Ciberseguridad en robots autónomos: Análisis y evaluación multiplataforma del bastionado ROS

Francisco Javier Rodríguez Lera, Vicente Matellán, Jesús Balsa, Fernando Casado[1]

Resumen— **Los robots autónomos son cada vez más comunes, lo que hace que la ciberseguridad de los sistemas empotrados que los controlan se esté convirtiendo en una preocupación. Muchos de los sistemas robóticos que se desarrollan actualmente utilizan el *framework* distribuido ROS. Este sistema se diseñó para uso fundamentalmente investigador y prácticamente no incluye ningún mecanismo de ciberseguridad. Proponemos utilizar el cifrado 3DES para solventar los problemas de confidencialidad asociados al uso del paradigma de publicación-subscripción en claro que usa ROS. Para valorar la influencia del cifrado en el rendimiento general de las plataformas, presentamos una comparativa del desempeño de dos plataformas robóticas con diferentes CPUs. Los campos analizados fueron tres: tiempo de cifrado/descifrado, *throughput* y tiempo consumido de CPU. En los experimentos utilizamos dos sensores ampliamente extendidos en robótica: una webcam y un sensor láser. Los resultados muestran que sobre robots con poca capacidad computacional, el cifrado afecta en el rendimiento de los sensores y en el tráfico de red generado.**

Palabras clave— **Robótica, ciberseguridad, bastionado, ROS**

I. Introducción

La línea principal de investigación de nuestro grupo es el desarrollo de software para el control de sistemas autónomos. En concreto, durante los últimos años hemos estado desarrollando una plataforma de asistencia [4] para personas mayores y dependientes.

Como muchos otros grupos de investigación, nuestro grupo [1] utiliza ROS (Robotic Operating System) como *framework* de desarrollo. Desde su popularización por Willow Garage en 2008 este entorno distribuido se ha convertido en el estándar *de facto* para el desarrollo de software para robots. Muchos sistemas autónomos utilizan ROS en diferentes entornos, plataformas robóticas para la investigación, manipuladores como Baxter de Rethink Robotics [1] o el recién anunciado soporte de los corta-cesped robóticos del fabricante Husavarna [2].

Nuestro interés en la seguridad de estos sistemas ha venido impuesta al comenzar los experimentos en entornos reales. Al desplegar nuestros robots en domicilios particulares o centros asistenciales, los responsables jurídicos nos solicitaron información sobre la seguridad de nuestros robots puesto que accederían a información protegida: imágenes, datos médicos, etc. de personas vulnerables.

De esta forma surgió esta investigación que forma parte de una línea de trabajo más amplio de securización de sístemas autónomos, y en la que prestamos atención a sistemas robóticos que utilizan ROS como *framework* de control.

A. ROS

La arquitectura de ROS se basa en un grafo de "nodos" encargados de los distintos procesos relevantes para el sistema (procesar sensores, calcular la localización, navegar, etc.) que funcionan de forma distribuida basada en subscripción asíncrona. Para que esto tenga lugar es necesario levantar lo que se denomina `roscore`. Su labor es la de levantar un grupo de nodos y programas base del sistema: el ROS Master, el servicio de parámetros de ROS (ROS Parameter Server) y el nodo de log (rosout).

Para el descubrimiento de los servicios ROS mantiene el nodo ROS Master, encargado de coordinar todo el sistema y que actua como servidor de nombres. Cuando se crea un nuevo nodo este registra su servicios en el "ROS Master" y le enumera los "topics" que ofrecerán información de forma autónoma y asíncrona. Cuando otro nodo quiere usar el servicio disponible en un topic, solicita la información al "ROS Master" que le devuelte la IP y el puerto relativo al topic. En ese momento se establece la comunicación entre nodos, se dice entonces que este segundo nodo está "subscrito" al primero.

El problema de este diseño es que todas estas comunicaciones se producen sin aplicar ningún mecanismo de seguridad, excepto en aquellos casos en los que se levanta la red ROS en un entorno VPN. De esta forma, como refleja la figura 1, se podrían generar los siguientes problemas:

Interrupción del servicio : un software malicioso suprimiría los paquetes, haciendo que los subscriptores dejasen de recibir información.

Modificación : el *malware* podría modificar los mensajes, produciendo información errónea.

Suplantación : un nodo malicioso podría suplantar a uno legítimo, fabricando mensajes completamente falsos.

Estos tres problemas se asocian a los tres principios básicos que debe asegurar la seguridad de los sistemas informáticos: Confidencialidad,

[1]Instituto de Ciencias Aplicadas a la Ciberseguridad. Grupo de Robótica. Universidad de León. Web: `http://robotica.unileon.es`. Autor de contacto: `vicente.matellan@unileon.es`.

[1]http://robotica.unileon.es

[2]http://www.ros.org/news/2016/05/husqvarna-research-platform-ctu.html

Integridad y Disponibilidad. Atendiendo al US-CERT [7], este es el orden de criticidad genérico ante un ataque en entornos IT. Ahora bien, el mismo documento presenta un orden diferente de criticidad cuando se consideran sobre un sistema de control industrial: Disponibilidad, Integridad y Confidencialidad.

Planteado el problema, el resto del artículo está organizado como sigue: la siguiente sección revisa el estado del arte de la seguridad en robótica y en ROS en particular. La siguiente sección analiza diferentes alternativas para bastionado de ROS y su rendimiento. Posteriormente se analiza el funcionamiento de dos robots que utilizan ROS y sobre los que se aplica un sistema de cifrado para dos campos de los menajes utilizados en dos sensores reales. A continuación se describe una propuesta para el análisis en tiempo de ejecución de las condiciones de seguridad. Finalmente se presentan unas conclusiones y las líneas de trabajo futuro de nuestro grupo en este campo.

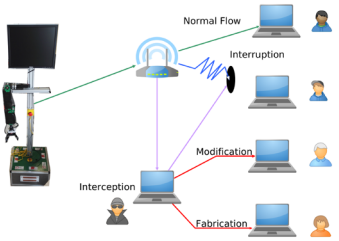

Fig. 1
Tipos de ciber-ataques en un robot de teleasistencia.

II. Estado del Arte

Poniendo el foco en la seguridad en sistemas ciberfísicos desde un punto de vista robótica doméstica [12],[13] frente al industrial US-CERT [7] es necesario tener en cuenta dos aproximaciones. En primer lugar, la robótica asistencial doméstica es totalmente diferente a las soluciones robóticas industriales: entorno de despliegue, usuarios, rendimiento o resiliencia por enumerar alguna de ellas. Por esta razón, las cuestiones de seguridad que se venían aplicando a nivel industrial no se pueden extrapolar 1 a 1 entre entornos.

En el trabajo de Denning [10] se evalúan los puntos de fuga en plataformas de juguete como Rovio, Spykee o RoboSapien v2. Entre sus análisis, evaluaban si era posible conectarse directamente al flujo de datos de la cámara o que era posible desplazar diferentes objetos del entorno una vez situado en el mismo segmento de red.

En segundo lugar, el software de control del robot y los datos que gestiona. En el entorno doméstico, si pensamos en un sistema aislado típico, un robot aspirador, el usuario no afrontará problemas de confidencialidad pero sí de disponibilidad, requiere que funcione cuando lo necesita. Ahora bien, si se trata de un sistema asistencial que trabaja información vital del usuario (niveles de azúcar, rutina en el hogar) el orden de importancia ante un ataque cambia y es preciso alterar el orden y las prioridades. La tabla I presenta nuestra propuesta para robots sociales y asistenciales junto con la propuesta para entornos críticos e industriales expuesta en [7].

Como se ha comentado, ROS es un software de control ampliamente extendido. El problema de la seguridad en ROS es un hecho conocido y se han descrito diferentes tipos de problemas. Investigadores como el Profesor Dr. Hartmut Pohl revisan ROS [11] atendiendo a los cuatro elementos básicos de la seguridad: disponibilidad, integridad, confidencialidad y autenticidad. Además, el Dr. Harmut plantea también ataques simples sobre el nodo Master de ROS aprovechando las vulnerabilidades asociadas a XML-RPC. Dicho autor plantea el problema que nosotros abordamos en este trabajo, el cifrado de las comunicaciones más allá del uso de las VPN's o el cifrado por defecto de las WLANS.

Un ejemplo práctico de la seguridad en ROS lo plantea McClean en su trabajo [5], donde se describe el despliegue de un *honeypot* para probar la seguridad de robot. En dicho trabajo se utilizó un robot que disponía de una brújula y varios sensores de percepción del entorno y que se tele-operaba desde un nodo ROS escrito en Javascript y alojado en un servidor remoto. Los autores se centran en los problemas de utilizar comunicaciones en texto plano, el uso de puertos TCP sin proteger, o el almacenamiento de información sin cifrar. Este tipo de problemas son el reflejo en robótica de los problemas clásicos de seguridad en redes de ordenadores.

En un trabajo previo [3] presentamos un primer análisis detallado del rendimiento de las comunicaciones de ROS utilizando el sistema 3DES. En esa primera aproximación las pruebas se limitaron a mensajes de tipo `sensor_msgs/Image.msg` y de tipo `String`. El primer tipo era el utilizado por un sensor real (cámara USB) sobre la que cifrabamos el campo array de tipo `uint8` que contiene la información de la imagen. El segundo tipo se utilizó para evaluar el comportamiento global del sistema embebido del robot en condiciones de estrés. En este caso se evaluó el comportamiento del sistema de cifrado ante bloques de 256 KB, 512 KB, y 1024 KB.

III. Bastionado de ROS

Atendiendo al problema de confidencialidad inherente a ROS al publicar en texto plano y teniendo en cuenta que el nivel criticidad es alta en robótica doméstica (robótica asistencial y robótica

TABLA I
PERFILES DE SEGURIDAD ASOCIADOS A LA ROBÓTICA.

	Criticidad		
Perfil	Confidencialidad	Integridad	Disponibilidad
IT	Alta	Alta	Baja
Control Industrial	Baja	Media	Muy alta
Robótica Asistencial	Muy alta	Muy alta	Muy alta
Robótica Sociales	Muy alta	Media	Baja

social). Proponemos una solución de bastionado pasivo basado en el cifrado de las comunicaciones entre los distintos nodos del sistema de control, tanto *on-board* del robot, como en especial las comunicaciones con nodos externos. De esta forma, se plantea una solución de cifrado punto a punto. La figura 2 presenta a alto nivel el sistema propuesto.

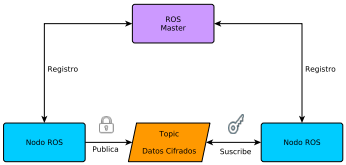

Fig. 2
ARQUITECTURA DEL SISTEMA DE CIFRADO EN COMUNICACIONES.

De esta forma, antes de publicar la información adquirida por un sensor, realizamos un cifrado de la misma. Desde ese momento cualquier nodo que se introduzca en la red no verá la información publicada si no dispone de la llave de descifrado.

En esta primera propuesta de bastionado ROS se cifran parcialmente aquellos mensajes publicados en la red con información relevante de los usuarios: imágenes y láseres. El motivo de cifrar parcialmente es que el rendimiento empeora mucho al cifrar grandes bloques de datos, como se analizó en trabajos previos [3] al cifrar grandes bloques de datos. Como se planteó en dicho trabajo, se cifrará el bloque de datos que lleva la información del sensor, no el mensaje completo.

IV. PLANTEAMIENTO DE LA EXPERIMENTACIÓN

Uno de los problemas básicos del uso de cifrado en sistemas empotrados es la pérdida de rendimiento del sistema. En la siguiente sección realizamos un análisis del rendimiento, tanto desde el punto de vista de las comunicaciones, como desde el punto de vista del funcionamiento del sistema.

A. Escenario HW de las pruebas

El entorno que hemos diseñado para realizar las pruebas consiste en dos robots, cada uno con un sistema embarcado de control diferente y un equipo externo. El primer robot monta un Intel(R) Atom(TM) CPU D525 @ 1.80GHz con 2GB de RAM. El segundo robot utiliza un equipo Intel(R) Core(TM) i7-4790 CPU @ 3.60GHz con 16GB de RAM. En adelante, denominaremos al primer robot como **Robot1 o R1** y al segundo como **Robot2 o R2**. El equipo externo utiliza el mismo procesador Intel(R) Core(TM) i7-4790, este equipo se denominará en adelante **EE**.

Para simplificar las pruebas utilizamos dos sensores, un láser y una cámara, de los instalados en sendos robots. El modelo de láser utilizado fue en ambos casos un URG-04LX-UG01. La alimentación de dicho dispositivo se hace a través del mismo conector USB que le comunica con el sistema de control. La cámara utilizada fue del modelo QuickCam Pro 9000 Logitech, Inc. que se ha configurado para que ofrezca 15 frames por segundo (fps) con una resolución de 640x480 pixels.

Las conexiones entre equipos se han realizado por Ethernet a un switch Alcatel-Lucent OmniSwitch 6860E. En todos los casos se han utilizado cables Cat 6.

B. Escenario SW de las pruebas

Las conexiones entre el robot y equipo externo se realizan a través de ROS. Todos los equipos utilizaron GNU/Linux, concretamente Ubuntu 14.04.4 LTS aunque cada robot es completamente autónomo desde el punto de vista software. Desde el punto de vista de ROS eso quiere decir que cada robot arranca un `roscore`.

En cada robot se ha añadido un nodo ROS de cifrado que utiliza el algoritmo 3DES. El algoritmo 3DES (*Triple Data Encryption Standard* - TDES) es de tipo clave privada y cifra por bloque. Es la evolución del algoritmo DES (reconocido como estándar por el Instituto Nacional de Estándares y Tecnología NIST en 1974) pero con triple cifrado mediante claves de 64bits [8].

Hemos decidido utilizar este algoritmo ante alternativas [14], como AES o *blowfish* por tres razones. La primera por compatibilidad, puesto que ciertos dispositivos antiguos (lectores de tarjetas) con los que estamos trabajando utilizan DES. La segunda la disponibilidad de sistemas embebidos que realizan el cifrado en hardware, en lugar de a nivel software, para mejorar el rendimiento. La tercera se basa en estudiar .el peor caso", estudiamos el algoritmo más costoso en términos de coste computacional [9] para analizar la carga que se introduce sobre el sistema

respecto al comportamiento nominal o esperado.

El equipo externo realizará siempre el descifrado. Se han desarrollado dos nodos de descifrado, uno para cada sensor, ya que como se ha descrito los sensores no comparten el mismo tipo de mensaje. Los dos nodos ROS conocen la clave y son por tanto capaces de descifrar la información proporcionada por el láser o la webcam. El intercambio seguro de la clave queda fuera del objetivo de este estudio. Por el contrario, si se utiliza alguna de las herramientas de ROS como `rostopic echo` se presentará la información cifrada ya que la herramienta no dispone de la llave.

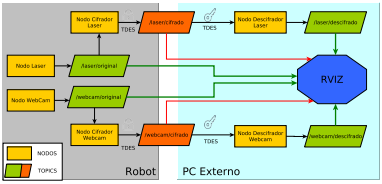

Fig. 3
Despliegue de nodos ROS planteado en los experimentos.

La figura 3 presenta el planteamiento general de los nodos utilizados en los experimentos. Se han utilizado dos nodos asociados a los sensores: el *Nodo Láser* que en el sistema de ROS es el `Hokuyo node` y el *Nodo Webcam* que corresponde en ROS al paquete `usb_cam`. Para facilitar el análisis se mantienen dos `topics` donde se puede consultar la información generada por el sensor sin cifrar (`laser/original` y `/webcam/original`). Esto permite tener un patrón para comparar la información real de los sensores.

Se han incluido dos nodos de cifrado, uno para cada sensor, que están suscritos a los topics que publican la información sin cifrar (en texto plano). Cada uno de ellos utiliza un tipo de mensaje diferente.

Para el láser se utiliza el mensaje de tipo compuesto `sensor_msgs/LaserScan.msg`:

```
Header header

float32 angle_min
float32 angle_max
float32 angle_increment

float32 time_increment
float32 scan_time

float32 range_min
float32 range_max

float32[] ranges  #array de datos del haz

float32[] intensities
```

Este tipo de mensaje está compuesto por diez campos que definen diferentes características del sensor y del haz láser.

A la hora de cifrar la información, el planteamiento propuesto en esta investigación cambiar únicamente aquel campo que contiene la información relativa al entorno y con el que se podría realizar un seguimiento del usuario. En este caso el campo afectado es el denominado `ranges` y sobre el se realizará el cifrado conveniente para evaluar el comportamiento general del sistema. El número de elementos medio observado en dicho array durante nuestros experimentos fue de 4853.

El segundo tipo de mensaje utilizado fue *sensor_msgs/Image.msg* y también es de tipo compuesto. La matriz de información de la imagen se envía en la variable *data* que es de tipo *uint8* e igual que en el caso anterior, será el único campo que se cifre durante los experimentos. El tamaño medio en este caso del array observado en nuestros experimentos fue 39.583 elementos.

```
std_msgs/Header header #custom msg

uint32 height
uint32 width
string encoding
uint8 is_bigendian
uint32 step

uint8[] data #matriz de datos de la imagen
```

Tas el proceso de cifrado cada nodo publica en un topic que sigue manteniendo el tipo de mensaje ROS `/laser/cifrado` y `/webcam/cifrado`.

Finalmente se incluyen dos nodos de descifrado que realizan los pasos inversos a los trazados en los nodos cifradores. Dichos nodos publican la información descifrada en los *topics*: `/laser/descifrado` y `/webcam/descifrado`

Por último, la figura 3 muestra un nodo denominada `rviz`. Este nodo representa la herramienta de visualización 3D que incorpora el ROS. En la figura se pueden ver dos tipos de flechas que se conectan a dicho nodo: las flechas rojas corresponden a suscripciones a *topics* cifrados y las flechas verdes a *topics* no cifrados. De esta forma la información que viene cifrada no será visualizable en `rviz` debido a que la aplicación por defecto no tiene la clave de descifrado. Por el contrario, la información de los *topics* en verde se puede visualizar perfectamente en la aplicación `rviz`.

C. Métricas de Análisis

Los elementos analizados durante la experimentación fueron tres:

A) Tiempo de cifrado y descifrado
B) Tiempo de proceso de CPU
C) *Throughput* o volumen de información generado por cada nodo.

En el primer caso medimos el tiempo necesario para realizar el proceso de cifrado y descifrado. Este parámetro es clave en sistemas de mensajes por el cuello de botella que se puede generar. En particular, medimos la llamada a la función que realiza el

proceso de cifrado.

En el segundo caso medimos el consumo de CPU, en concreto medimos el tiempo total que cada nodo está usando la CPU. De este modo podríamos valorar la repercusión que hay sobre el resto de procesos del sistema.

En el tercer caso, dado que es necesario evaluar el rendimiento de red, medimos el *throughput* del sistema, es decir, el número total de bytes/segundo que publica cada nodo.

V. Resultados de la Experimentación

A. Tiempos de ejecución del cifrado y descifrado

A.1 Robot1 - Cifrado

El *Nodo Cifrador Láser* realizó 20.584 llamadas a la función de cifrado, obteniendo un tiempo mínimo 7,812 ms y 12,461 ms de tiempo máximo. La media fue 8,190 milisegundos, con una desviación estándar de 0,211 ms.

El *Nodo Cifrador Webcam* en el robot que utilizaba el Atom(TM) obtuvo 66,394 ms de tiempo mínimo para el cifrado y un tiempo máximo de 100,157 milisegundos. La media de este proceso fue de 77,448 ms sobre 18.988 mensajes, con una desviación estándar de 0,007 ms.

A.2 Robot2 - Cifrado

En este caso el *Nodo Cifrador Láser* desplegado sobre un Core(TM) i7 generó 11.875 llamadas a la función de cifrado obteniendo uno tiempo medio de 1,871 ms y una desviación típica de 0,001 ms. El valor mínimo fue de 1.064 ms y el máximo fue de 14,874 ms.

El *Nodo Cifrador Webcam* sobre el robot 2, presentó un valor mínimo de 1,309 ms y una duración máxima en el proceso de cifrado de 26,909 ms. La media en este caso fue de 10,948 ms con unas desviación estándar de 0,004 ms.

A.3 Equipo Externo - Descifrado

El comportamiento de los descifrados es similar, sea cual sea el origen de los datos. La tabla II resume los resultados de los ocho casos contemplados.

TABLA II
Rendimiento de la llamada a la función de descifrado. Los casos definidos son: 1) Descifrado Láser en EE con origen R2; 2)Descifrado imágenes en EE on origen R2; 3) Descifrado Láser en EE con origen R1; 4) Descifrado imágenes en EE on origen R1. (Unidades: milisegundos)

	1.Láser(R2-EE)	2.Webcam(R2-EE)	3.Láser(R1-EE)	4.Webcam(R1-EE)
Media	1,714	14,390	1,299	12,940
Desv. est.	0,689	2,642	0,259	3,5670
Min	1,117	11,739	1,198	10,856
Max	6,803	50,829	6,174	49,872

El resumen de todos los datos analizados se presenta en forma de box-plot en la figura 4

B. Tiempos de proceso de CPU

Para evaluar el tiempo real consumido de CPU por cada nodo se utilizó el comando `time` disponible en los sistemas Unix. Dicho comando ofrece el tiempo de ejecución de la aplicación (REAL), el tiempo de usuario (USER) y el tiempo del sistema (SYS). El tiempo total de CPU utilizado es la suma de los tiempos de USER y de SYS.

La tabla III muestra los resultados de la evaluación del consumo de CPU de los nodos utilizados (sin contar la aplicación `rviz`) para el experimento con el láser. Del mismo modo, la tabla IV muestra los tiempos asociados al experimento con la cámara.

TABLA III
Consumo de CPU de los nodos. Caso revisado: láser.(Unidad:minutos)

Robot 1	Nodo Láser (R1)	Nodo Cifrador Láser (R1)	Nodo Descifrador Láser (EE)
real	35.20095	31.7332666667	31.451
user	0.2138666667	19.4243333333	2.0888666667
sys	0.1362666667	1.0046	0.0918666667
Robot 2	Nodo Láser (R2)	Nodo Cifrador Láser (R2)	Nodo Descifrador Láser (EE)
real	19.9671333333	19.8355833333	19.4242833333
user	0.0472	1.4501333333	1.242
sys	0.0424	0.0548666667	0.0479333333

Las duraciones de los experimentos no son fijas y superan en casi todos los casos los 30 minutos. La razón es que en la competición RoboCup@home hay una prueba denominada `Enhanced Endurance General Purpose Service Robot` en la que se busca tener el robot realizando diferentes tareas entre 30 y 45 minutos. La única prueba que no se realizó en este intervalo fue la del robot 2 con el sensor láser en la que estuvo casi 20 minutos, que corresponde a una prueba previa a utilizar el patrón RoboCup.

TABLA IV
Consumo de CPU de los nodos. Caso revisado: webcam.(Unidad:minutos)

Robot 1	Webcam (R1)	Nodo Cifrador Webcam(R1)	Nodo Descifrador Webcam(EE)
real	38,3697666667	36,4949666667	33,8407666667
user	14,5042666667	33,2512666667	4,6739333333
sys	0,7362666667	1,0421333333	0,1518666667
Robot 2	Webcam (R2)	Nodo Cifrador Webcam(R2)	Nodo Descifrador Webcam(EE)
real	48,6318833333	47,7332833333	45,6914166667
user	2,6126666667	15,6547333333	11,8293333333
sys	0,1509333333	0,3896666667	0,3404

C. Throughput

En este caso, el *Throughput* mide el flujo de datos asociado a cada nodo que publica en la red. Se han evaluado los dos flujos entre los tres nodos que corren en cada cada experimento: *Nodo Láser*, *Nodo Láser Cifrador* y *Nodo Láser Descifrador* y sus homólogos para la cámara.

En estas circunstancias los valores de *Throughput* para el láser del robot 1 son:

1. *Nodo Láser* →*Nodo Cifrador Láser*: 11 mensajes por segundo de media con un trafico de 23199 bytes.
2. *Nodo Cifrador Láser* →*Nodo Descifrador Láser*: 28.3874 bytes por segundo con un total de 10 mensajes por segundo de media

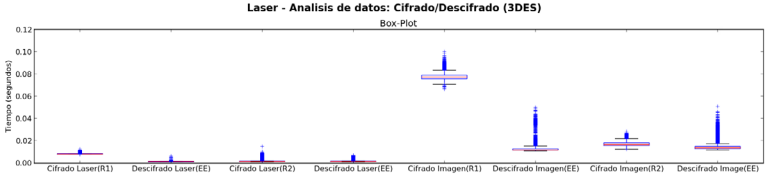

Fig. 4
BOX-PLOT GENERADO A PARTIR DE LOS DATOS OBTENIDOS DE NUESTROS PROCESOS EXPERIMENTALES.

Finalmente el flujo completo para el sensor del láser para el robot 2 mostró los siguientes datos:

1. *Nodo Láser* →*Nodo Cifrador Láser*: 11 mensajes por segundo de media con un tráfico de 21090 bytes.
2. *Nodo Cifrador Láser* →*Nodo Descifrador Láser*: 298722 bytes por segundo en un total de 10 mensajes de media por segundo.

En el caso asociado a la cámara, el flujo varía de acuerdo al formato utilizado. El *Nodo Webcam* (paquete usb_cam de ROS) permite dos métodos de trabajo *raw* y *Compressed*. El utilizado en el experimento fue el *Compressed*.

Si comparamos ambos en condiciones ideales se observará que la diferencia entre ambos es amplia. Para el caso *raw* teneos un flujo de 14.285.394 bytes por segundo de media. En el segundo caso (*Compressed*) los valores arrojan un flujo de 22.438,67 bytes de media por mensaje enviado en el robot 1 y un flujo de 19.498,62 bytes de media por mensaje enviado en el robot 2.

El *Throughput* para las imágenes en el robot 1 fue:

1. *Nodo Webcam* →*Nodo Cifrador Webcam*: 8 mensajes por segundo de media con un tráfico de 425.348 bytes.
2. *Nodo Cifrador Webcam* →*Nodo Descifrador Webcam*: 431.008 bytes por segundo en un total de 8 mensajes que son enviados por segundo de media.

Los datos obtenidos en el caso del robot 2 son:

1. *Nodo Webcam* →*Nodo Cifrador Webcam*: 15 mensajes por segundo de media con un trafico de 376.704 bytes.
2. *Nodo Cifrador Webcam* →*Nodo Descifrador Webcam*: 638.080 bytes por segundo en un total de 15 mensajes por segundo de media.

VI. DISCUSIÓN SOBRE EL RENDIMIENTO

De los resultados obtenidos anteriormente se pueden extraer las conclusiones asociadas a dos conceptos: el rendimiento del sistema y la funcionalidad del sistema.

A. Rendimiento de los sistemas

La figura 4 resume todos los casos analizados distinguiendo entre el robot 1 y el robot 2. Como era de esperar, el tiempo utilizado para el cifrado en el sistema R1 es más alto. En el caso del láser, es ligeramente mayor en media (7ms) y en el caso de la cámara es mucho mayor, siendo la diferencia de casi 60 ms.

Además del incremento en los tiempos de cifrado, es significativo el tiempo de CPU que necesita para realizar la tarea en el robot 1. Las figuras 5 y 6 muestran los tiempos (en minutos) de los experimentos con el láser y la cámara respectivamente.

Los experimentos con el láser muestran que el comportamiento del sistema es parecido en ambos robots. El número de mensajes servidos por el *Nodo Láser* es 10 msgs/segundo en todos los casos y no se produce ningún rechazo de mensajes en el nodo descifrador. A la vista de los datos presentados en la figura 5, resalta el aumento de uso de CPU que genera el *Nodo Cifrador Láser* sobre la plataforma Atom(TM), donde el cifrado requiere un 65 % del tiempo (20.43 minutos) disponible de CPU.

A partir de los datos de *throughput*, se puede observar que la cantidad de datos transmitidos por el *Nodo Cifrador Láser* es mayor que el generado por *Nodo Láser*. El volumen de datos es casi el doble analizando las medias. Así el volcado del *Nodo Cifrador Laser*, es aproximadamente 14 veces mayor: 21.090 bytes frente a 298.722 bytes. Esto es debido a que es necesario acondicionar la información cifrada realizando un cambio de base para no modificar el tipo de mensaje.

Atendiendo al experimento con la cámara, observamos que el *Nodo Webcam* muestra que se requiere de un procesamiento extra en el robot 1 para conseguir los 15 fps que ofrece por defecto. Cuando el *Nodo Cifrador Webcam* corre en R1, el tiempo de CPU utilizado es superior al 90 %. En el robot 2 ese consumo de CPU es inferior al 45 %. En estas circunstancias, el robot 1 no cifra el 100 % de los mensajes enviados (15 fps) y rechaza de media 7 mensajes, obteniendo un rendimiento de 8 fps. El elevado consumo de CPU influye en el comportamiento del resto de aplicaciones

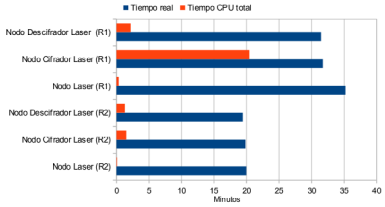

Fig. 5
EXP. LÁSER: CONSUMO DE CPU CORRIENDO UN SISTEMA DE CIFRADO EN ROBOT CON CORE(TM) I7 Y EN UN ROBOT CON UN ATOM(TM).

corriendo en el robot, pero no impide el correcto funcionamiento del mismo.

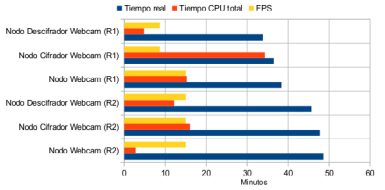

Fig. 6
EXP. WEBCAM: CONSUMO DE CPU CORRIENDO UN SISTEMA DE CIFRADO EN ROBOT CON CORE(TM) I7 Y EN UN ROBOT CON UN ATOM(TM).

B. Discusión sobre la Funcionalidad

En el caso de las imágenes el mayor problema funcional es el generado por la disminución en el `frame rate`, que baja de 15 fps a 8 fps.

En el caso del láser el problema no está ligado con el rendimiento, viene dado por el error sistemático inducido al cifrar la información en el mensaje.

La figura 7 presenta dos capturas del rviz con información del láser conectado al robot con Core(TM) i7 (superior) y al robot con un Atom(TM) (figura inferior). El haz sin cifrar se presenta en color verde en la figura, y el haz láser después de haber sido cifrado y descifrado se presenta en rojo.

En el primer caso se observa claramente que no coinciden al mismo tiempo ambos el haz de láser a partir de una distancia superior a 2,25 metros. Esto es debido a que la precisión es menor a esa distancia y el sistema de cifrado utilizado (PyCrypto) realiza un truncado de dos cifras, lo que hace que la coincidencia entre ambos no sea perfecta. En el caso del Atom (TM) se observa que el error se produce incluso a distancias menores de dos metros y que es posible encontrar diferencias significativas (entre 2 y 10 cm). Si bien el error inferior a 5cm puede ser aceptable en entornos de interior a la hora de navegar o seguir personas, se hace necesario un análisis pormenorizado de este comportamiento ante situaciones de despliegue del robot en entornos reales.

VII. CONCLUSIONES Y TRABAJO FUTURO

ROS parece consolidarse como la plataforma estándar para el desarrollo de software para robots autónomos, pero es un entorno que no ha sido diseñado para la seguridad informática.

Resaltar que en nuestra propuesta no hemos cambiado ningún campo de los que conforman el mensaje ROS y que el resto de valores que contiene el mensaje se transmiten sin cifrar.

Tras el trabajo realizado en esta investigación hemos observado que cifrar en las comunicaciones resolvería problemas de confidencialidad. Esto supone contener ataques del tipo `man-in-the-middle` que puedan afectar al sistema. La desventaja de esta solución con respecto a las propuestas de VPN, es el empeoramiento del rendimiento de ciertos sensores e incremento de consumo de CPU. Esta situación limitará el rendimiento del resto de aplicaciones que son desplegadas en el robot.

Un segundo resultado observado en esta investigación, no por esperable menos significativo, es que la cantidad de datos vertida a la red es mucho mayor con cifrado que cuando se usa ROS sin cifrar.

Como trabajo futuro estamos trabajando en alternativas al cifrado para ofrecer un sistema de seguridad en ROS para permitir establecer políticas de autenticación a los nodos y a los equipos de la red.

AGRADECIMIENTOS

El presente trabajo ha sido financiado parcialmente por el Ministerio de Economía y Competitividad del Reino de España a través del proyecto DPI2013-40534-R y por el Instituto Nacional de Ciberseguridad (INCIBE) por la Adenda21 del convenio entre la Universidad de León e INCIBE.

REFERENCIAS

[1] Conor Fitzgerald, Developing Baxter IEEE International Conference Technologies for Practical Robot Applications (TePRA), 2013

[2] Huang, J., Erdogan, C. Zhang, Y. Moore, B. Luo, Q. Sundaresan, A. Rosu, G. ROSRV: Runtime verification for robots Lecture Notes in Computer Science, Vol. 8734, pp 247-254 DOI: 10.1007/978-3-319-11164-3_20

[3] Francisco Javier Rodríguez Lera, Jesús Balsa, Fernando Casado, Camino Fernández, Francisco Martín Rico, and Vicente Matellán. *Cybersecurity in Autonomous Systems: Evaluating the performance of hardening ROS* Proceedings of the XVII Workshop of Physical Agents, pp. 47-53. 16-17 Junio 2016. Málaga (Spain).

[4] Francisco Martín, José Mateos, Francisco Javier Rodríguez Lera, Pablo Bustos y Vicente Matellán, *A robotic platform for domestic applications* , XV Workshop of Physical Agents, 2014

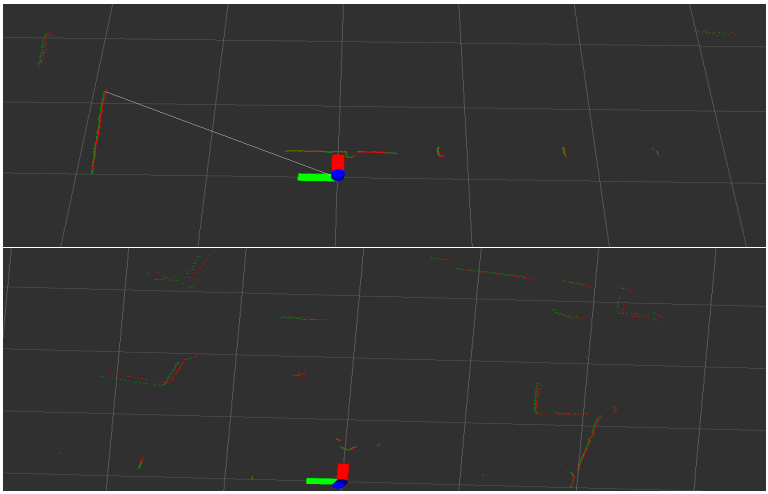

Fig. 7
VISUALIZACIÓN EN RVIZ DE UN SENSOR LÁSER (REPRESENTADO POR EL EJE DE COORDENADAS).

[5] Jarrod McClean, Christopher Stull, Charles Farrar, David Mascareñas. *A preliminary cyber-physical security assessment of the Robot Operating System (ROS)* SPIE Defense, Security, and Sensing. Vol 8741, 2013
[6] ROS Wiki http://wiki.ros.org/Robots Accedido por última vez 5 de mayo de 2016
[7] CSSP, D. (2009). *Recommended Proctice: Improving Industrial Control Systems Cybersecurity with Defense-In-Depth Strategies.* US-CERT Defense In Depth (Octubre 2009)
[8] Merkle, R. C., and Hellman, M. E. (1981). *On the security of multiple encryption..* Communications of the ACM, 24(7), 465-467.
[9] Elminaam, D. S. A., Abdual-Kader, H. M., Hadhoud, M. M. (2010). *Evaluating The Performance of Symmetric Encryption Algorithms.* IJ Network Security, 10(3), 216-222.
[10] Denning, T., Matuszek, C., Koscher, K., Smith, J. R., & Kohno, T. (2009). *A spotlight on security and privacy risks with future household robots.* Proceedings of the 11th International Conference on Ubiquitous Computing - Ubicomp '09, 105. doi:10.1145/1620545.1620564
[11] Hartmut Pohl (softScheck GmbH). (2016). *Robot Operating System (ROS): Safe & Insecure.* Automationspraxis, whitepaper. Disponible online [25/06/2016] en http://www.automationspraxis.de/whitepaper
[12] Morante, S., Victores, J. G., & Balaguer, C. (2015). *Cryptobotics: Why robots need cyber safety.* Frontiers in Robotics and AI, 2(23), 1–4. doi:10.3389/frobt.2015.00023
[13] Tadele, T. S. (2014, November 7). *Human-friendly robotic manipulators: safety and performance issues in controller design.* Tesis. Disponible en http://dx.doi.org/10.3990/1.9789036537841
[14] Nadeem, A., & Javed, M. Y. (2005, August). *A performance comparison of data encryption algorithms.* In 2005 International Conference on Information and Communication Technologies (pp. 84-89).

