A Study on ROS Vulnerabilities and Countermeasure

Se-Yeon Jeong, I-Ju Choi, Yeong-Jin Kim, Yong-Min Shin, Jeong-Hun Han
KITRI BoB(Best of the Best)
Seoul, Korea
best6653@gmail.com, eju94@naver.com

Goo-Hong Jung
GrayHash
Gyonggi, Korea
Cybermong@grayhash.com

Kyoung-Gon Kim
Korea University
Seoul, Korea
anesra@korea.ac.kr

ABSTRACT
ROS (Robot Operating System) is an open source software framework used for robot control. In this paper, we analyzed and tested 4 vulnerabilities related to ROS authentication scheme insufficiency, ROS Bag, communication vulnerability, and service hijacking. Then, We devised countermeasures against them.

1. INTRODUCTION
With the advent of the Hyper-connected Society and the emergence of the cyber-physics system, robots such as NAO of Aldebaran Robotics and Pepper of Softbank have come to reality. With CPS(Cyber Physical System), we can easily picture a society where humans and robots interact with each other in real time[1]. In order to make this happen, it is essential to not only operate a robot safety, but also secure the robot platform which is operating the robot. In this paper, we will be dealing with robot security in CPS security, especially focusing on the vulnerability of the robot platform. We have analyzed the vulnerability of the most representative robot platform — 'ROS (Robot Operating System)'[2] and have provided some countermeasures that are appropriate. Vulnerability analysis was conducted with KOBUKI of Eugene Robot, a research and development robot that supports ROS, along with a Turtlebot node similar to the actual environment.

2. VULNERABILITY ANALYSIS OF ROS

2.1 Broken authentication of ros
There is no authentication procedure for communication between master and node. Therefore, if the attacker matches only the environment variable ROS_MASTER_URI with the IP address and port number of the master, communication with all nodes connected to the master is possible, which can be exploited for various attacks. In this paper, we have performed in a private IP environment, but it can attack any number of URIs only in public IP environment.

The configuration environment for the ROS authentication scheme is ROS Indigo version, and Fig. 1 shows the Raspberry pi 2 environment running the master, and Fig. 2 shows the attacker's environment. As show in Fig. 1, After running the master, set the environment variable ROS_MASTER_URI on the attacker's PC to the IP address and port number of the target as shown in Fig. 2.

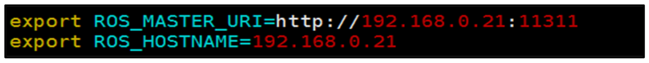

```
export ROS_MASTER_URI=http://192.168.0.21:11311
export ROS_HOSTNAME=192.168.0.21
```

Figure 1. Network configuration of Victim(Raspberry Pi 2) environment

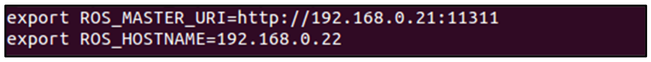

```
export ROS_MASTER_URI=http://192.168.0.21:11311
export ROS_HOSTNAME=192.168.0.22
```

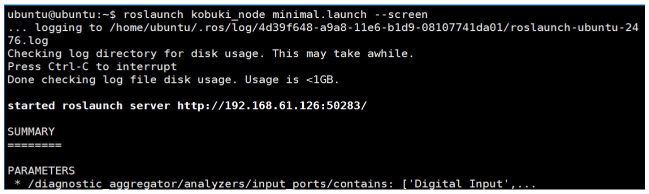
Figure 2. Network configuration of Attacker environment

This allows the attacker to access Raspberry Pi 2 environment without any further authentication. To test this, we use Turtlebot, a ROS simulator. First, run turtlesim_node in Raspberry Pi 2 environment to launch Turtlebot. Then run the turtle_teleop_key node in the attacker's environment to control Turtlebot.

HRI '17 Companion, March 06-09, 2017, Vienna, Austria A
CM 978-1-4503-4885-0/17/03.
http://dx.doi.org/10.1145/3029798.3038437

If attacker try keyboard input, it can be confirmed that the Turtlebot moves due to the key value inputted in the attacker's environment.

2.2 Ros bag replay attack
In the ROS, data messages exchanged between the master and the node, called a Bag, Bag file can be used to play this back when you need it and repeat the same action to the Robot. Thus, the attacker can play victim's robot behavior repeatedly by intercepting the message of victim's master and saving it as a Bag file using the ROS vulnerable authentication scheme. The configuration environment for 'ROS Bag Replay Attack' is ROS Indigo version. In this test, we connected Raspberry Pi with ROS Indigo version to KOBUKI main body. To connect Raspberry Pi to KOBUKI, run the node as shown in Fig. 4.


```
ubuntu@ubuntu:~$ roslaunch kobuki_node minimal.launch --screen
... logging to /home/ubuntu/.ros/log/4d39f648-a9a8-11e6-b1d9-08107741da01/roslaunch-ubuntu-24
76.log
Checking log directory for disk usage. This may take awhile.
Press Ctrl-C to interrupt
Done checking log file disk usage. Usage is <1GB.

started roslaunch server http://192.168.61.126:50283/

SUMMARY
========

PARAMETERS
 * /diagnostic_aggregator/analyzers/input_ports/contains: ['Digital Input',...
```

Figure 4. Running kobuki_node

As shown in Fig. 5, Kobuki: initialised log is displayed, and KOBUKI and Raspberry Pi are connected via ROS.

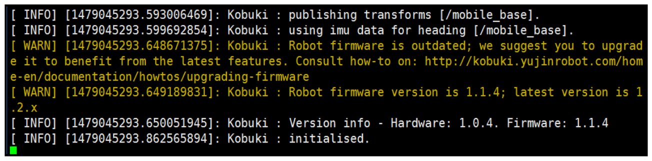

```
[ INFO] [1479045293.593006469]: Kobuki : publishing transforms [/mobile_base].
[ INFO] [1479045293.599692854]: Kobuki : using imu data for heading [/mobile_base].
[ WARN] [1479045293.648671375]: Kobuki : Robot firmware is outdated; we suggest you to upgrad
e it to benefit from the latest features. Consult how-to on: http://kobuki.yujinrobot.com/hom
e-en/documentation/howtos/upgrading-firmware
[ WARN] [1479045293.649189831]: Kobuki : Robot firmware version is 1.1.4; latest version is 1
.2.x
[ INFO] [1479045293.650051945]: Kobuki : Version info - Hardware: 1.0.4. Firmware: 1.1.4
[ INFO] [1479045293.862565894]: Kobuki : initialised.
```

Figure 5. Completion of connection with KOBUKI and Raspberry

Run the keyop package in the victim's environment to control KOBUKI with the keyboard. When the connection is completed, the KeyOp: connected log is displayed as shown in Fig. 6, and from this point, the KOBUKI can be controlled through the keyboard.

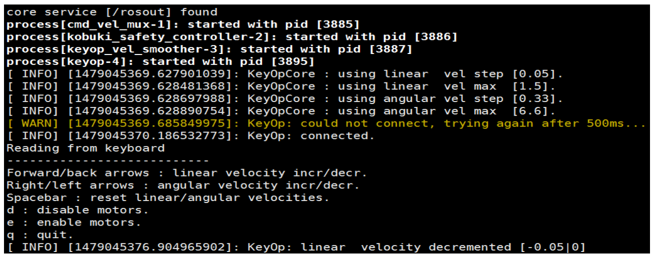
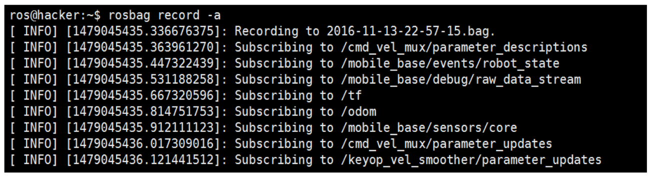

```
core service [/rosout] found
process[cmd_vel_mux-1]: started with pid [3885]
process[kobuki_safety_controller-2]: started with pid [3886]
process[keyop_vel_smoother-3]: started with pid [3887]
process[keyop-4]: started with pid [3895]
[ INFO] [1479045369.627901039]: KeyOpCore : using linear  vel step [0.05].
[ INFO] [1479045369.628481368]: KeyOpCore : using linear  vel max  [1.5].
[ INFO] [1479045369.628697988]: KeyOpCore : using angular vel step [0.33].
[ INFO] [1479045369.628890754]: KeyOpCore : using angular vel max  [6.6].
[ WARN] [1479045369.685849975]: KeyOp: could not connect, trying again after 500ms...
[ INFO] [1479045370.186532773]: KeyOp: connected.
Reading from keyboard
---------------------------
Forward/back arrows : linear velocity incr/decr.
Right/left arrows : angular velocity incr/decr.
Spacebar : reset linear/angular velocities.
d : disable motors.
e : enable motors.
q : quit.
[ INFO] [1479045376.904965902]: KeyOp: linear  velocity decremented [-0.05|0]
```

Figure 6. Running kobuki_keyop node

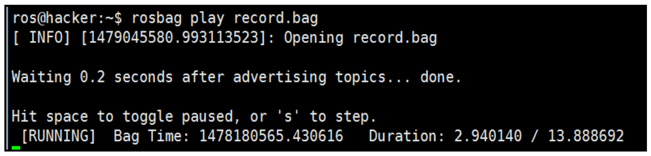
The attacker can use the rosbag utility to write keyboard input to a Bag file that controls KOBUKI. At this time, KOBUKI and the attacker's environment are different devices, and the attacker must set the environment variable ROS_MASTER_URI to the victim's IP and HOST_NAME to the attacker's IP. The command used to record values with Rosbag is rosbag record [option] [topic name]. This command allows you to record a message by specifying a specific topic[3]. In this paper, we record the keyboard input values of KOBUKI in the attacker's environment using the -a option to record all topics as shown in Fig. 7.



```
ros@hacker:~$ rosbag record -a
[ INFO] [1479045435.336676375]: Recording to 2016-11-13-22-57-15.bag.
[ INFO] [1479045435.363961270]: Subscribing to /cmd_vel_mux/parameter_descriptions
[ INFO] [1479045435.447322439]: Subscribing to /mobile_base/events/robot_state
[ INFO] [1479045435.531188258]: Subscribing to /mobile_base/debug/raw_data_stream
[ INFO] [1479045435.667320596]: Subscribing to /tf
[ INFO] [1479045435.814751753]: Subscribing to /odom
[ INFO] [1479045435.912111123]: Subscribing to /mobile_base/sensors/core
[ INFO] [1479045436.017309016]: Subscribing to /cmd_vel_mux/parameter_updates
[ INFO] [1479045436.121441512]: Subscribing to /keyop_vel_smoother/parameter_updates
```

Figure 7. KOBUKI's move record

As the attacker is recording, the victim moves KOBUKI. The movement is recorded in the Bag file in the attacker's environment. As shown in Fig. 8, when the Bag file recorded in the attacker's environment is reproduced, it can be seen that the movement of KOBUKI is reproduced in the victim's environment.



```
ros@hacker:~$ rosbag play record.bag
[ INFO] [1479045580.993113523]: Opening record.bag

Waiting 0.2 seconds after advertising topics... done.

Hit space to toggle paused, or 's' to step.
 [RUNNING]  Bag Time: 1478180565.430616   Duration: 2.940140 / 13.888692
```

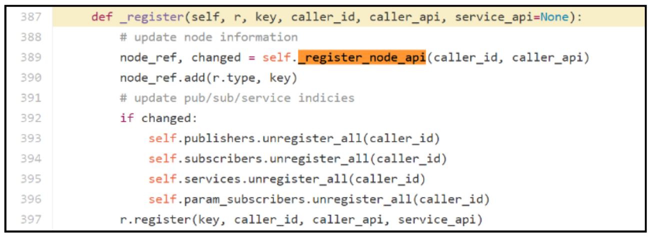
Figure 8. Run recorded Bag

2.3 Vulnerability of ros communication

In the ROS environment, XMLRPC communication is basically performed. However, since no separate encryption processing is performed, the communication history is directly exposed to the attacker. Therefore, if an attacker catches a packet in the middle, the robot can be controlled by manipulating the communication details freely from the remote.

2.4 Service hijacking

The Master does not have redundancy check logic for service requests of the same name. Therefore, if an attacker arbitrarily registers a service with the same name, it can intercept the control of the previous service. When the service is registered to the master, the register_service method is called through the reg_manager (class RegistrationManager) object. This method internally calls the _register method. As shown in Fig. 11, _register calls the _register_node_api method internally. It determines whether to register the publisher, subscriber, or service according to the returned changed variable at this time.



```
    def _register(self, r, key, caller_id, caller_api, service_api=None):
        # update node information
        node_ref, changed = self._register_node_api(caller_id, caller_api)
        node_ref.add(r.type, key)
        # update pub/sub/service indicies
        if changed:
            self.publishers.unregister_all(caller_id)
            self.subscribers.unregister_all(caller_id)
            self.services.unregister_all(caller_id)
            self.param_subscribers.unregister_all(caller_id)
        r.register(key, caller_id, caller_api, service_api)
```

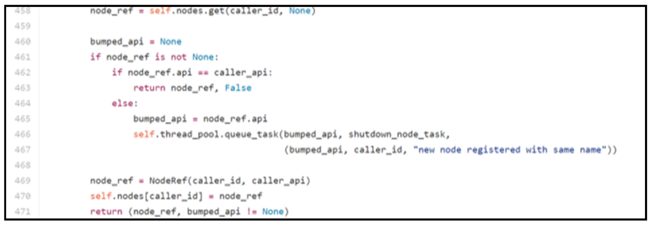
Figure 11. Call register_node_api (registrations.py)

The variable names and their respective values to look at in the _register_node_api method are the same as Table 1.

Table 1. Parameters of register_node_api

Variable	Description
Caller_id	ROS node names such as / keyop, / cmd_vel_mux
Caller_api	XML-RPC URI of the registration request node

In Fig. 12, the existing node compare caller_api in which is operating and caller_api in which the new request is received. If the two APIs are different, such as when a request comes in from another device, the value is set to bumped_api, which is set to 'None', and the string "new node registered with same name" is appended. After that, it returns bool type whether bumped_api is modulated together with the corresponding node information. If the two APIs are different, such as when a request comes from a different device, the value is set to bumped_api, which is set to 'None', and the string "new node registered with same name" is appended. After that, it returns bool type whether bumped_api is modulated together with the corresponding node information.



```
        node_ref = self.nodes.get(caller_id, None)

        bumped_api = None
        if node_ref is not None:
            if node_ref.api == caller_api:
                return node_ref, False
            else:
                bumped_api = node_ref.api
                self.thread_pool.queue_task(bumped_api, shutdown_node_task,
                                            (bumped_api, caller_id, "new node registered with same name"))

        node_ref = NodeRef(caller_id, caller_api)
        self.nodes[caller_id] = node_ref
        return (node_ref, bumped_api != None)
```

Figure 12. register_node_api (registrations.py)

In Fig. 11, if bumped_api is not 'None', changed variable is set to 'True' and it enters into if condition statement. After that, all the services corresponding to the node are unregistered and registered with the requested new service. Except for the comparison via Caller_api, the request service name is not checked for duplication. Therefore, when a service of the same name is requested from a remote location, the existing service is terminated and the new service which is made by remote attacker can be registered.

3. COUNTERMEASURE

In the previous chapters 2, when we looked up at the ROS vulnerability, it was analyzed that most of the ROS vulnerabilities occurred because there was no authentication procedure when connecting to the master node. Therefore, it is necessary to strengthen these authentication procedures for security. In this paper, we propose three countermeasures.

3.1 Introducing user request logic

Currently, there is no authentication procedure for ROS network in constructing distributed environment. Therefore, it is necessary to introduce a user request system in the process of communicating with the first master. A user who wishes to access the master first requests a connection to the master and makes the access only when the master's approval is made.

3.2 Allow access only to authorized mac

When the access of the first user is approved, the master registers the MAC address of the user who accesses it. In the subsequent communication process, it is checked whether the user is authorized based on the registered MAC address.

3.3 Introduction of encryption logic in communication process

Currently, communication between nodes is made in plain text because there is no encryption process. Therefore, considering that the inter-node communication is a one-to-one method, share the encryption key through 'Diffie-Hellman Key Exchange' in the communication process and add logic to communicate by AES-256 encryption process.

4. CONCLUSION

Robot hacking and the resulting crime, damage was a thing that existed only in the fantasy of people for a long time. However, as the robots become closer to our lives, robotic hacking has become closer to our daily lives. ROS is currently the most well-known promising robot operating system, so the importance of security is emphasized. Robot security is essential when considering the damage that can be caused. It is expected that a more secure robot environment can be constructed if a series of contents are known to robot developers who are the main target audience of this paper. It is expected that further research will enhance the security of robots with ROS and various operating systems, along with the enhancement of comprehensive CPS security.

5. REFERENCES

[1] Baheti, R. and Gill, H. 2011. *Cyber-physical Systems*, The impact of control technology.

[2] ROS, http://www.ros.org

[3] Quigley, M., Gerkey, B., and Conley, K. 2009. *ROS: an open-source Robot Operating System*, ICRA workshop on.