra explicada con detalle en los capítulos posteriores de Experimentación y Resultados. En el caso de que se quisieran generar unos nuevos ficheros o realizar una experimentación con otros nodos y SROS2, el proceso a seguir será el mismo.

4.2. Trabajando con OpenSCAP

Una vez llevado a cabo el modelado del sistema ROS2 y como ya se ha mencionado anteriormente, uno de los objetivos principales de este proyecto será realizar un proceso de chequeo y verificaciones de seguridad de la manera más automática posible. Para ello, de ahora en adelante se trabajará con *OpenSCAP*.

Los pasos a seguir para cumplir este objetivo propuesto será en primer lugar la instalación de las herramientas. En concreto, la herramienta *Workbench*, que además de proporcionar al usuario un entorno gráfico más manejable que la utilización de una *shell*, también ofrece la opción de la crear y customizar políticas de seguridad. El proceso de instalación de la herramienta es bastante simple y está perfectamente explicado en su página oficial como se muestra en la siguiente Figura 4.2. [8]



Figura 4.2: Instalación de *SCAP Workbench*. [8]

La operación que se llevará a cabo con esta herramienta será cargar ficheros de tipo xccdf en formato XML en los que se encuentran diferentes políticas de seguridad y una por una verificar si las normas o reglas definidas se cumplen. De esta manera, el usuario puede hacerse a la idea del nivel de seguridad que tiene el sistema en el instante de la evaluación de seguridad.

Otro de los puntos que también se han mencionado anteriormente en este proyecto es que el robot en cuestión tiene un sistema Ubuntu 18.04, por lo tanto no será necesario crear una política concreta para este robot, si no que ya existen políticas predefinidas para este sistema concreto. Este detalle será de gran relevancia para esta parte del desarrollo, ya que además de las debilidades que puede tener ROS2, que es la herramienta principal que queremos auditar para este trabajo, también podremos realizar chequeos más generales del sistema operativo Ubuntu 18.04. Además este detalle nos será beneficioso a la hora de crear el nuevo producto ya que en lugar de crear uno de cero, heredaremos de uno ya conocido, Ubuntu 18.04.

4.2.1. Repositorio para el desarrollo del trabajo

Para modificar y crear nuevas normas SCAP, trabajaremos a partir del siguiente repositorio de GitHub [36]. El método de trabajo a utilizar será realizar un Fork [37] de este repositorio e ir subiendo los nuevos cambios a una nueva rama llamada ros2 y así tener estos cambios disponibles para cualquier usuario que quiera probarlos. Pero antes de comenzar a trabajar con él, hay que comprender qué contiene este repositorio, cómo y por qué se utiliza [37]:

- La finalidad de este repositorio es crear y mantener políticas de seguridad para diferentes plataformas. [37]
- Está compuesto de contenido SCAP para realizar chequeos de seguridad automáticos, contenido *Ansible* generado a partir de perfiles de seguridad y scripts de remediación bash, generados también a partir de los perfiles de seguridad. [37]
- El contenido de este repositorio se puede utilizar desde diferentes herramientas como oscap y *SCAP Workbench*. [37]

La estructura de este repositorio está dividida en varios directorios. La mayoría de estos directorios son "productos" que corresponden con una plataforma por ejemplo rhel17 (Red Hat Enterprise Linux 7), ubuntu1804... etc. Otros directorios tienen otras finalidades, por ejemplo en el directorio shared se encuentran muchos scripts de remediación y templates ya definidos de los que pueden hacer uso los perfiles de seguridad.

Otro directorio importante a resaltar dada la finalidad de este proyecto será el directorio linux_os/guide. Este directorio es de gran relevancia ya que en él se encuentran la mayoría de verificaciones de seguridad para Linux, estas verificaciones se encuentran divididas según el tipo de comprobación a realizar, permisos, ficheros de logs...etc. En este directo añadiremos nuestras nuevas normas para realizar las comprobaciones relacionadas con ROS2.

Finalmente, en el directorio build, se generará todos los ficheros finales después de compilar, esto es, todas las nuevas normas SCAP, ficheros y scripts de remediación... etc. Se puede llevar a cabo una compilación independiente, es decir, únicamente de una plataforma (mediante el script build_product), o del repositorio entero.

Una vez comprendido todo lo que nos ofrece este repositorio, comenzaremos a trabajar con él. Realizaremos un clone de este repositorio en nuestro sistema, que como ya hemos comentado

es un Ubuntu 18.04. Crearemos un nuevo directorio `ubuntu1804ros2`, que será el nuevo producto sobre el que trabajaremos, crearemos y modificaremos normas SCAP.

4.2.2. Uniendo OpenSCAP y ROS2

La idea central de este proyecto es fusionar la seguridad que nos ofrece ROS2 con la que puede aportarnos el propio sistema operativo del robot. Por ello antes de comenzar con la creación del perfil, es importante tener claro que puntos a tratar son importantes y queremos que maneje nuestro perfil `ubuntu1804ros2` ya que como todo buen sistema informático, a nuestro robot también se le quiere proporcionar integridad, control de acceso y disponibilidad.

- Se chequearán todos los posibles punto de fuga que tenga `ubuntu1804`, por ejemplo, que tenga ciertos servicios activados, permisos de ficheros y directorios críticos en el sistema...etc.
- En lo referente a ROS2, se chequeará que se encuentre instalado en el sistema la versión correcta de ROS2.
- Chequear que SROS2 esté activado, definir ciertos comandos necesarios para que se pueda ejecutar la seguridad antes de lanzar los nodos, que los comandos ejecutados sean correctos y la generación de ficheros o acciones procedentes de la ejecución de estos comandos sea correcta... etc.
- Al realizar este chequeo con *SCAP Workbench* si no se cumplieran todas las reglas definidas en el nuevo perfil habría que solucionarlo de la manera que aconsejan dichas reglas y al completar el chequeo obteniendo el mayor porcentaje de reglas aprobadas comenzar a trabajar con ROS2 y realizar los experimentos y desarrollos que se quieran.

Siguiendo estos puntos y realizando las tareas que se describen en cada apartado estaremos realizando un correcto bastionado de nuestro robot.

4.2.3. Creando un nuevo perfil de seguridad

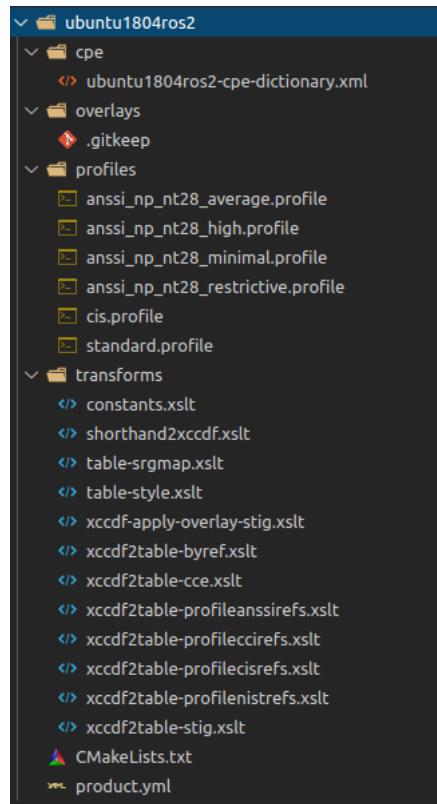


Figura 4.3: Estructura de ubuntu1804ros2.

En la anterior Figura 4.3 se muestra la estructura y los contenidos que va a tener el nuevo perfil de seguridad. La forma de añadir nuevas normas a un perfil será la siguiente:

1. Dirigirse al directorio linux_os/guide.
2. Crear un nuevo directorio dentro de linux_os/guide en el que se encontrarán las nuevas normas utilizadas para los chequeos relacionados con ROS2. Este directorio se llamará ros2system.
3. Crear un nuevo fichero group.yml para que se tenga en cuenta en las compilaciones.

```
documentation_complete: true

title: 'ROS2 System Checks'

description: |
    When you try to use ROS2 technology, you can also
    add the SROS2 security package to the installation.
    The SROS2 package provides security
    to the ROS2 node system,
    adding authentication, encryption and access control.
    With these rules, the check will be carried out that
    the configuration of this package has been
    carried out correctly.
```

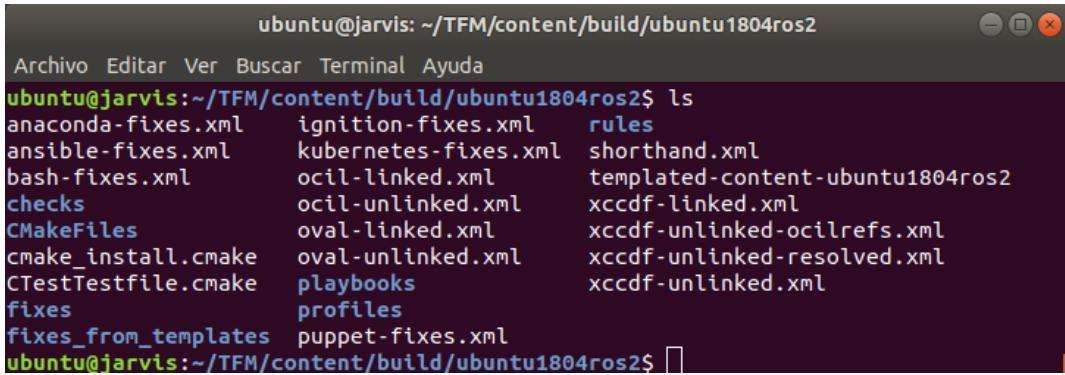
Código 4.2: Contenido del fichero group.yml.

4. Para cada nueva regla se creará a su vez un directorio con un nombre que indique qué hace esa regla y conteniéndola.
5. Estos directorios tendrán obligatoriamente un fichero rule.yml en el que se definirá el propósito de la regla, las referencias, templates, tests y scripts de remediación si es que tienen.
6. Finalmente y para poder utilizar esa nueva regla SCAP habrá que dirigirse al directorio ubuntu1804ros2/profiles y añadir la regla a un perfil.
7. Compilar el producto.

```
cd TFM/content
./build_product ubuntu1804ros2
```

Código 4.3: Compilación de ubuntu1804ros2.

8. Los ficheros .xml resultantes que utilizaremos con *SCAP Workbench* se generarán en el directorio build/ubuntu1804ros2.



```
ubuntu@jarvis: ~/TFM/content/build/ubuntu1804ros2$ ls
anaconda-fixes.xml    ignition-fixes.xml    rules
ansible-fixes.xml     kubernetes-fixes.xml   shorthand.xml
bash-fixes.xml         ocil-linked.xml      templated-content-ubuntu1804ros2
checks                 ocil-unlinked.xml    xccdf-linked.xml
CMakeFiles             oval-linked.xml      xccdf-unlinked-ocilrefs.xml
cmake_install.cmake    oval-unlinked.xml   xccdf-unlinked-resolved.xml
CTestTestfile.cmake    playbooks           xccdf-unlinked.xml
fixes                  profiles
fixes_from_templates   puppet-fixes.xml
ubuntu@jarvis:~/TFM/content/build/ubuntu1804ros2$
```

Figura 4.4: Ficheros generados por el producto ubuntu1804ros2.

En concreto para este proyecto se han creado las siguientes normas relacionadas con ROS2:

- file_owner_ros2_logs
- file_owner_sros2_demo_ca_cert
- file_owner_sros2_demo_ca_key
- file_owner_sros2_demo_governance_p7s
- file_owner_sros2_demo_governance_xml
- file_permissions_sros_demo
- package_ros2_elloquent_installed

La finalidad de crear estas nuevas reglas será demostrar que el sistema tiene instalado ROS2 eloquent y los ficheros que se han generado para proporcionar dicha seguridad no están corruptos y tienen una fiabilidad bastante elevada, en concreto, el motivo de la creación de esta regla es que antes de realizar cualquier chequeo de seguridad relacionado con ROS2, sería necesario comprobar también si ROS2 se encuentra instalado en el sistema, en el caso contrario habría que instalarlo o bien ignorarlo ya que puede ser que el robot en cuestión no esté haciendo uso de esta tecnología.

A continuación se mostrará el proceso que se ha llevado a cabo para la creación de package_ros2_elloquent_installed.

1. Crear el directorio package_ros2_eloquent_installed dentro de linux_os/guide/system/ros2system.
2. Crear el fichero rule.yml y completarlo. Para completarlo habrá que repasar primero la documentación para saber con qué completar cada campo.

```

documentation_complete: true

title: 'Ensure ROS2 Eloquent is Installed'

description: 'ros-eloquent-desktop is installed. {{{
describe_package_install(package="ros-eloquent-desktop") }}}'

rationale: |-
    The ros-eloquent-desktop provides the ROS2 Eloquent
    repositories, RViz, demos and tutorials.

severity: medium

identifiers:
    cce@rhel6: CCE-26809-4
    cce@rhel7: CCE-80187-8
    cce@rhel8: CCE-80847-7

references:
    cis@debian8: 5.1.1
    anssi: BP28 (R5), NT28 (R46)
    cis@rhel8: 4.2.1.1
    disa: CCI-001311, CCI-001312
    hipaa: 164.312(a) (2) (ii)
    iso27001-2013: A.12.4.1, A.12.4.2, A.12.4.3, A.12.4.4, A.12.7.1
    nist: CM-6(a)
    nist-csf: PR.PT-1
    isa-62443-2013: 'SR 2.10, SR 2.11, SR 2.12, SR 2.8, SR 2.9'
    isa-62443-2009: 4.3.3.3.9, 4.3.3.5.8, 4.3.4.4.7,
    4.4.2.1, 4.4.2.2, 4.4.2.4

```

```

cobit5: APO11.04,BAI03.05,DSS05.04,DSS05.07,MEA02.01
cis-csc: 1,14,15,16,3,5,6
srg: SRG-OS-000479-GPOS-00224,SRG-OS-000051-GPOS-00024

ocil_clause: 'the package is not installed'

ocil: '{{ ocil_package(package="ros-eloquent-desktop") }}'

template:
  name: package_installed
  vars:
    pkgname: ros-eloquent-desktop

```

Código 4.4: Contenido del fichero rule.yaml.

3. Crear el directorio tests dentro de package_ros2_eloquent_installed y dos scripts que realizarán la comprobación de si está instalado ROS2 eloquent.

```
apt install ros-eloquent-desktop
```

Código 4.5: Contenido del fichero installed.passh.sh.

```
sudo apt-get remove ros-eloquent-desktop
```

Código 4.6: Contenido del fichero noinstalled.fail.sh.

4. Elegir un perfil de los ya existentes en ubuntu1804ros2/profiles y añadir la regla al fichero, en este caso a standard.profile.

```
documentation_complete: true

title: 'Standard System Security Profile for Ubuntu 18.04'

description: |-
    This profile contains rules to ensure standard security baseline
    of an Ubuntu 18.04 system.

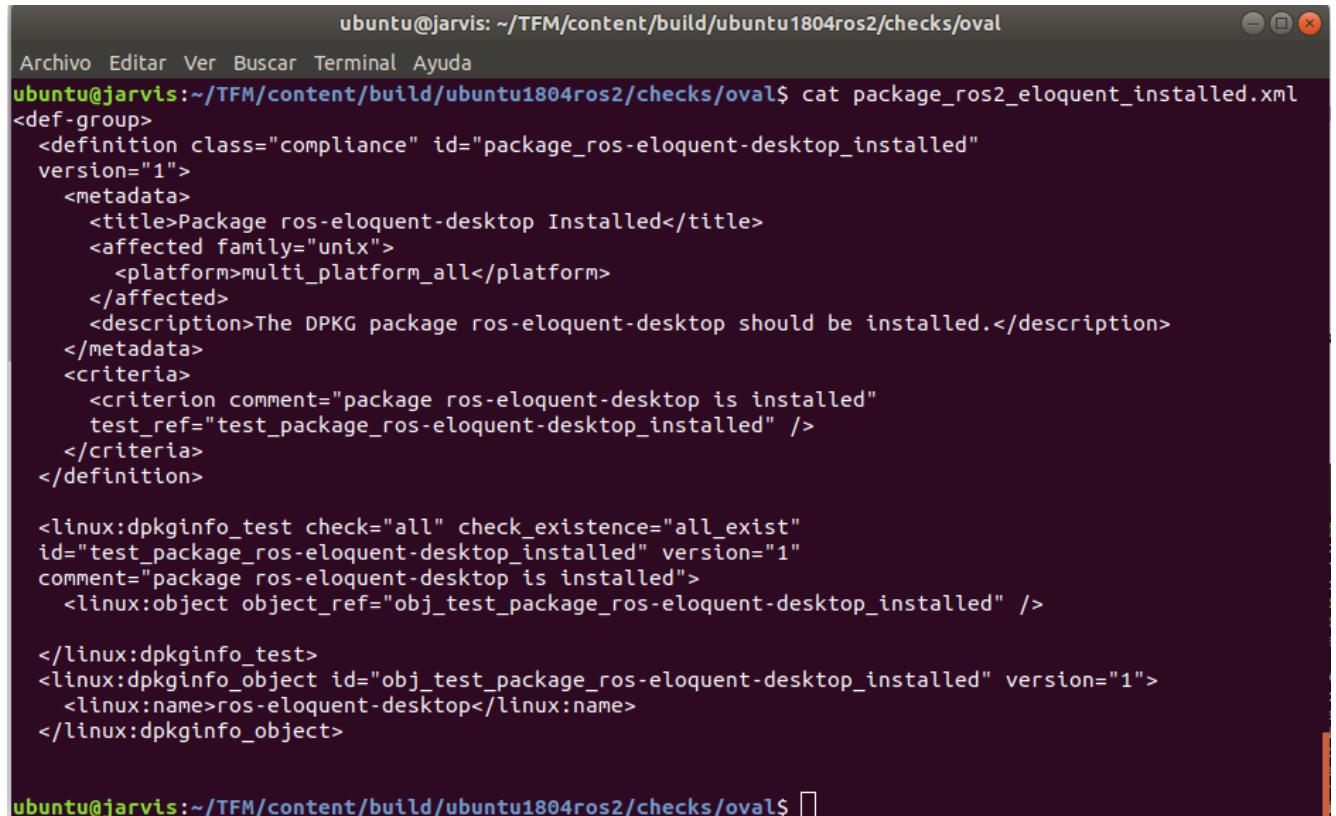
    Regardless of your systems workload all of
    these checks should pass.

selections:
  - partition_for_tmp
  - partition_for_var
  - partition_for_var_log
  - partition_for_var_log_audit
  - partition_for_home
  - package_audit_installed
  - package_cron_installed
  - package_ntp_installed
  - package_rsyslog_installed

  ..
  ..

  - sysctl_fs_protected_symlinks
  - sysctl_fs_protected_hardlinks
  - sysctl_fs_suid_dumpable
  - sysctl_kernel_randomize_va_space
  - file_owner_ros2_logs
  - file_owner_sros2_demo_ca_cert
  - file_owner_sros2_demo_ca_key
  - file_owner_sros2_demo_governance_p7s
  - file_owner_sros2_demo_governance_xml
  - file_permissions_sros2_demo
  - package_ros2_elloquent_installed
```

5. Compilar y ver que no se produce ningún error de compilación y se han generado los ficheros correctos. (Por ejemplo comprobar que en el directorio build se encuentra el fichero que se muestra en la Figura 4.5).



```

ubuntu@jarvis: ~/TFM/content/build/ubuntu1804ros2/checks/oval$ cat package_ros2_elquent_installed.xml
<def-group>
  <definition class="compliance" id="package_ros-eloquent-desktop_installed"
  version="1">
    <metadata>
      <title>Package ros-eloquent-desktop Installed</title>
      <affected family="unix">
        <platform>multi_platform_all</platform>
      </affected>
      <description>The DPKG package ros-eloquent-desktop should be installed.</description>
    </metadata>
    <criteria>
      <criterion comment="package ros-eloquent-desktop is installed"
      test_ref="test_package_ros-eloquent-desktop_installed" />
    </criteria>
  </definition>

  <linux:dpkginfo_test check="all" check_existence="all_exist"
  id="test_package_ros-eloquent-desktop_installed" version="1"
  comment="package ros-eloquent-desktop is installed">
    <linux:object object_ref="obj_test_package_ros-eloquent-desktop_installed" />
  </linux:dpkginfo_test>
  <linux:dpkginfo_object id="obj_test_package_ros-eloquent-desktop_installed" version="1">
    <linux:name>ros-eloquent-desktop</linux:name>
  </linux:dpkginfo_object>
</linux:dpkginfo_test>
</linux:dpkginfo_object id="obj_test_package_ros-eloquent-desktop_installed" version="1">
  <linux:name>ros-eloquent-desktop</linux:name>
</linux:dpkginfo_object>

```

Figura 4.5: Ejemplo de fichero generado en build para ubuntu1804ros2.

Se realizarán estas mismas operaciones para cada regla, para finalmente cargar el perfil en la herramienta *SCAP Workbench* y seleccionar el perfil al que hayamos añadido las nuevas normas, en este caso *Standard System Security Profile* como se muestra en la Figura 4.6.¹

¹El código con todas las modificaciones realizadas para este apartado del proyecto se encuentra disponible en GitHub, rama ros2. [37]

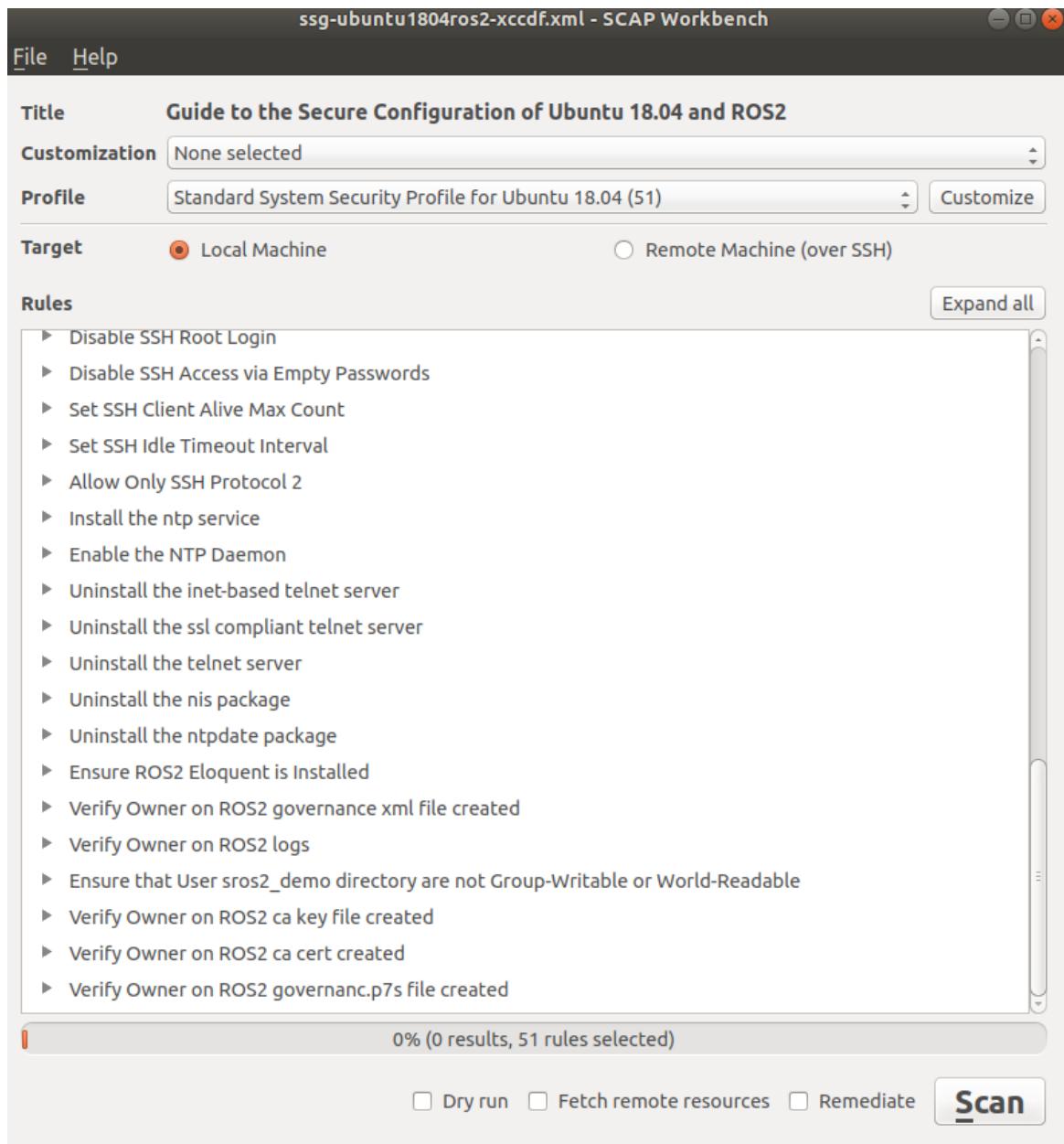


Figura 4.6: Perfil cargado en *SCAP Workbench* de *ubuntu1804ros2*.

Una vez cargado el perfil, pulsar el botón Scan y esperar a que la herramienta nos muestre una salida con los resultados del chequeo de reglas como se muestra en la siguiente Figura 4.7 y 4.8.

▶ Verify Permissions on group File	pass
▶ Disable Core Dumps for SUID programs	fail
▼ Enable Randomized Layout of Virtual Address Space	fail
To set the runtime status of the kernel.randomize_va_space kernel parameter, run the following command: \$ sudo sysctl -w kernel.randomize_va_space=2 To make sure that the setting is persistent, add the following line to a file in the directory /etc/sysctl.d: kernel.randomize_va_space = 2	
▶ Set SSH Client Alive Max Count	pass
▶ Allow Only SSH Protocol 2	pass
▶ Disable SSH Root Login	pass
▶ Set SSH Idle Timeout Interval	pass
▶ Disable SSH Access via Empty Passwords	pass
▶ Uninstall the ntpdate package	pass
▶ Uninstall the ssl compliant telnet server	pass
▶ Uninstall the inet-based telnet server	pass
▶ Uninstall the nis package	pass
▶ Uninstall the telnet server	pass
▶ Disable unauthenticated repositories in APT configuration	pass
▼ Install the ntp service	fail
The ntpd service should be installed.	

Figura 4.7: Salida de *SCAP Workbench* después de realizar el escaneo.

En este caso, como se puede comprobar en la figura anterior el resultado de realizar este chequeo no es del todo satisfactorio, ya que el estado del sistema en algunos casos no se considera el correcto según la política que hemos cargado con *SCAP Workbench*. Además en dicha salida se nos dan detalles también de cómo podríamos mejorar ese aspecto del sistema para que pase los chequeos de seguridad en un futuro. Por ejemplo instalando *ntp service* como se recomienda...

```
sudo apt-get install ntp
```

Código 4.8: Comando de instalación de ntp.

▶ Verify User Who Owns shadow File	pass
▶ Verify Permissions on gshadow File	pass
▶ Verify Group Who Owns passwd File	pass
▶ Verify User Who Owns gshadow File	pass
▶ Verify Permissions on group File	pass
▶ Disable Core Dumps for SUID programs	fail
▶ Enable Randomized Layout of Virtual Address Space	fail
▶ Set SSH Client Alive Max Count	pass
▶ Allow Only SSH Protocol 2	pass
▶ Disable SSH Root Login	pass
▶ Set SSH Idle Timeout Interval	pass
▶ Disable SSH Access via Empty Passwords	pass
▶ Uninstall the ntpdate package	pass
▶ Uninstall the ssl compliant telnet server	pass
▶ Uninstall the inet-based telnet server	pass
▶ Uninstall the nis package	pass
▶ Uninstall the telnet server	pass
▶ Disable unauthenticated repositories in APT configuration	pass
▶ Install the ntp service	pass

Figura 4.8: Salida de SCAP Workbench después de realizar el escaneo.

Observamos que ahora el estado de esa norma ahora se encuentra en verde ya que ha pasado la verificación. Lo correcto sería hacer lo mismo con todas las normas en estado *fail* hasta que se cumplan todas.

También se puede obtener el presente informe de manera más detallada pulsando *Save Results*, que nos llevará a una página como la que se muestra en la siguiente figura en la que se encuentra todo más detallado y de una manera más amistosa y legible para el usuario (Figura 4.9).

file:///tmp/qt_temp.d10475.html		
Ensure /var/log/audit Located On Separate Partition	low	fail
▶ Sudo		
▼ Configure Syslog 1x fail 3x notchecked		
▼ Ensure Proper Configuration of Log Files 3x notchecked		
Ensure Log Files Are Owned By Appropriate User	medium	notchecked
Ensure System Log Files Have Correct Permissions	medium	notchecked
Ensure Log Files Are Owned By Appropriate Group	medium	notchecked
▼ Ensure All Logs are Rotated by logrotate 1x fail		
Ensure Logrotate Runs Periodically	medium	fail
▼ File Permissions and Masks 2x fail		
Verify Permissions on Important Files and Directories		
▼ Restrict Programs from Dangerous Execution Patterns 2x fail		
▼ Disable Core Dumps 1x fail		
Disable Core Dumps for SUID programs	medium	fail
▼ Enable ExecShield 1x fail		
Enable Randomized Layout of Virtual Address Space	medium	fail
▼ Services 1x fail		
▶ SSH Server		
▶ Deprecated services		
▶ APT service configuration		
▼ Network Time Protocol 1x fail		
Install the ntp service	high	fail
Show all result details		

Figura 4.9: Informe generado por *OpenSCAP*.

Capítulo 5

Experimentación y casos de uso

En este capítulo se llevará a cabo una explicación detallada de los experimentos realizados con el objetivo de sacar unas conclusiones precisas acerca de las aportaciones desempeñadas mediante la realización de este trabajo.

Para la realización de los experimentos descritos a continuación será necesario crear un *workspace* en el que tengamos el código que vamos a utilizar para las demos, este código a su vez estará dividido en paquetes dependiendo de la demo que vayamos a realizar, por ejemplo, en este *workspace* habrá un paquete en el que se encontrará el código para la demo con dos nodos de tipo publicador-subscriptor.

Con este *workspace* ya podríamos realizar demos completas tanto de ejecuciones básicas y simples como de ejecuciones y tareas complejas que pueden ser llevadas a cabo por un robot mediante ROS2, pero para poner a prueba la seguridad que proporciona ROS2 necesitaremos crear otros dos *workspaces* adicionales, en el que uno se encontrará el código necesario para ejecutar todo lo referente a la seguridad en ROS2 y en el otro se guardará todo lo generado (claves, certificados... etc) por dicha seguridad.

5.1. Demo. Poniendo a prueba SROS2

La finalidad de esta primera demo será dejar constancia con el mínimo número de nodos que la seguridad en ROS2 funciona, pero también tiene brechas. Para ello vamos a hacer uso de dos nodos más simples posibles.

El primer nodo se encargará de la publicación de un string en el topic chatter:

```
#include "rclcpp/rclcpp.hpp"
#include "std_msgs/msg/string.hpp"

using namespace std::chrono_literals;

int main(int argc, char * argv[])
{
    rclcpp::init(argc, argv);

    auto node = rclcpp::Node::make_shared("simple_node_pub");
    auto publisher = node->create_publisher<std_msgs::msg::String>(
        "chatter", 10);

    std_msgs::msg::String message;
    int counter = 0;

    rclcpp::Rate loop_rate(500ms);
    while (rclcpp::ok()) {
        message.data = "Hello, world! " + std::to_string(counter++);
        RCLCPP_INFO(node->get_logger(),
                    "Publishing [%s]", message.data.c_str());

        publisher->publish(message);

        rclcpp::spin_some(node);
        loop_rate.sleep();
    }

    rclcpp::shutdown();

    return 0;
}
```

Código 5.1: Nodo simple_node_pub.

El segundo nodo se encargará de leer del topic chatter:

```
#include "rclcpp/rclcpp.hpp"
#include "std_msgs/msg/string.hpp"

rclcpp::Node::SharedPtr node = nullptr;

void callback(const std_msgs::msg::String::SharedPtr msg)
{
    RCLCPP_INFO(node->get_logger(), "I heard: [%s]", msg->data.c_str());
}

int main(int argc, char * argv[])
{
    rclcpp::init(argc, argv);

    node = rclcpp::Node::make_shared("simple_node_sub");
    auto subscription = node->create_subscription<std_msgs::msg::String>(
        "chatter", 10, callback);

    rclcpp::spin(node);

    rclcpp::shutdown();

    return 0;
}
```

Código 5.2: Nodo simple_node_sub.

Una vez tengamos los nodos compilados, comprobamos que funcionan, para ello los ejecutamos de la siguiente forma en terminales separadas:

<code>ros2 run intro_ros2 simple_node_pub</code>
--

Código 5.3: Comando para la ejecución de simple_node_pub.

```
ros2 run intro_ros2 simple_node_sub
```

Código 5.4: Comando para la ejecución de simple_node_sub.

Y obtendremos las siguientes salidas representadas en la siguiente figura:

The figure consists of two vertically stacked terminal windows, both titled "ubuntu@jarvis: ~".

Top Terminal Window (simple_node_pub):

```
[INFO] [simple_node_pub]: Publishing [Hello, world! 83]
[INFO] [simple_node_pub]: Publishing [Hello, world! 84]
[INFO] [simple_node_pub]: Publishing [Hello, world! 85]
[INFO] [simple_node_pub]: Publishing [Hello, world! 86]
[INFO] [simple_node_pub]: Publishing [Hello, world! 87]
[INFO] [simple_node_pub]: Publishing [Hello, world! 88]
[INFO] [simple_node_pub]: Publishing [Hello, world! 89]
[INFO] [simple_node_pub]: Publishing [Hello, world! 90]
[INFO] [simple_node_pub]: Publishing [Hello, world! 91]
[INFO] [simple_node_pub]: Publishing [Hello, world! 92]
[INFO] [simple_node_pub]: Publishing [Hello, world! 93]
[INFO] [simple_node_pub]: Publishing [Hello, world! 94]
[INFO] [simple_node_pub]: Publishing [Hello, world! 95]
[INFO] [simple_node_pub]: Publishing [Hello, world! 96]
[INFO] [simple_node_pub]: Publishing [Hello, world! 97]
[INFO] [simple_node_pub]: Publishing [Hello, world! 98]
[INFO] [simple_node_pub]: Publishing [Hello, world! 99]
[INFO] [simple_node_pub]: Publishing [Hello, world! 100]
[INFO] [simple_node_pub]: Publishing [Hello, world! 101]
[INFO] [simple_node_pub]: Publishing [Hello, world! 102]
[INFO] [simple_node_pub]: Publishing [Hello, world! 103]
[INFO] [simple_node_pub]: Publishing [Hello, world! 104]
^C[INFO] [rclcpp]: signal_handler(signal_value=2)
ubuntu@jarvis:~$
```

Bottom Terminal Window (simple_node_sub):

```
[INFO] [simple_node_sub]: I heard: [Hello, world! 83]
[INFO] [simple_node_sub]: I heard: [Hello, world! 84]
[INFO] [simple_node_sub]: I heard: [Hello, world! 85]
[INFO] [simple_node_sub]: I heard: [Hello, world! 86]
[INFO] [simple_node_sub]: I heard: [Hello, world! 87]
[INFO] [simple_node_sub]: I heard: [Hello, world! 88]
[INFO] [simple_node_sub]: I heard: [Hello, world! 89]
[INFO] [simple_node_sub]: I heard: [Hello, world! 90]
[INFO] [simple_node_sub]: I heard: [Hello, world! 91]
[INFO] [simple_node_sub]: I heard: [Hello, world! 92]
[INFO] [simple_node_sub]: I heard: [Hello, world! 93]
[INFO] [simple_node_sub]: I heard: [Hello, world! 94]
[INFO] [simple_node_sub]: I heard: [Hello, world! 95]
[INFO] [simple_node_sub]: I heard: [Hello, world! 96]
[INFO] [simple_node_sub]: I heard: [Hello, world! 97]
[INFO] [simple_node_sub]: I heard: [Hello, world! 98]
[INFO] [simple_node_sub]: I heard: [Hello, world! 99]
[INFO] [simple_node_sub]: I heard: [Hello, world! 100]
[INFO] [simple_node_sub]: I heard: [Hello, world! 101]
[INFO] [simple_node_sub]: I heard: [Hello, world! 102]
[INFO] [simple_node_sub]: I heard: [Hello, world! 103]
[INFO] [simple_node_sub]: I heard: [Hello, world! 104]
^C[INFO] [rclcpp]: signal_handler(signal_value=2)
ubuntu@jarvis:~$
```

Figura 5.1: Salida por terminal de la ejecución de los nodos simple_node_pub y sub.

En los pasos anteriores hemos ejecutado los nodos de ROS2 sin seguridad, pero ahora para

con la ayuda de los perfiles de *OpenSCAP* que se han definido previamente en capítulos anteriores, simplemente con la ejecución del nuevo perfil en el que se han definido los siguientes comandos que se muestran en el Cuadro 5.5 podemos estar seguros de que estamos haciendo uso de SROS2.

```
export ROS_SECURITY_ROOT_DIRECTORY=~/sros2_demo/demo_simple_pubsub
export ROS_SECURITY_ENABLE=true
export ROS_SECURITY_STRATEGY=Enforce
ros2 run intro_ros2 simple_node_pub
export ROS_SECURITY_ROOT_DIRECTORY=~/sros2_demo/demo_simple_pubsub
export ROS_SECURITY_ENABLE=true
export ROS_SECURITY_STRATEGY=Enforce
ros2 run intro_ros2 simple_node_sub
```

Código 5.5: Comandos para la activación de SROS2 en dos terminales.

Estamos seguros de que estos cambios tomarán efecto, ya que en el nuevo perfil de *OpenSCAP* también se han añadido normas que comprueban que la generación de estos certificados y claves se han creado correctamente antes de ejecutar los comandos del Cuadro 5.5.

La salida por terminal obtenida será idéntica a la anterior, sin embargo, si observamos la nueva captura de tráfico que hemos tomado con Wireshark podemos comprobar que existen diferencias relacionadas con la seguridad. Estas diferencias y el consiguiente análisis de estos resultados y lo que conlleva será realizado en el siguiente capítulo.

En esta demo, hemos podido comprobar la eficacia de la utilización de SROS2 en el ámbito de la ciberseguridad, sin embargo es un ejemplo poco ilustrativo, ya que observar una simple comunicación entre dos nodos "que se saludan" no es un problema del todo real en el mundo de la robótica software.

5.2. Demo real. Navegación

Pongamos un escenario más real, imaginemos que somos propietarios de un bar en el que tenemos como empleado un robot camarero que realiza las entregas de los pedidos realizados por los clientes. Probablemente no sea el entorno más crítico, pero en este escenario se ponen en

riesgo tanto la integridad del robot como de las personas que lo rodean, ya que si se produjera una navegación demasiado inestable las consecuencias podrían ser de elevado coste.

En la siguiente figura se puede ver una simulación de este escenario, concretamente cómo funcionaría la navegación de un robot con ROS2 (para este proyecto, a priori el funcionamiento de la navegación de ROS2 es irrelevante, los detalles que adquieren mayor relevancia respecto al proyecto será saber que la navegación de ROS2 se compone de un conjunto de nodos que interactúan entre sí para ofrecer el movimiento del robot por el entorno de la manera más precisa, determinista y controlada posible, para ello los factores que más intervienen será el manejo de los datos obtenidos por el láser y la posición del robot respecto al mapa que se ha generado previamente o sobre la marcha mientras el robot navega y hasta que alcanza su destino)¹.

Como se puede observar, con la ayuda de gazebo (paquete de ROS2 que simula un entorno virtual controlado en el que se pueden realizar experimentaciones sin hacer uso del robot real) podemos simular un robot, que se sitúa en un punto del entorno y mapa (izquierda) y con la herramienta rviz2² (derecha) podemos añadir toda la información que ofrece este robot y que se quiera visualizar, por ejemplo, lecturas del láser, punto en el que está situado y al que se dirige y la trayectoria que va a llevar a cabo...etc.

¹Para cualquier lector que desee saber más acerca de la navegación en ROS2, puede consultar su página de Github. [38]

²Herramienta de ROS2 para la visualización en 3D de los datos que genera un sistema de nodos ROS2. En ella se pueden seleccionar los datos que quieras o no visualizar e incluso interactuar con los nodos ROS2 en ejecución, estableciendo por ejemplo el punto inicial en el que se encuentra un robot en el mapa o visualizando la salida de la cámara de un robot.

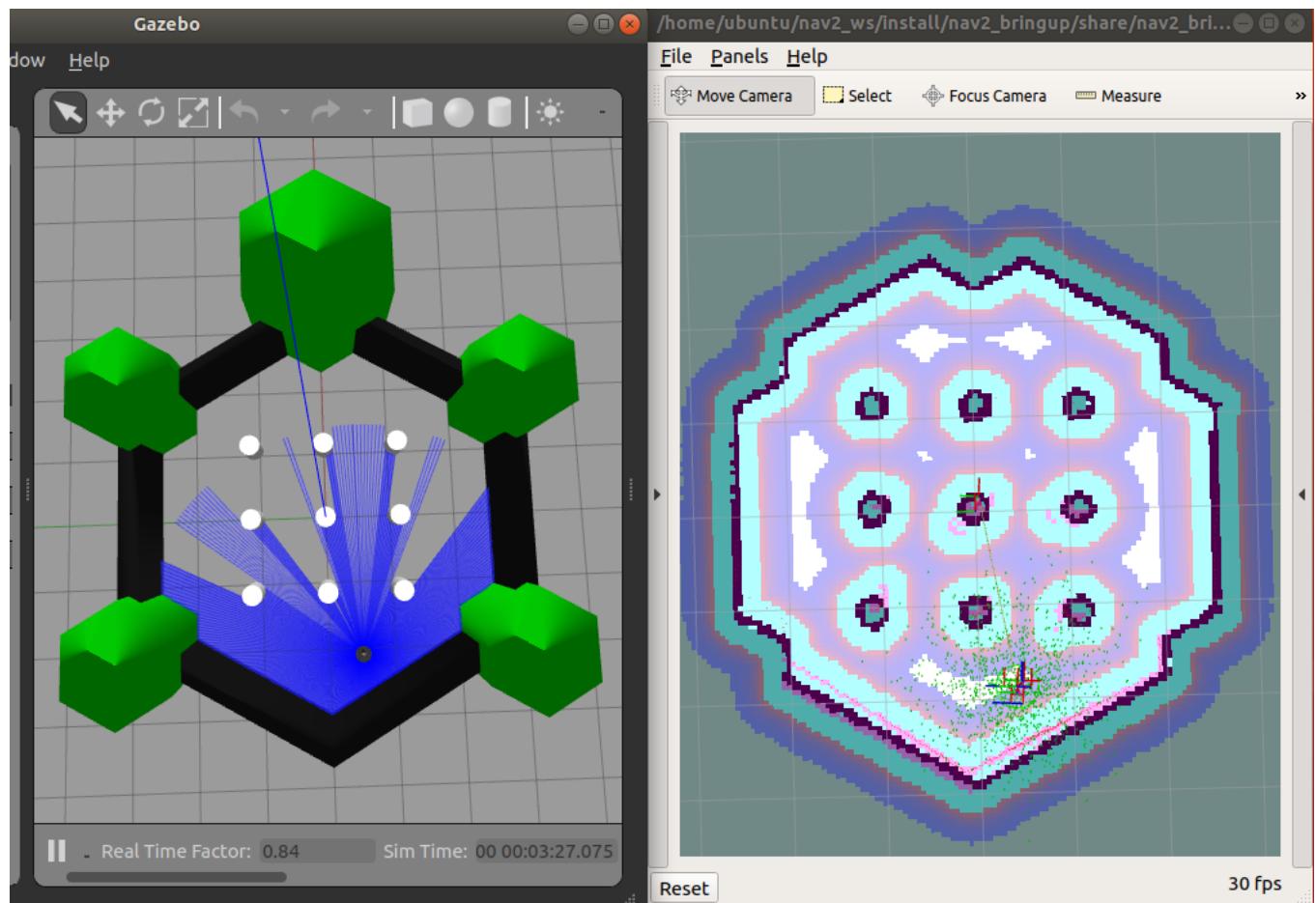


Figura 5.2: Navegación en ROS2. En el punto inicial

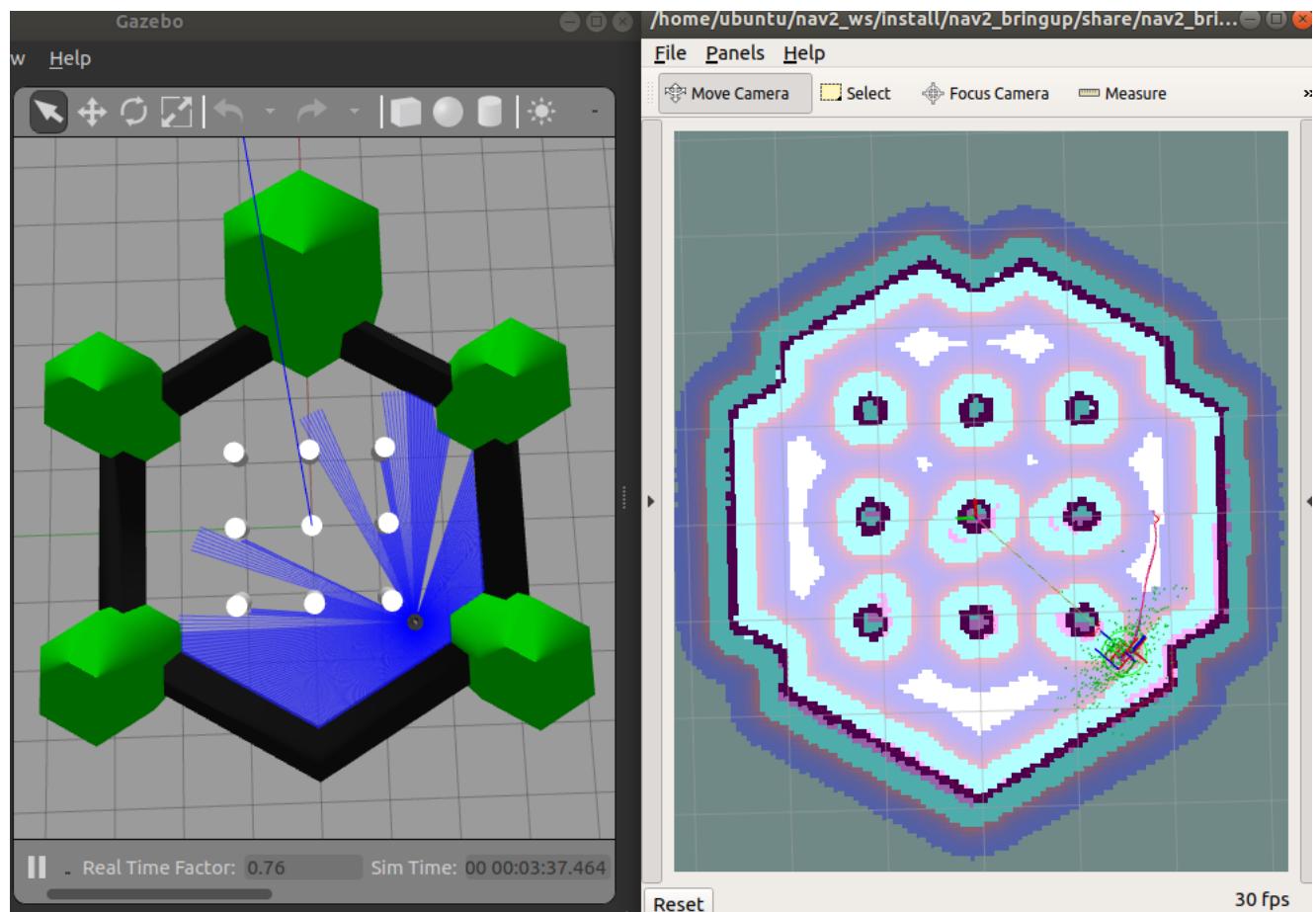


Figura 5.3: Navegación en ROS2. Navegando hasta el destino final.

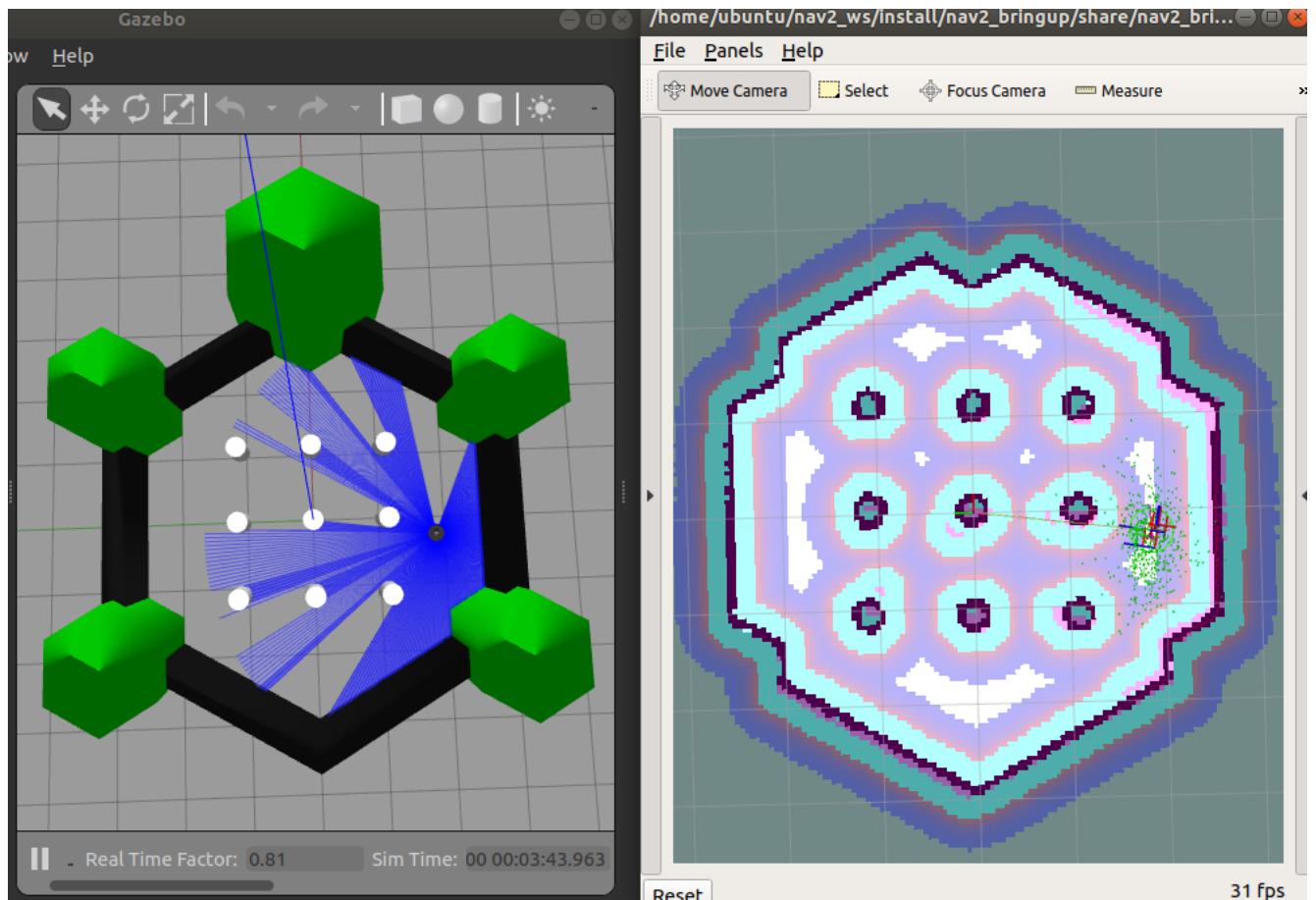


Figura 5.4: Navegación en ROS2. Destino alcanzado.

En las tres figuras anteriores (Figura 5.2, 5.3, 5.4) se pueden ver algunos de los elementos que intervienen en la navegación y ayuda a dejar en evidencia por qué estos elementos son tan importantes de cara a proteger nuestro robot. Procedamos a realizar un análisis de la información del robot que se muestra en estas figuras desde el punto de vista de un atacante.

Procesar y enviar toda esta información sin las medidas de protección adecuadas un atacante podría:

- Modificar las lecturas del láser, de manera que el robot recalculase su ruta y se desplazase por un terreno ocupado por objetos o personas.
- Modificar el punto de origen del robot. Si el robot se encuentra deslocalizado se podría realizar una navegación errónea por terreno ocupado.
- Modificar el punto de destino del robot. El robot se dirigiría a un destino erróneo.

- En el caso de tener cámara, un atacante podría acceder a las imágenes que estuviese registrando dicho robot mientras realiza su trabajo y utilizarlas para su propio beneficio, por ejemplo espionaje industrial.

Todas estas ideas son motivo más que suficiente para intentar proteger la información que está tratando este robot para el caso concreto de la navegación. Aplicar SROS2 en los nodos que intervienen en dicho proceso podría ser un buen comienzo para proteger esa información, ya que de entrada SROS2 hace que las comunicaciones necesarias para realizar este proceso sean encriptadas, o que un atacante externo no pueda ver en qué topics se encuentra esta información.

Pero, ¿es SROS2 infalible? La respuesta es no.

Con la primera demo hemos comprobado el correcto funcionamiento de SROS2, pero es posible que un atacante pudiese hacer que SROS2 no funcione tan bien si tenemos en cuenta los siguientes puntos:

- A un robot se puede acceder de manera remota (por ejemplo ssh).
- A un robot se puede acceder físicamente (Figura 5.5).



Figura 5.5: Acceso físico a un robot.

Si se diesen alguna intrusión mediante alguno de estos dos puntos habría que:

- Comprobar los directorios y *workspaces* generados así como sus permisos de acceso.
- Comprobar si los certificados que se han generado con SROS2 son correctos y tienen los permisos necesarios.
- Comprobar si los certificados generados por SROS2 los han generado las entidades correctas.
- Comprobar si los *logs* generados al utilizar ROS2 son normales o han sido alterados y está ocurriendo algo fuera de lo común.

Aquí entra en juego *OpenSCAP* que nos ayudará a realizar estas comprobaciones así como otras comprobaciones típicas de un sistema Ubuntu de manera automática.

Capítulo 6

Resultados

En este capítulo se realizará un análisis en detalle del resultado de la ejecución de las demos del capítulo de experimentación.

6.1. Análisis de resultados.

En el Capítulo 5 se ha llevado a cabo un experimento en el que se realizaba una comunicación entre nodos ROS2 con la seguridad desactivada en el primer caso y luego activando SROS2.

Durante la ejecución de estos nodos, se ejecutó la herramienta Wireshark para realizar una captura de los paquetes intercambiados durante la comunicación de estos dos nodos. La realización de esta captura será interesante ya que ahora no estamos utilizando seguridad, por lo tanto, los paquetes con la información del topic será enviada en claro, sin ningún tipo de medidas de protección. Seleccionando un paquete al azar en la captura realizada podemos comprobar que esta última afirmación es cierta (Figura 6.1):

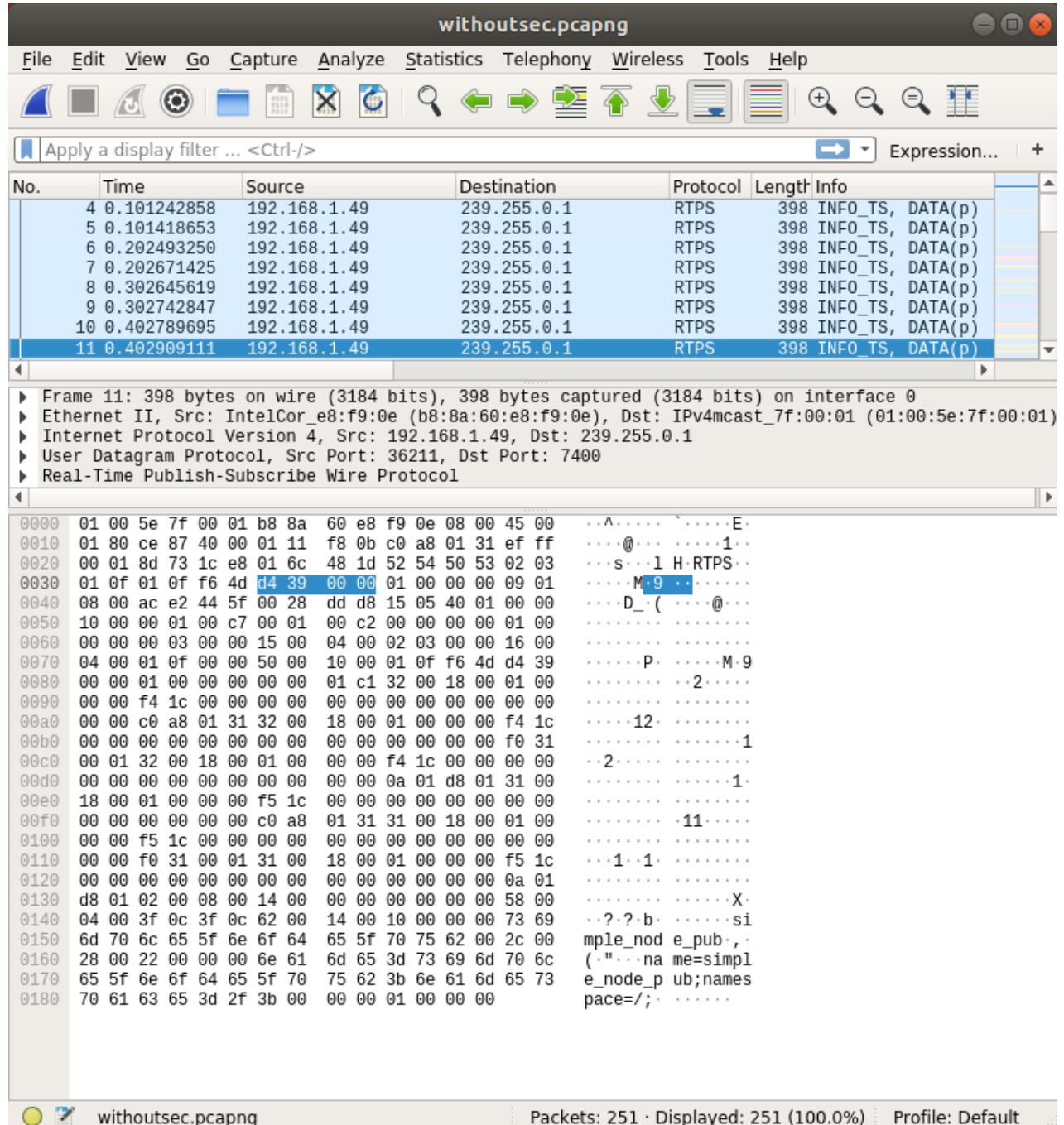


Figura 6.1: Captura de Wireshark durante la ejecución de los nodos simple_node_pub y sub.

Sin embargo, una vez realizada la correcta configuración de SROS2 y con el comienzo de la comunicación entre nodos se realizará una nueva captura con Wireshark para ver las diferencias que trae la activación de la seguridad de ROS2.

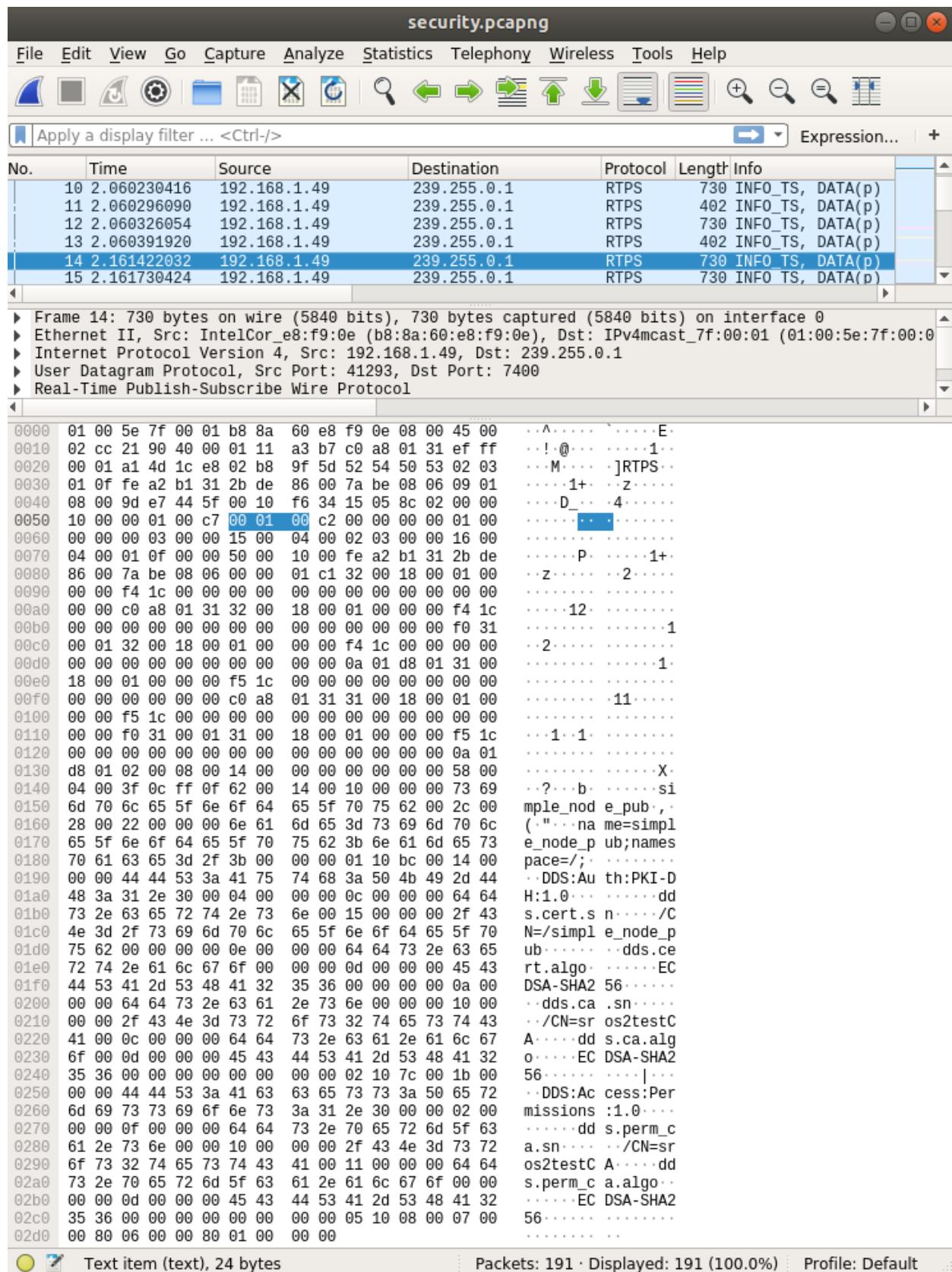


Figura 6.2: Captura de Wireshark durante la ejecución de los nodos simple_node_pub y sub con SROS2.

En la Figura 6.2 podemos observar que el mensaje se envía junto con todas las medidas de seguridad que hemos configurado en SROS2.

Pero el envío de los mensajes encriptados junto con los certificados y claves que hemos configurado con SROS2 para este caso específico, no será la única diferencia sustancial que encontraremos. Con la ayuda de Wireshark se ha realizado un filtrado para que aparezcan únicamente los paquetes intercambiados en las comunicaciones entre los nodos de ROS2. En las dos figuras siguientes se puede observar que hay un pico de intercambio de paquetes en la comunicación, esto es porque antes de establecerse una comunicación en ROS2 se realiza una fase de descubrimiento previa entre nodos y además en el caso de haber activado la seguridad se realizan otras comprobaciones como por ejemplo si el nodo que intenta suscribirse a un topic tiene permiso para ello, o si es quién dice ser y finalmente se establece y estabiliza dicha conexión.

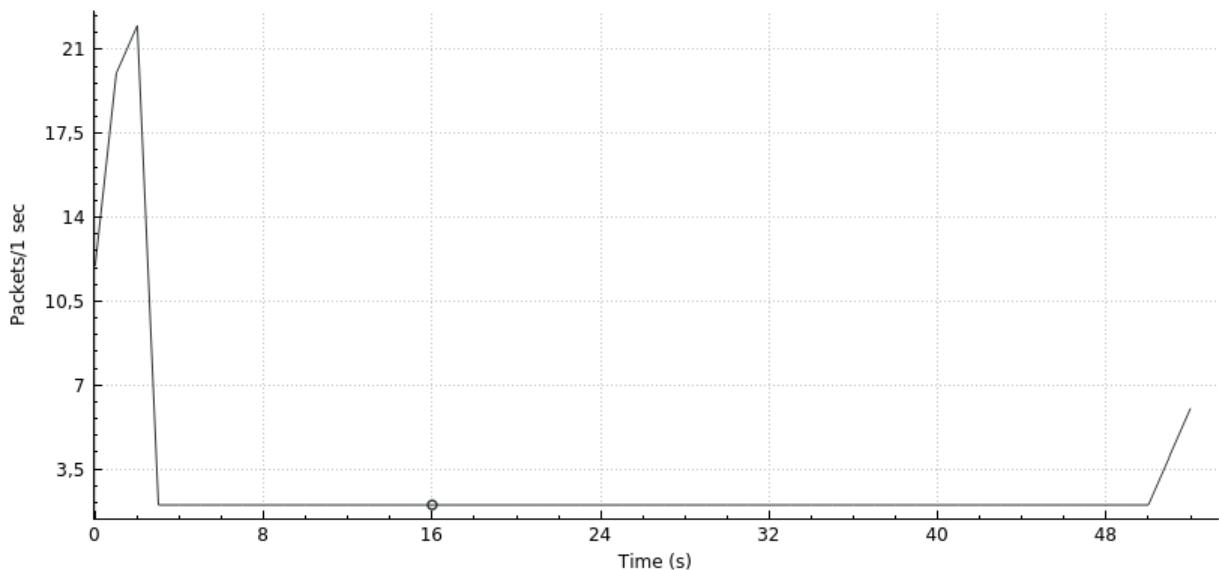


Figura 6.3: Medida de la latencia en la comunicación entre nodos sin SROS2.

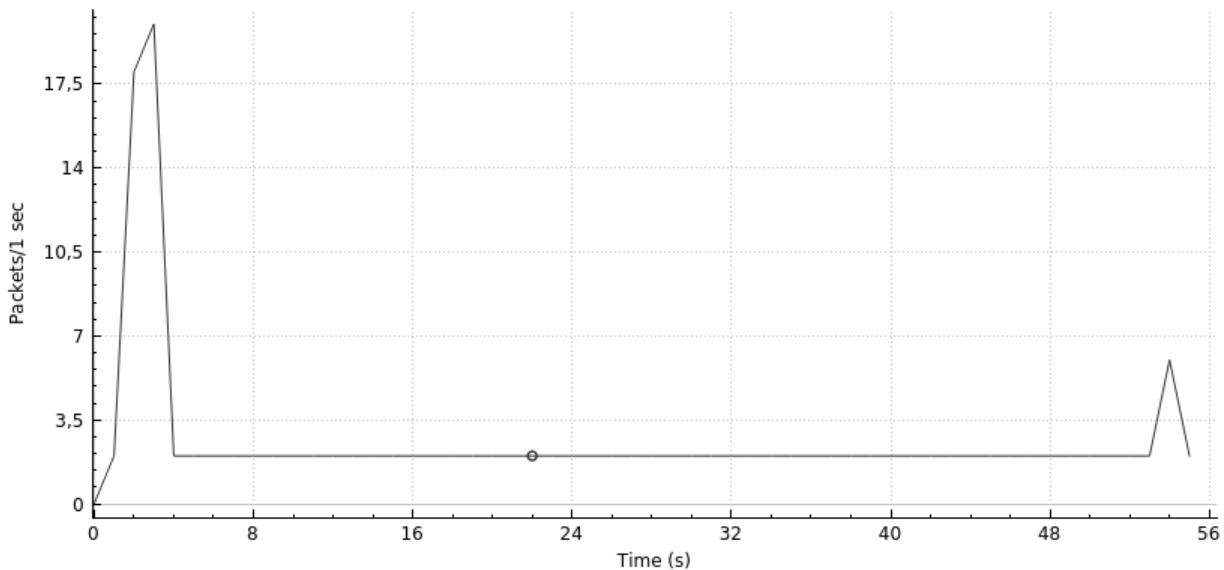


Figura 6.4: Medida de la latencia en la comunicación entre nodos con SROS2.

Como se puede comprobar en las gráficas representadas en las Figuras 6.3 y 6.4, existe una diferencia bastante elevada entre la cantidad de paquetes intercambiados con y sin aplicar seguridad en las comunicaciones. Mientras que en la primera gráfica se realiza una transmisión más rápida y un menor número de paquetes, en la segunda gráfica se envían un mayor número de paquetes ya que al estar encriptados probablemente se ha tenido que dividir el contenido de uno de esos paquetes en varios para realizar el envío.

Las conclusiones que podemos extraer de esta sección es que la utilización de la seguridad en ROS2 es beneficiosa, ya que aporta al sistema medidas adicionales de seguridad a parte de las que se pueden añadir al entorno físico del robot o a las medidas comunes en un sistema Ubuntu (en este caso concreto). Por estos motivos, podemos afirmar que la agregación de comprobaciones de seguridad de ROS2 a SCAP están justificadas y por lo tanto las aportaciones realizadas durante el desarrollo de este proyecto son interesantes, positivas y de gran valor tanto al mundo de la ciberseguridad como de la robótica.

Capítulo 7

Conclusiones

En este proyecto se ha llevado a cabo una investigación de cómo trabajar sobre entornos robóticos de una manera más segura, basándonos en el software más popular utilizado a día de hoy en el mundo de la robótica.

En este capítulo se realizará un análisis de la consecución de los objetivos propuestos al comienzo de este trabajo y finalmente se presentarán una serie de elementos e ideas para ayudar en una futura mejora del proyecto ya descrito y sus implicaciones.

7.1. Consecución de objetivos

Al comienzo de este proyecto se propusieron los siguientes objetivos:

- Investigar todas las posibles amenazas que pueden poner en riesgo la integridad de un robot y su sistema.
- Indagar sobre las herramientas software más utilizadas en el campo de la robótica. En este caso ROS2, conocer las diferentes opciones de securización que ofrece y tratar de probarlas y elegir la mejor opción.
- Realizar un modelo de amenazas centrado en ROS2.
- Desarrollar y modificar un conjunto de nuevas reglas con OpenSCAP que ayuden al usuario a realizar un análisis de seguridad automatizado y su posterior bastionado.

La consecución de estos objetivos se han cumplido en su totalidad, ya que el objetivo principal de este proyecto era demostrar que no hay un sistema, en este caso sistema robótico, que pueda estar completamente securizado, sin embargo sí que es posible automatizar el sistema de securización y chequeo de vulnerabilidades al máximo para dificultar el acceso al sistema en la mayor medida posible.

Anteriormente me he referido a la consecución de objetivos casi en su totalidad ya que hay puntos que no se han desarrollado todo lo posible dado la situación actual. Por ejemplo, no se ha podido tener acceso físico a un robot real y por eso a lo largo de este proyecto las pruebas y experimentos se han llevado a cabo en el simulador gazebo, que aunque sirve para el mismo fin y tiene los mismos resultados, no era uno de los puntos principales de este proyecto. Además, este contratiempo también ha afectado al punto de la creación de reglas SCAP, ya que no se han creado reglas para un robot específicamente.

En cualquier caso y bajo mi punto de vista, se ha resuelto el presente proyecto de manera eficaz y con las herramientas de las que se podía disponer dado la situación actual.

7.2. Competencias utilizadas y adquiridas

Durante el desarrollo de este máster se han adquirido una serie de competencias que han servido de apoyo a la hora de realizar este proyecto. Asignaturas como Criptografía Aplicada y Comunicaciones Seguras han sido fundamentales para comprender y saber diferenciar las comunicaciones encriptadas de las que no utilizan ninguna medida de seguridad en sus comunicaciones y realizan la transmisión de los datos más delicados para el sistema en claro. Por otra parte, la asignatura de Fundamentos del Software y los Componentes sirvieron de manera introductoria a entender algunas herramientas relacionadas con este trabajo para realizar algunas verificaciones automáticas como SCAP y sus componentes y lenguajes (XCCDF, OVAL... etc.). Y finalmente, la asignatura de Fundamentos de la Gestión de la Seguridad y la Información que sirvió para comprender la importancia de llevar a cabo una correcta auditoría de todos los sistemas de información y en general cualquier sistema en los que se almacenen o realicen trabajos esenciales o que puedan poner en compromiso una organización o la tarea final para la que se han diseñado.

Sin embargo, durante el desarrollo de este trabajo también se han adquirido otros conocimientos y habilidades que no estaban directamente relacionados con el tema principal del proyecto, pero que han sido muy útiles para comprender mejor el funcionamiento de los sistemas y las herramientas utilizadas.

mientos adicionales, como la utilización en profundidad de OpenSCAP así como muchas de las herramientas de las que provee al usuario como OpenSCAP Workbench. También la modificación de políticas de seguridad y creación de nuevas reglas más detalladas para crear unas políticas más específicas y que cumplan con su trabajo con un nivel de detalle mayor.

7.3. Trabajos futuros

A continuación se enumeran una serie de puntos con los que mejorar el proyecto de cada a la realización de un trabajo futuro partiendo del mismo.

- **Realizar tareas de mantenimiento.** Aunque la versión de eloquent para ROS2 actualmente es una versión estable, se siguen realizando mejoras en el código de manera eventual. Esto produce que sea necesario añadir o revisar algunas normas que lleven a cabo otros trabajos de comprobación adicionales.
- **Añadir nuevas reglas para las diferentes versiones de ROS2.** Al comienzo de este trabajo se tomó la decisión de llevarlo a cabo con la versión eloquent de ROS2 sobre Ubuntu 18.04 que era la versión más actual en el comienzo del proyecto, pero con el paso de los meses han aparecido nuevas versiones de ROS2, como ROS2 Foxy, que pasará a ser estable en breves y correrá sobre Ubuntu 20.04, por lo que sería de gran utilidad realizar este mismo desarrollo y pruebas para esta versión.
- **Añadir nuevas reglas más complejas.** Tanto para la versión actual de ROS2 para la que se ha realizado este trabajo como para futuras versiones, habrá que realizar un estudio más detallado de la generación de normas SCAP para crear unas normas más complejas y que abarquen en mayor medida los puntos débiles del sistema así como sus scripts de remediación.
- **Añadir nuevas reglas específicas para cada robot y sistema operativo.** En este trabajo se han desarrollado reglas *OpenSCAP* para un uso bastante generalista en robots. Con la creación de reglas más específicas para cada robot, las vulnerabilidades que puedan presentar estarán cubiertas en mayor medida.
- **Realizar este mismo proyecto desde un punto de vista más "agresivo".** El desarrollo

de este proyecto se ha llevado a cabo desde la perspectiva de una persona que hace un uso habitual de su robot en su trabajo y tiene como finalidad bastionar su sistema para evitar ataques y proteger su trabajo, sin embargo este mismo proyecto también se podría haber realizado desde el perfil de un atacante que intenta que intente penetrar las defensas del robot de manera constante.

- **Probar herramientas alternativas a *OpenSCAP*.** *OpenSCAP* es por excelencia una de las herramientas más utilizada y estandarizada para la automatización del chequeo de políticas de privacidad, pero existen muchas otras que se podrían investigar y que realizasen el mismo trabajo.

Bibliografía

- [1] E. Universal, “ROBOTS, LOS NUEVOS ENFERMEROS QUE TRATAN A PACIENTES CON COVID-19.” Available: <https://www.eluniversal.com.mx/ciencia-y-salud/tecnologia/coronavirus-robots-los-enfermeros-que-tratan-pacientes>.
- [2] A. Robotics, “AKERBELTZ.” Available: <https://arxiv.org/pdf/1912.07714.pdf>.
- [3] P. Robotics, “CALL FOR TIAGO LOAN FOR THE 1ST SCIROC CHALLENGE.” Available: <http://blog.pal-robotics.com/call-tiago-robot-loan-sciroc-challenge/>.
- [4] F. Martín, “ROS2 for ROS Developers, 2020.” Available: https://github.com/fmrico/ros_to_ros2_talk_examples.
- [5] “ROS 2.” Available: <https://github.com/ros2>.
- [6] “ROS.” Available: <https://www.ros.org>.
- [7] “OPENSCAP.” Available: <https://www.open-scap.org>.
- [8] “SCAP WORKBENCH.” Available: <https://www.open-scap.org/tools/scap-workbench/download>.
- [9] F. T. Expansión, Aliya Ram, “LOS HACKERS SACAN A RELUCIR LA FRAUDULENTIDAD DE LOS ROBOTS.” Available: <https://www.expansion.com/economia-digital/innovacion/2018/10/05/5bb642d0e5fdea13648b45f0.html>.
- [10] naiz, “HACKERS ÉTICOS GASTEIZTARRAS CREAN UN VIRUS QUE INFECTA EL ROBOT INDUSTRIAL MÁS VENDIDO.” Available: <https://www.naiz.eus/es/actualidad/noticia/20191219/hackers-eticos-gasteitztarras-crean-un-virus-que-infecta-el-robot-industrial-mas-vendido-para-concienciar-de-los-ataques>.

- [11] P. S. Cordero, “HARDENING, ASEGURAR O MORIR.” Available: <http://conexioninversa.blogspot.com/2015/04/hardening-formase-o-morir.html>.
- [12] “HARDENING DE DISTINTOS SISTEMAS Y APLICACIONES.” Available: <https://blog.seguinfo.com.ar/2013/02/hardening-de-distintos-sistemas-y.html>.
- [13] “CIS BENCHMARKS.” Available: <https://www.cisecurity.org/cis-benchmarks/>.
- [14] B. Gerkey, “WHY ROS 2.0?” Available: http://design.ros2.org/articles/why_ros2.html.
- [15] R. Hat, “¿QUÉ ES LINUX?.” Available: <https://www.redhat.com/es/topics/linux>.
- [16] “SISTEMA OPERATIVO ROBÓTICO.” Available: https://es.wikipedia.org/wiki/Sistema_Operativo_Robótico.
- [17] “ROS, INTRODUCTION.” Available: <http://wiki.ros.org/ROS/Introduction>.
- [18] “ROS INDEX. CONCEPTS.” Available: <https://index.ros.org/doc/ros2/Concepts/quick-overview-of-graph-concepts>.
- [19] “TOPICS.” Available: <http://wiki.ros.org/Topics>.
- [20] “ROS INDEX. ABOUT QUALITY OF SERVICE SETTINGS.” Available: <https://index.ros.org/doc/ros2/Concepts/About-Quality-of-Service-Settings/>.
- [21] E. Robotics, “MIDDLEWARE INTERFACE.” Available: http://docs.erlerobotics.com/robot_operating_system/ros2/basic_concepts/middleware_interface.
- [22] D. Thomas, “ROS 2 MIDDLEWARE INTERFACE.” Available: http://design.ros2.org/articles/ros_middleware_interface.html.
- [23] “ROS ON DDS.” Available: http://design.ros2.org/articles/ros_on_dds.html.
- [24] I. Open Source Robotics Foundation, “ROS ON DDS.” Available: http://design.ros2.org/articles/ros_on_dds.html.
- [25] “ROS INDEX. ABOUT DIFFERENT ROS 2 DDS/RTPS VENDORS.” Available: <https://index.ros.org/doc/ros2/Concepts/DDS-and-ROS-middleware-implementations/>.
- [26] “EPROSIMA FAST DDS.” Available: <https://github.com/eProsima/Fast-DDS>.

- [27] “WORKING WITH MULTIPLE ROS 2 MIDDLEWARE IMPLEMENTATIONS.” Available: <https://index.ros.org/doc/ros2/Tutorials/Working-with-multiple-RMW-implementations/>.
- [28] K. Fazzari, “ROS 2 DDS-SECURITY INTEGRATION.” Available: https://design.ros2.org/articles/ros2_dds_security.html.
- [29] “PKI: PUBLIC KEY INFRAESTRUCTURE.” Available: <https://infosegur.wordpress.com/unidad-4/pki-public-key-infrastructure/>.
- [30] “X.509.” Available: <https://es.wikipedia.org/wiki/X.509>.
- [31] “PRINCIPALES AUTORIDADES DE CERTIFICACIÓN.” Available: <https://firmaelectronica.gob.es/Home/Empresas/Autoridades-Certificacion.html>.
- [32] “SCAP(3/5): USANDO OPENSCAP.” Available: <https://www.mancomun.gal/es/artigo-tic/scap-35-usando-openscap/>.
- [33] “SCAP (1/4): SEGURIDAD EN LAS TIC Y EL MODELO SCAP.” Available: <https://www.mancomun.gal/es/artigo-tic/seguridad-en-las-tic-y-el-modelo-scap/>.
- [34] “OPENSCAP.” Available: <http://www.flealf.com/doku.php?id=linux:openscap>.
- [35] “TRY SROS2 IN LINUX.” Available: https://github.com/ros2/sros2/blob/eloquent/SROS2_Linux.md.
- [36] “COMPLIANCEASCODE/CONTENT.” Available: <https://github.com/ComplianceAsCode/content>.
- [37] “COMPLIANCEASCODE/CONTENT.” Available: <https://github.com/lbajo/content>.
- [38] “ROS2 NAVIGATION.” Available: <https://github.com/ros-planning/navigation2>.